INFless

A Native Serverless System for Low-Latency , High-Throughput Inference

ASPLOS'22

Yanan Yang,Laiping Zhao,Yiming Li,Huanyu Zhang,Jie Li,Mingyang Zhao,Xingzhen Chen,Keqiu Li
Tianjin University, TANKLAB

Introduction

Serverless computing (Inference)

- Advantages 👍
  - resource management-free, auto-scaling, cost efficiency
- Often used into ML inferences with **strict latency requirements**
- Disadvantages 👎
  - **can't guarantee latency** (SLO on latency)
  - **low resource efficiency** (CPU only)

Inference Latency	Fraction of Models (%)
<50ms	86.2
50-200ms	11.6
200-500ms	1.1
500-1000ms	0.6
>1000ms	0.3

(d) Latency SLO distribution

→ **How to balance ?**

Introduction

AWS Lambda

Limitation of existing serverless platforms — 5 Observations

Observation #1: High latency — lack accelerators' support for large models

- **CPU's memory size** can't fix latency challenge **all the time** (128M~3072M)
  - small-sized model (<50ms)
  - large-sized model (>200ms)

OTP -> Native

Observation #2: High latency — poor OTP batch for small models

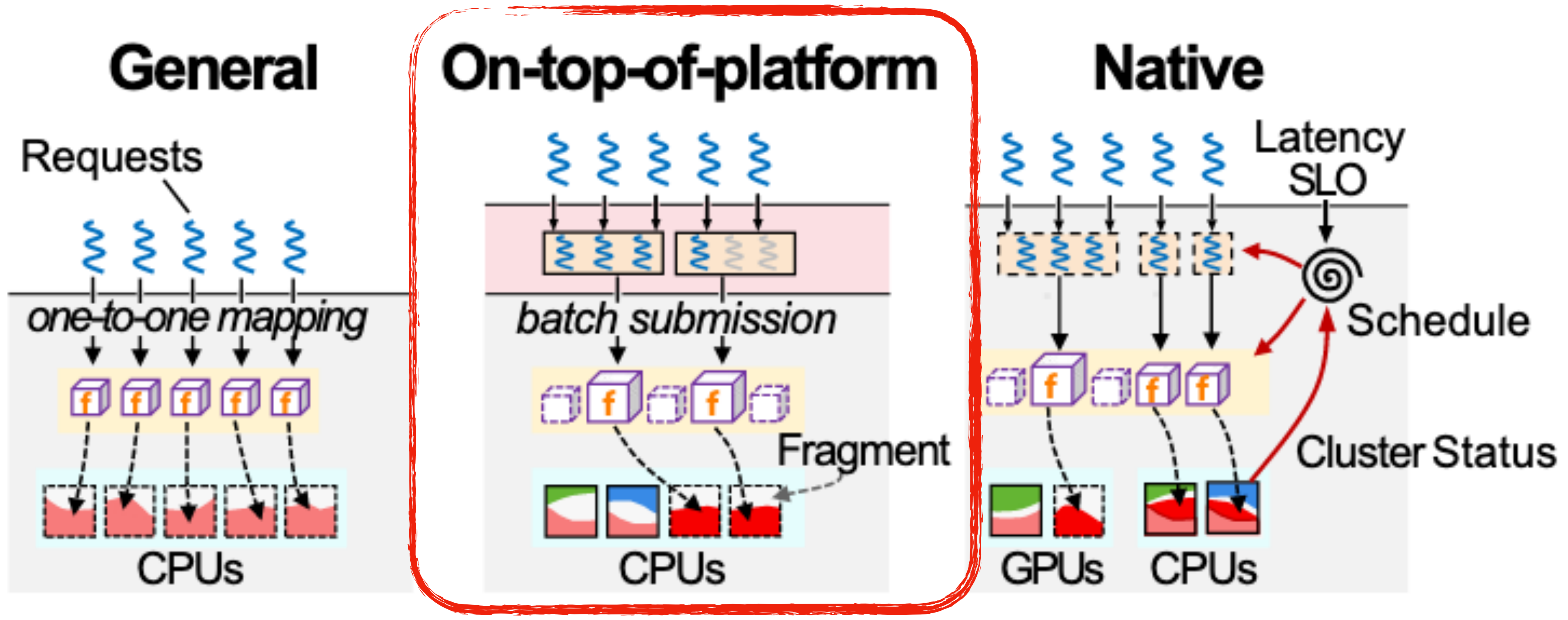
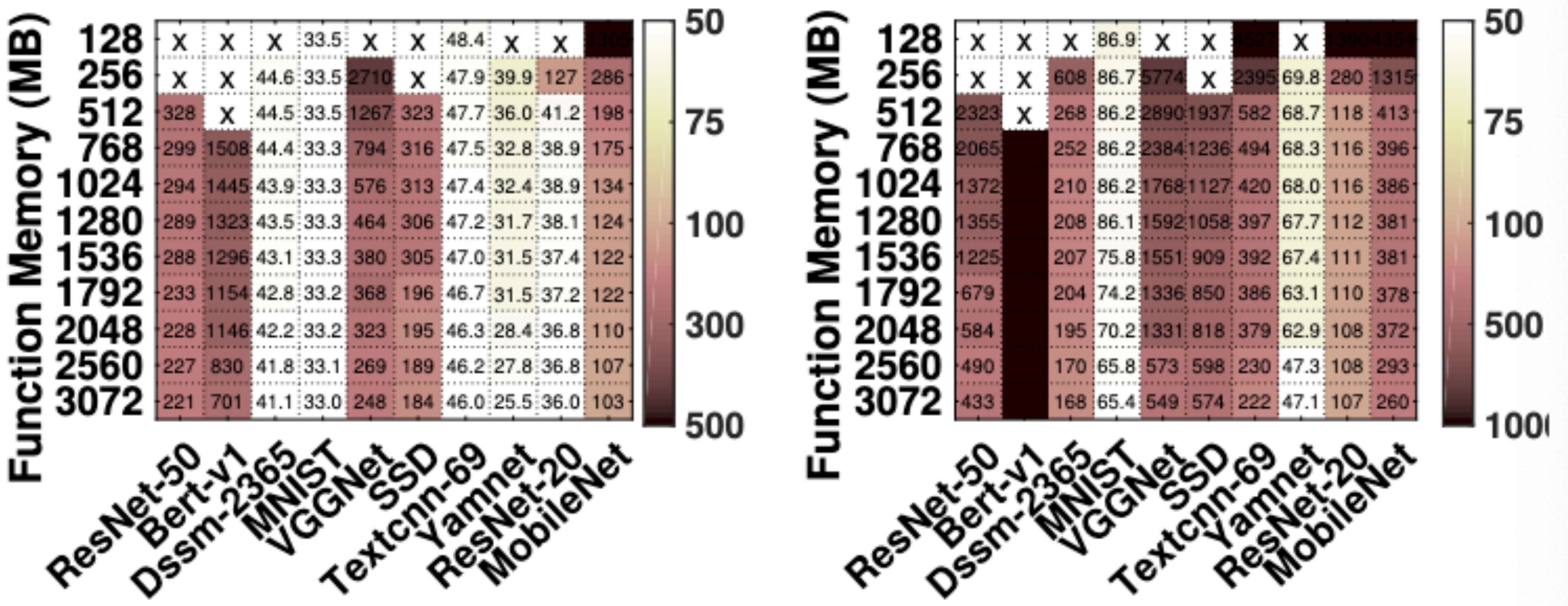


Figure 1: Schematic overview of serverless inference systems.



(a) Latency under no-batching (b) Latency under batching

Introduction

AWS Lambda

Limitation of existing serverless platforms — 5 Observations

Observation #3: Resource over-provisioning — more memory than CPU need

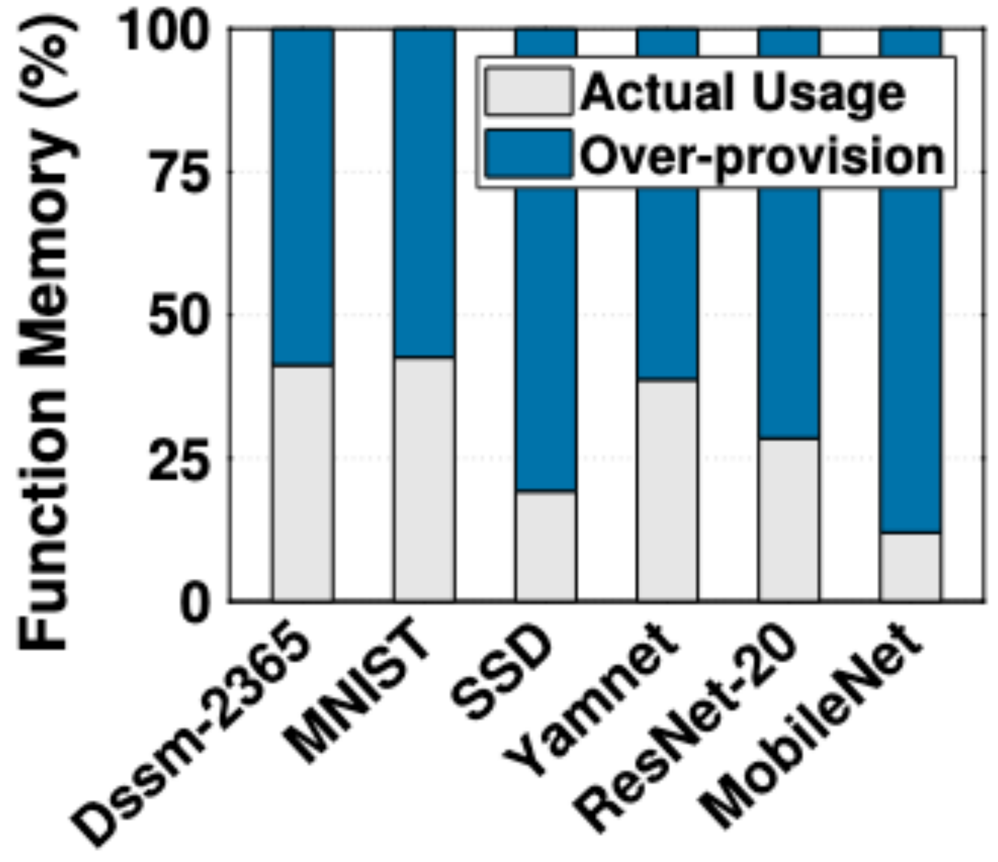
- Commercial serverless provider **allocate CPU power in proportion to memory**
  - most inference models are **computationally-intensive**

Observation #4: Resource over-provisioning — "one-to-one mapping" policy

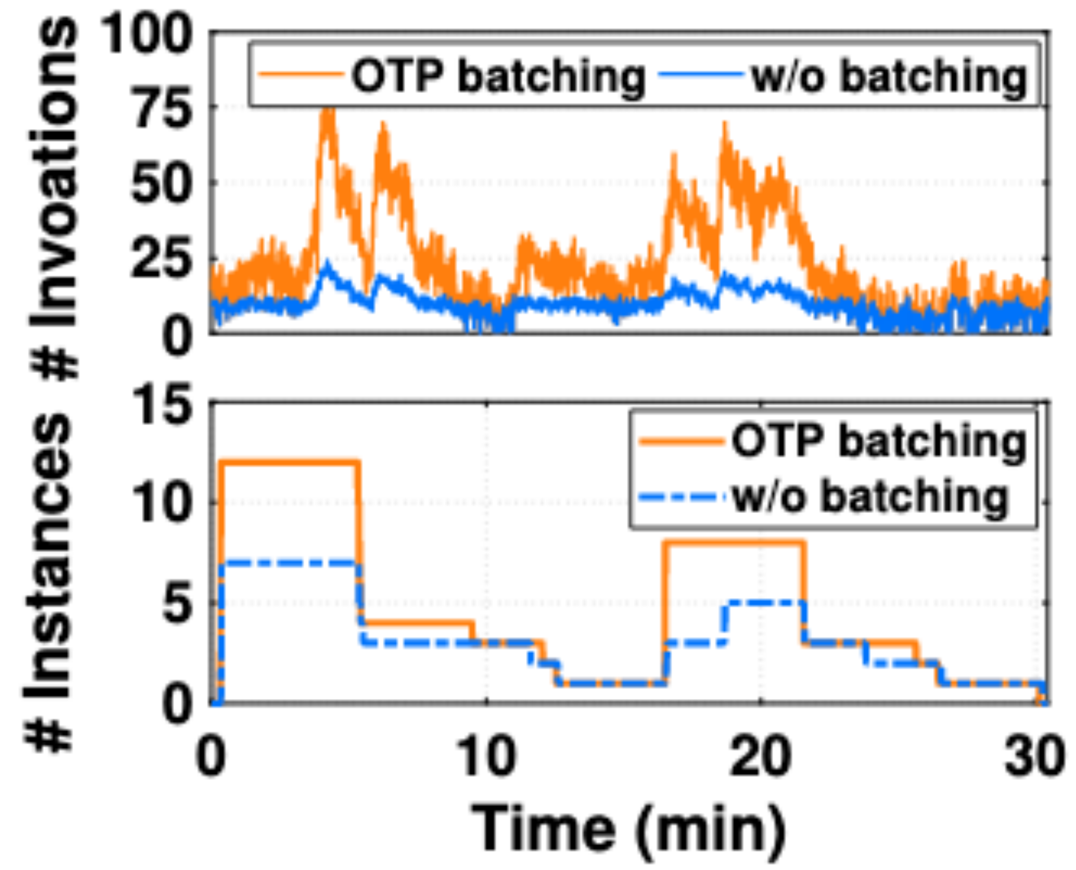
- each inference request is dispatched to a separate instance cause **low resource utilization**
  - **too much instances** are created

↓

Batching



(c) Function memory over-provisioning



(a) Excessive function scaling

AWS Lambda

Introduction

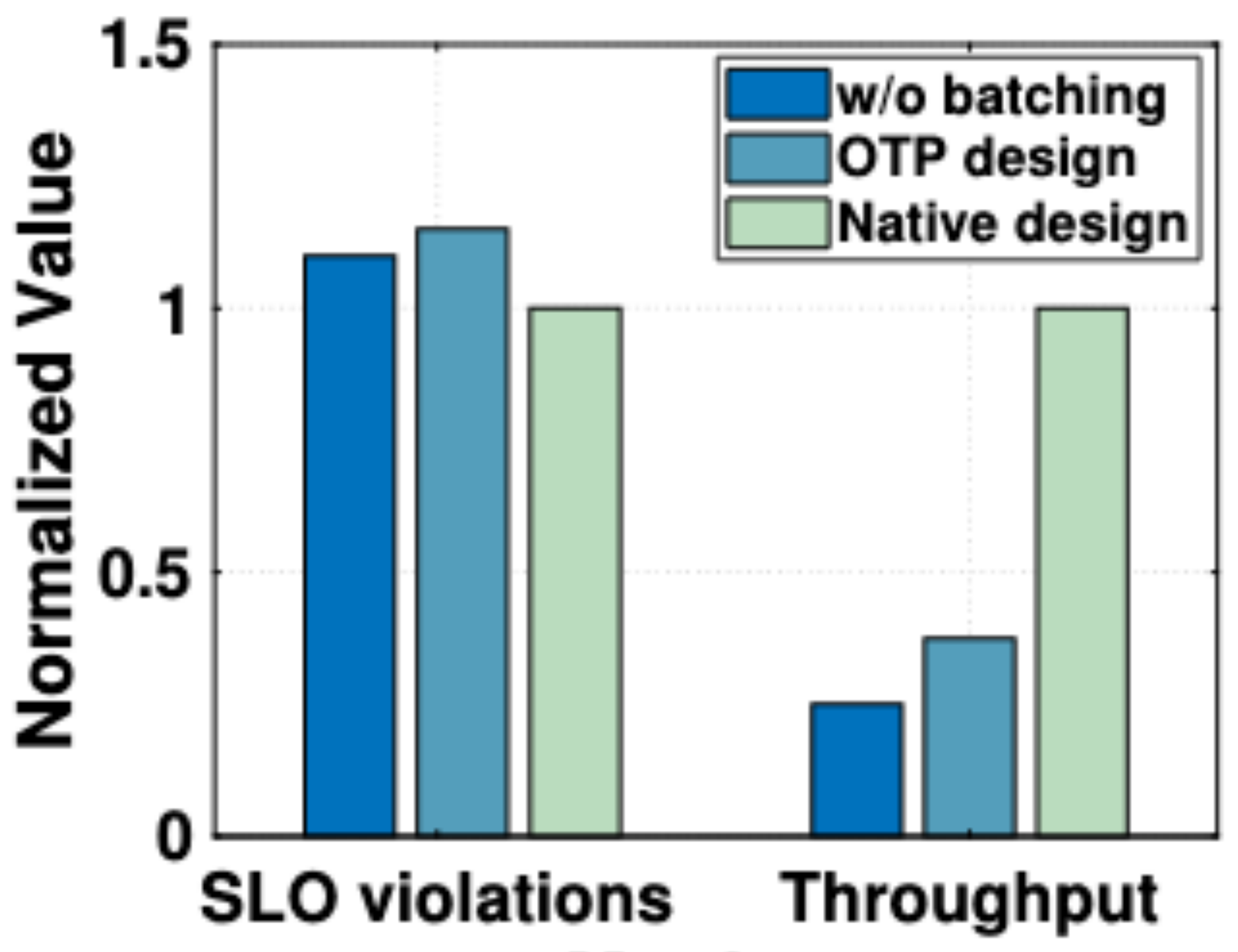
Limitation of existing serverless platforms — 5 Observations

Observation #5: OTP batching — limited throughput improvement

- **lack the codesign** of batch configuration, instance scheduling and resource allocation
  - OTP **batch requests before submit** to platforms
- disadvantage
  - need another dedicated server for batching
  - unaware of schedule inside platform
  - uniform scaling policy

↓

Codesigned Batching & Optimization



(b) OTP batching v.s. INFless

Implications & Challenges

A novel, native serverless inference system

- Need to address the observations below:
  - support **hybrid CPU/accelerators** (1,2)
  - produce **resource-efficient schedule** (3)
  - support **built-in batching** (4,5)
- **Challenges**
  - The hardware complicate the **management of hybrid CPU/accelerators**
  - **Batchsize and resource selection enlarge the search space of scheduling decisions**
  - The running **overhead should be low** to make the system practicable

INFless' design

System architecture

$$l = t_{cold} + t_{batch} + t_{exec}$$

- **Basic idea** : exploit the **features and characteristics** of inference (**shared operators, batching and computation intensive**) to optimize system performance

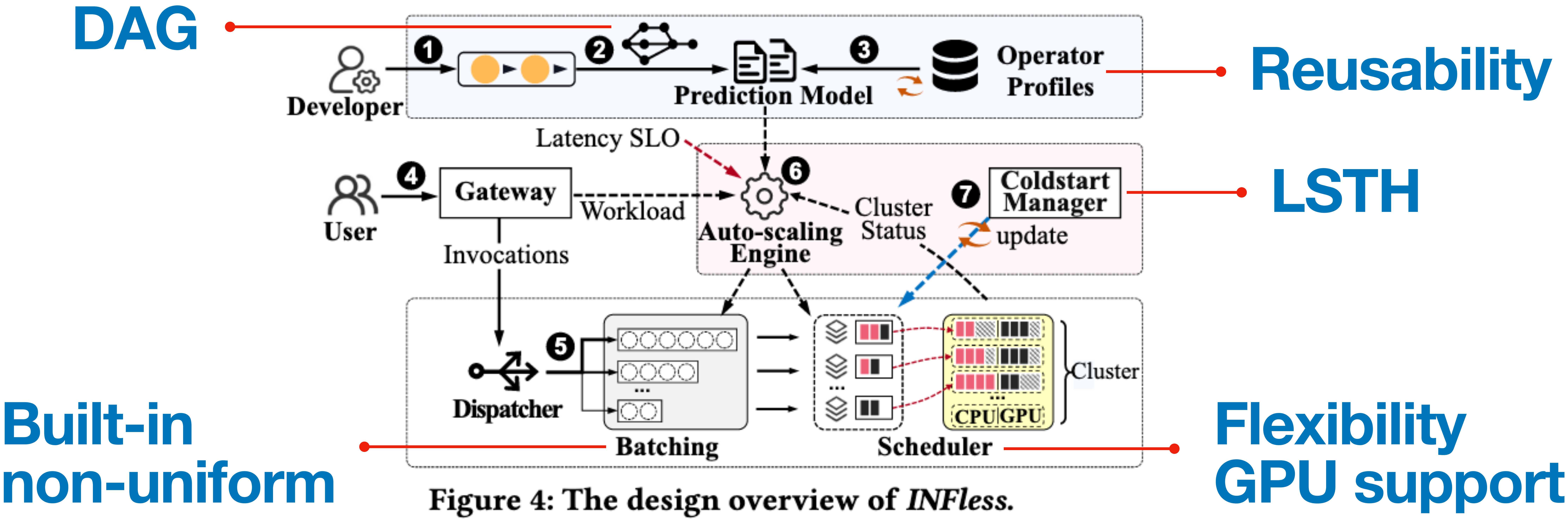


Figure 4: The design overview of *INFless*.

INFless — Built-in, Non-Uniform Batching

Batch method

- **Built-in**
  - native
  - **simultaneous, collaborative control** over batchsize, resource allocation and placements
- **Non-uniform**
  - **Each instance has an individual batch queue to aggregate inference requests**

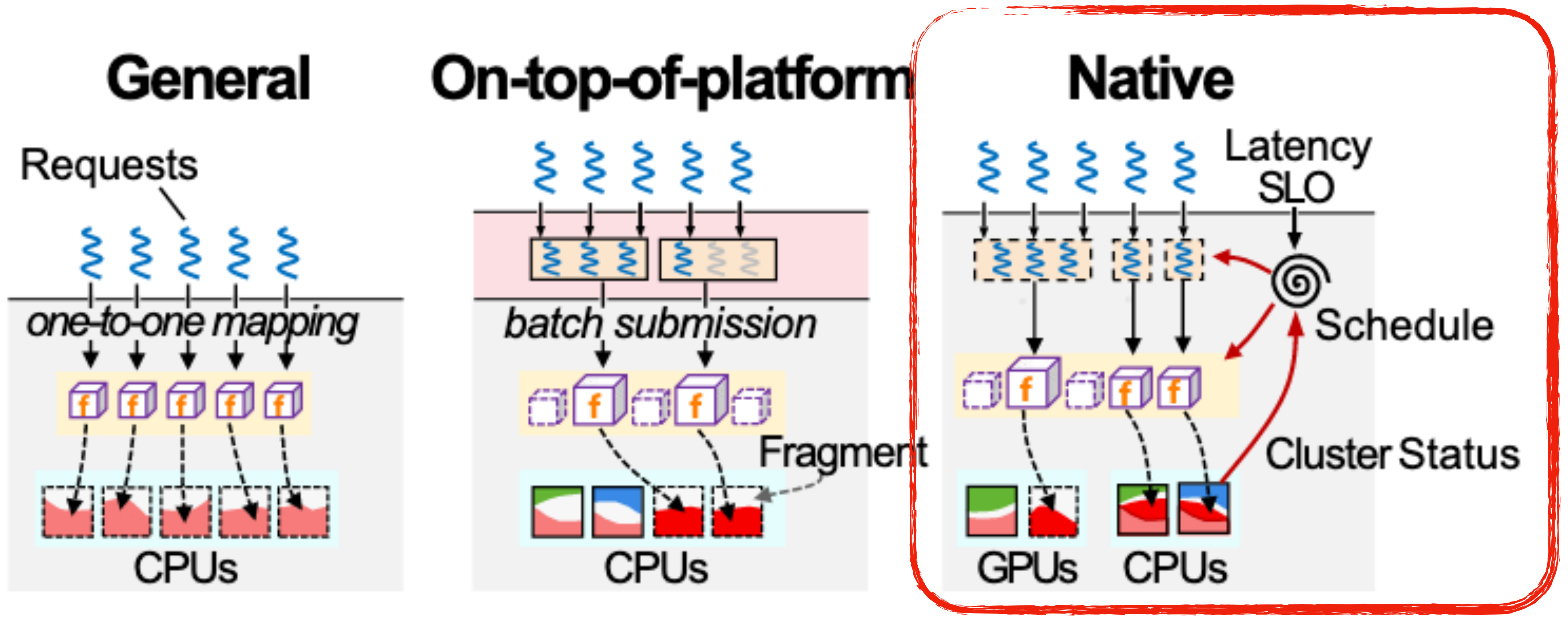


Figure 1: Schematic overview of serverless inference systems.

improve upon
utilization of resources & throughput

INFless — Built-in, Non-Uniform Batching

Batch method — RPS

- To **guarantee the latency SLO without dropping any requests**, the request arrival rate (Request per second — RPS) toward each instance is strictly kept within the range

$$l \leq t_{slo}$$

$$[r_{low}, r_{up}] \longrightarrow r_{up} = \lfloor \frac{1}{t_{exec}} \rfloor \times b, \quad r_{low} = \lceil \frac{1}{t_{slo} - t_{exec}} \rceil \times b$$

$$t_{exec} \leq t_{slo}/2$$

$$R_{max} = \sum_{i \in [1,..,n]} r^i_{up} \text{ and } R_{min} = \sum_{i \in [1,..,n]} r^i_{low}$$

instances for an function

$$R > R_{max}$$

$$\alpha R_{min} + (1 - \alpha) R_{max} \leq R \leq R_{max}$$

$$R < \alpha R_{min} + (1 - \alpha) R_{max}$$

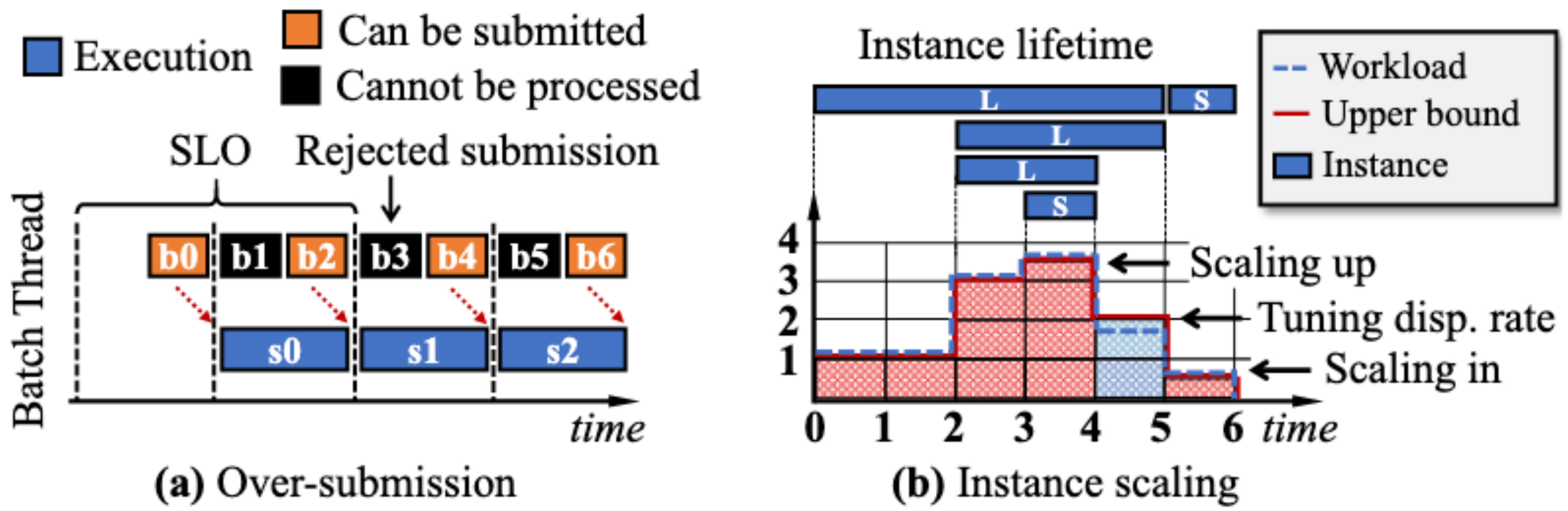


Figure 6: (a) The over-submission of requests lead to request drop. (b) An example of the instance scaling procedure, and L (S) represents function instance configured with large (small) batchsize, respectively.

INFless — Combined Operator Profiling

A lightweight combined operator profiling method — COP

- INFless **improve throughput** by **deploying as many instance as possible with limited resources**

insufficient resource allocation → **High latency**

↓ fix via **latency prediction**

High throughput
Low SLO violations

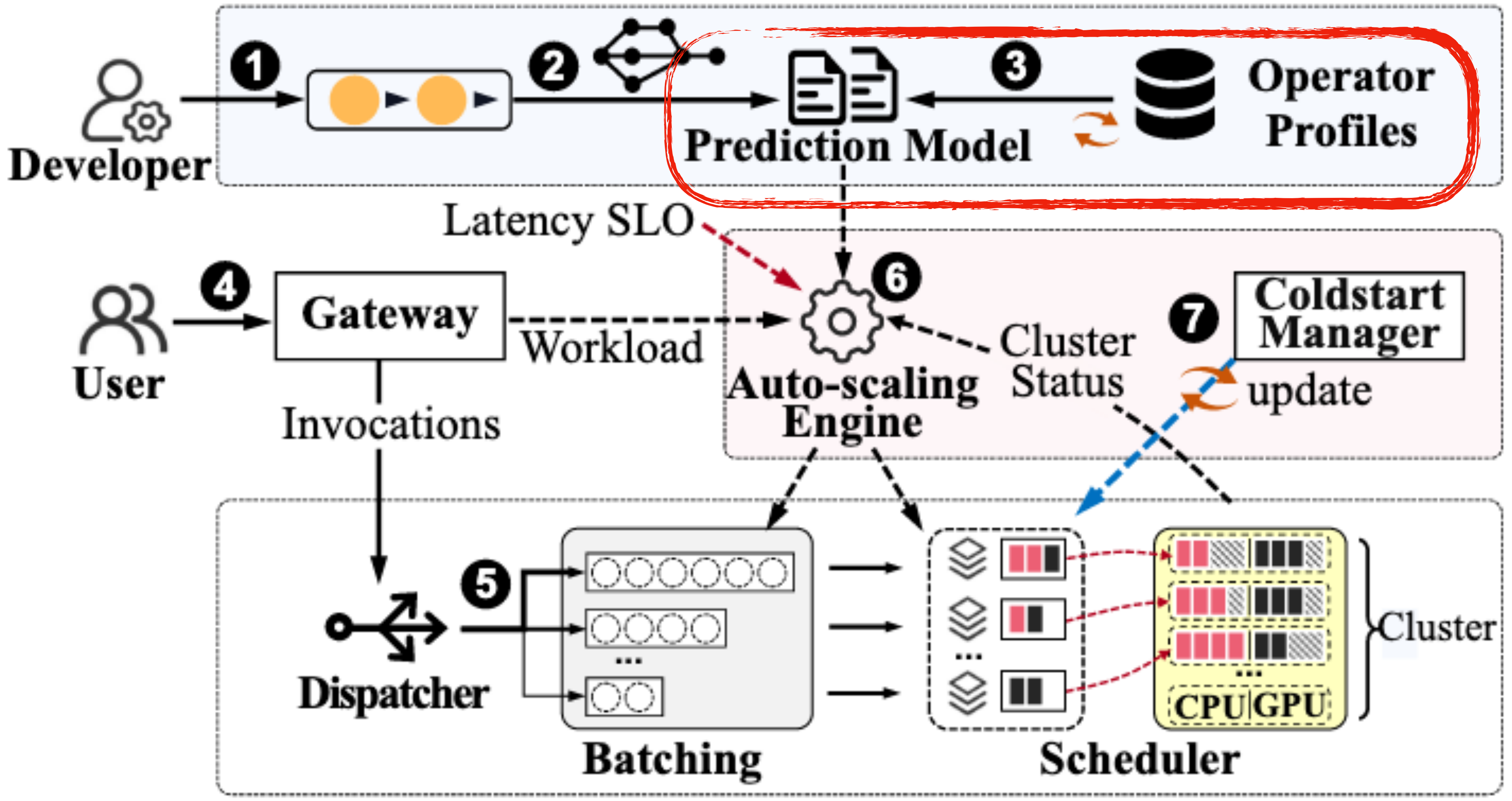


Figure 4: The design overview of *INFless.*

INFless — Combined Operator Profiling

A lightweight combined operator profiling method — COP

Observation #6: Inference functions share common ops and t_{exec} is dominated by a small subset

$$o_i = \langle p_i, b_i, c_i, g_i, t_i \rangle$$

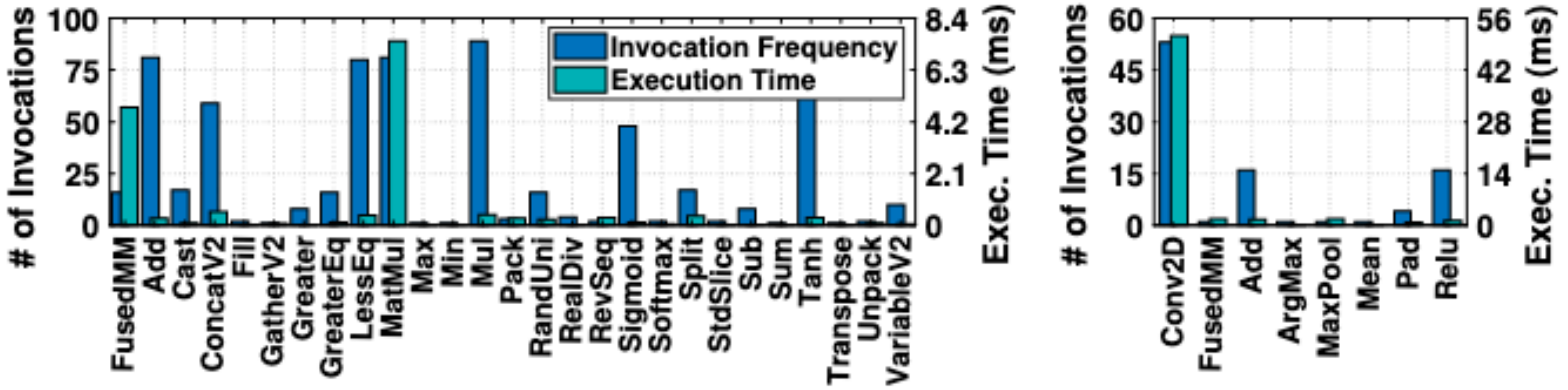
$$b_i \in \{2^0, 2^1, \dots, 2^{max}\}$$

$$G = (O, E)$$

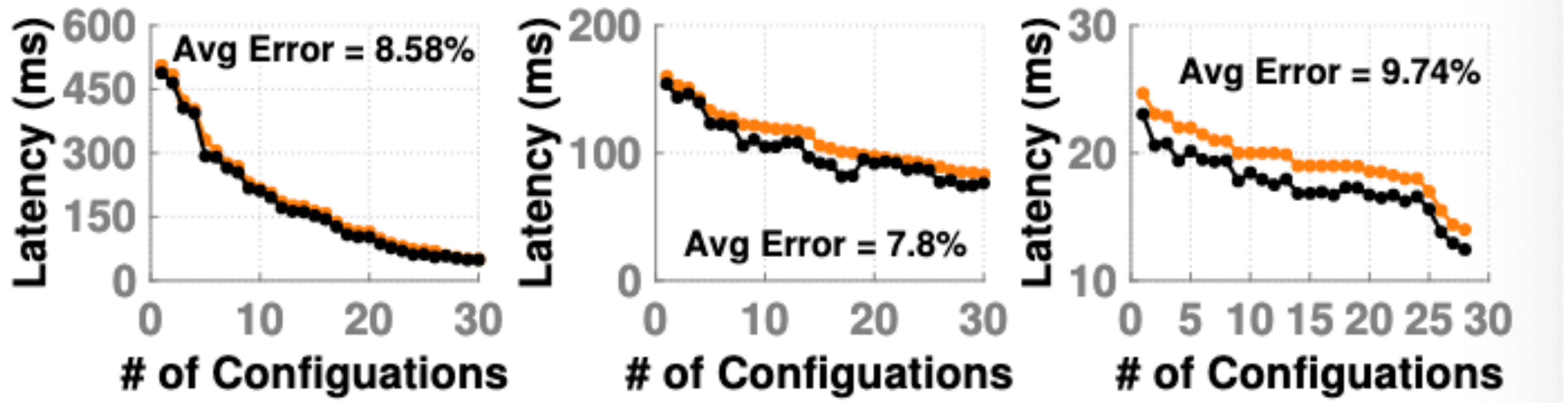
$$t_{chain} = \sum_{i \in \{chain\}} t_i$$

$$t_{branches} = \max_{i \in \{branches\}} t_i$$

DAG



(a) LSTM-maxclass (b) ResNet-50



(a) ResNet-50 (b) MobileNet (c) LSTM-2365

Figure 8: The prediction results of operator combination model across different batch-resource configurations.

INFless — Scheduling

Auto-scaling engine's aim

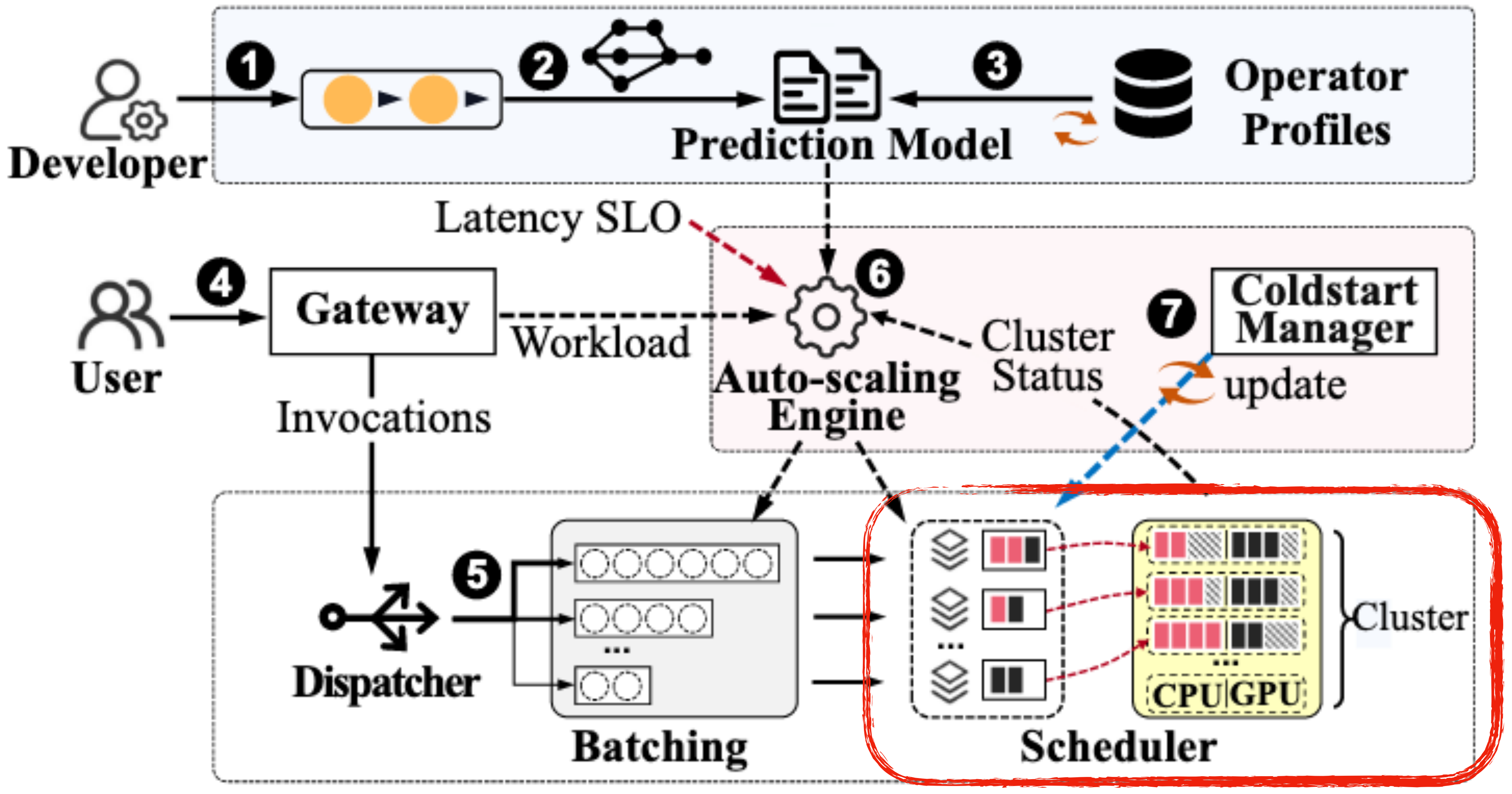


Figure 4: The design overview of *INFless*.

$$r_{up} = \lfloor \frac{1}{t_{exec}} \rfloor \times b, \quad r_{low} = \lceil \frac{1}{t_{slo} - t_{exec}} \rceil \times b \qquad (1)$$

$$minimize: \sum_j (\beta C_j + G_j) y_j \qquad (2)$$

$$t^i_{wait} + t^i_{exec} \le t^i_{slo}, \ \forall i \in [1, .., n] \qquad (3)$$ — instances

$$t^i_{exec} \le t^i_{wait}, \ \forall i \in [1, .., n] \qquad (4)$$

$$\sum_i^n c_i x_{ij} \le C_j y_j, \ \forall j \in [1, .., m] \qquad (5)$$ — servers

$$\sum_i^n g_i x_{ij} \le G_j y_j, \ \forall j \in [1, .., m] \qquad (6)$$

$$\alpha R^k_{max} + (1-\alpha) R^k_{min} \le R_k \le R^k_{max}, \ \forall k \in I \qquad (7)$$

$$x_{ij} \in \{0, 1\}, \ y_j \in \{0, 1\} \qquad (8)$$

$$b_i, c_i \in Z_+, \ g_i \in Z \qquad (9)$$

Instance config : b_i , c_i , g_i **schedule** : x_{ij} (0/1)

Server schedule : y_j (0/1)

INFless — Scheduling

Auto-scaling engine's algorithem

- **Batchsize** is the key components that **contributes most to throughput**

1.decide Instance's config

2.where to launch (Instance's schedule)

meet **SLO** & no **time out**

$$e_{ij} = \frac{RPS/resource}{fragmentation} = \frac{r_{up}/(\beta c_i + g_i)}{1 - (\beta c_i + g_i)/(\beta C_j + G_j)} \quad (10)$$

Resource efficiency metric

Algorithm 1: Schedule(R_k, $\boldsymbol{B}$, $\boldsymbol{M}$, t_{slo})

Input:
R_k ▹ The residual RPS towards the function k;
$\boldsymbol{B}$ ▹ The batchsize set of k, sorted in the descending order;
$\boldsymbol{M}$ ▹ The available resource capacities for the m-server cluster
t_{slo} ▹ The latency SLO for function k;
Output:
n_k ▹ The number of instances for function k;
b_i, c_i, g_i ▹ The batchsize/CPU/GPU configs. of instances;
x_{ij} ▹ the placement of each instance;

$n_k \leftarrow 0, x_{ij} \leftarrow 0$;
while $R_k > 0$ **do**
  for $b \in \mathbf{B}$ **do**
    $I_b \leftarrow$ `AvailableConfig`(b, R_k, t_{slo})
    /* *e.g.*, $\mathbf{I}_b = \{\langle b, c_1, g_1 \rangle, \cdots, \langle b, c_n, g_n \rangle\}$ */
    if $I_b = NULL$ **then**
      continue; `// try next largest batchsize`
    for $\forall \langle b, c_i, g_i \rangle \in \mathbf{I}_b, \forall j \in \mathbf{M}$ **do**
      Derive the res. efficiency e_{ij} using Equation 10;
    if $\{e_{ij}\} \neq NULL$ **then**
      $i, j \leftarrow argmax\{e_{ij}\}, \forall i \in [1..n], j \in [1..m]$;
      $n_k \leftarrow n_k + 1, x_{ij} \leftarrow 1$; `// schedule` i `to` j
      $\langle C_j, G_j \rangle \leftarrow \langle C_j, G_j \rangle - \langle c_i, g_i \rangle$;
      $R_k \leftarrow R_k - r_{up}$;`// update` R_k
      break; `// scaling for the rest of` R_k

Function `AvailableConfig`(b, R_k, t_{slo}):
  $I_b \leftarrow NULL$;
  for $\forall \langle b, c_i, g_i \rangle \in all\_configurations$ **do**
    $t_{exec} \leftarrow f(b, c_i, g_i)$;`// predict the` t_{exec}
    if $b = 1$ **then**
      if $t_{exec} \leq t_{slo}$ **then**
        $I_b \leftarrow I_b \cup \{\langle b, t_i, g_i \rangle\}$;
    else
      Derive the r_{up} and r_{low} using Equation 1;
      if $t_{exec} \leq t_{slo}/2 \wedge R_k \geq r_{low}$ **then**
        $I_b \leftarrow I_b \cup \boxed{\{\langle b, c_i, g_i \rangle\}}$
  return I_b

INFless — Manage Cold Start with LSTH

Long-Short Term Histogram – LSTH

- **Cold starts cause significant performance degradation for serverless functions**
- state-of-art approach is **hybrid histogram policy (HHP)** 👎
  - **pre-warming** window & **keep-alive** window
  - **too conservative** ⟶ **too much resource waste**
- workload's feature
  - **Long-term periodicity (LTP)**
  - **Short-term burst (STB)**

$$pre-warm = \gamma L_{prewarm} + (1-\gamma) S_{prewarm}$$

$$keep-alive = \gamma L_{keepalive} + (1-\gamma) S_{keepalive}$$

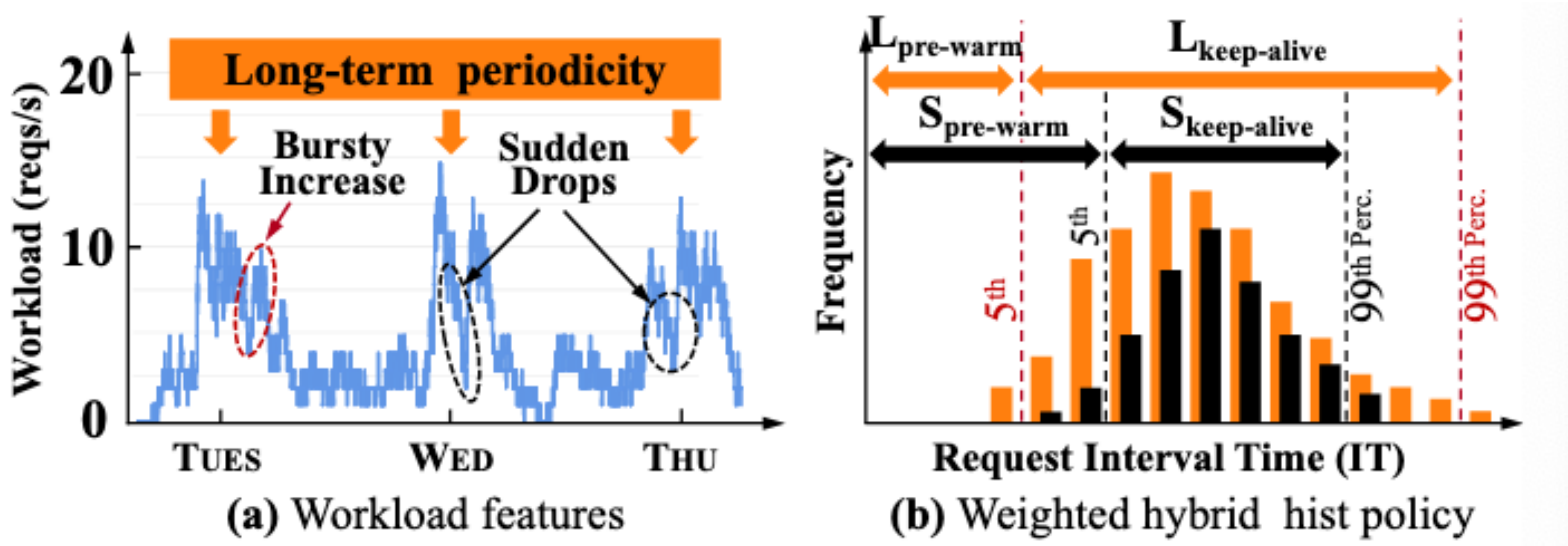


Figure 9: (a) The long-term periodicity and short-term burst behaviors of inference workloads. (b) We propose the weighted hybrid hist policy by charactering both long-term and short-term workload patterns.

Evaluation

Setup & Workload

Table 2: Experimental testbed configuration.

Component	Specification	Component	Specification
CPU device	Intel Xeon Silver-4215	Shared LLC Size	11MB
Number of sockets	2	Memory Capacity	128GB
Processor BaseFreq.	2.50 GHz	Operating System	Ubuntu 16.04
CPU Threads	32 (16 physical cores)	SSD Capacity	960GB
GPU device	Nvidia RTX 2080Ti	GPU Memory Config	11GB DGDDR6
GPU SM cores	4352	Number of GPUs	16

Workload

Table 1: ML inference models collected from the MLPerf benchmark and real-world commercial services.

ML Model	Network Size	GFLOPs	Description & Source
Bert-v1	391M	22.2	Language processing[12]
ResNet-50	98M	3.89	Image classification[23]
VGGNet	69M	5.55	Feature localisation[33]
LSTM-2365	39M	0.10	Text Q&A system[9]
ResNet-20	36M	1.55	Image classification[23]
SSD	29M	2.02	Object detection[8]
DSSM-2365	25M	0.13	Text Q&A system[2]
YamNet	17M	1.60	Speech recognition[27]
MobileNet	17M	0.05	Mobile network[42]
TextCNN-69	11M	0.53	Text classification[7]
MNIST	72k	0.01	Number recognition[18]

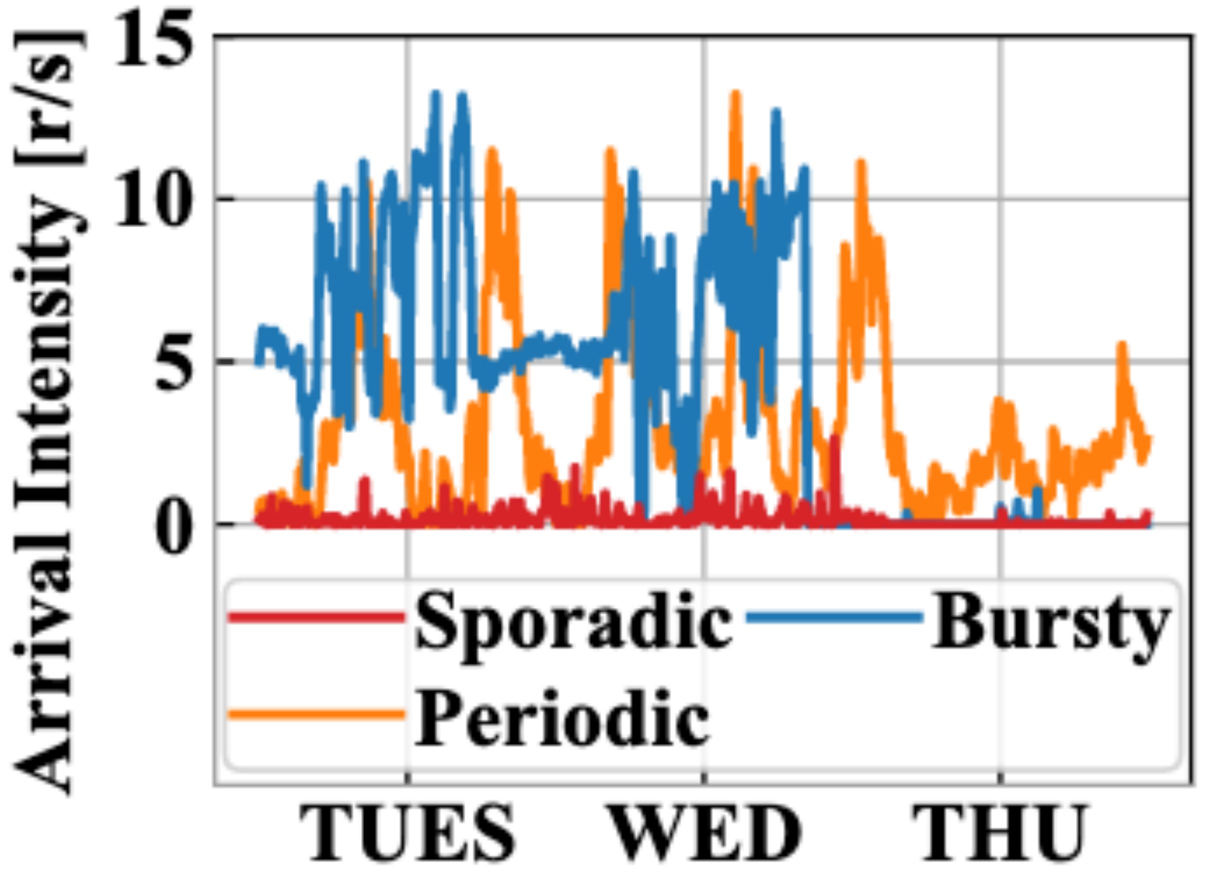


Figure 10: The three production trace examples.

Evaluation

Baseline

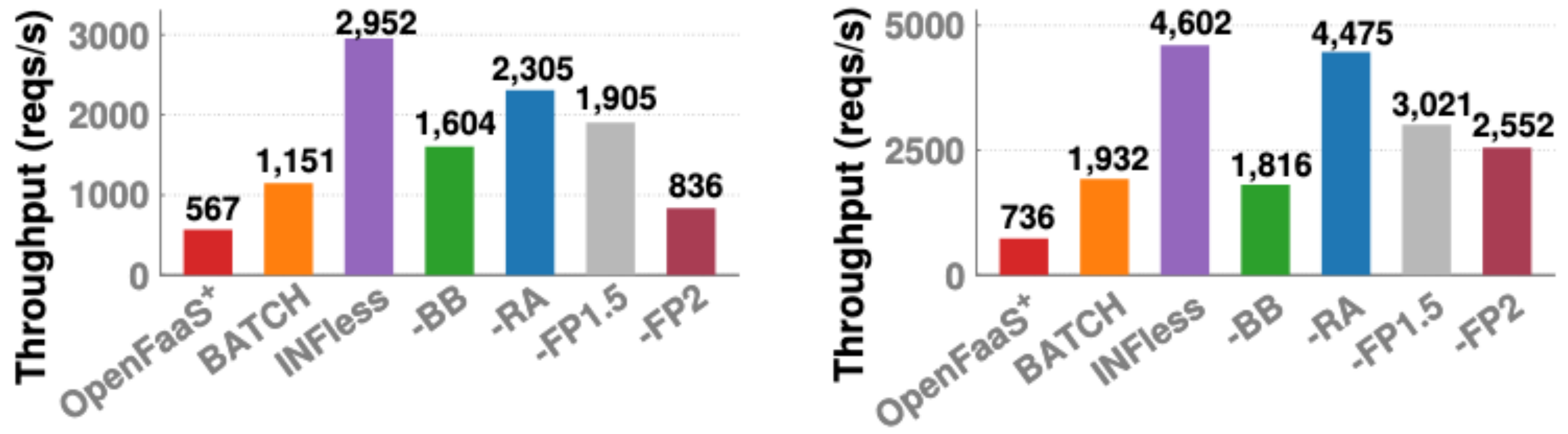
- $OpenFaaS^+$ (gpu support)
- *BATCH* (OTP design， gpu support)

Table 3: Comparison of serverless inference systems.

Features	OpenFaaS+	BATCH	INFless
GPU devices support	Yes	Yes	Yes
Batching mechanism	No	OTP	Built-in
Function profiling	No	Yes	Combined Ops
Instance auto-scaling	Uniform	Uniform	Non-Uniform
Batch-aware dispatcher	No	No	Yes
Keep-alive policy	Fixed	Fixed	Dynamic

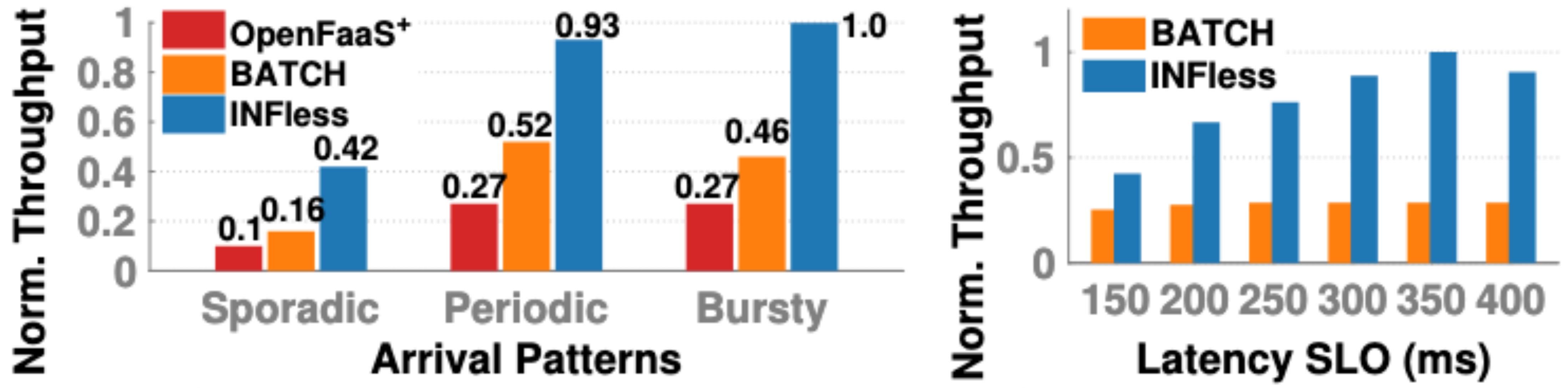
Evaluation

Local Cluster Evaluation — High throughput & Component analysis



(a) OSVT (b) Q&A robot

Figure 11: Throughput comparisons and component analysis of INFless. BB: build-in, non-uniform batching; RS: resource scheduling; OP1.5: add the predicted latency by 50%; OP2: add the predicted latency by 100%.



(a) Thp. under production traces (b) Thp. under different SLOs

Figure 12: The normalized throughput comparison of INFless with baselines: (a) under different production traces; (b) under different latency SLOs.

benefit from **much fewer fragments**

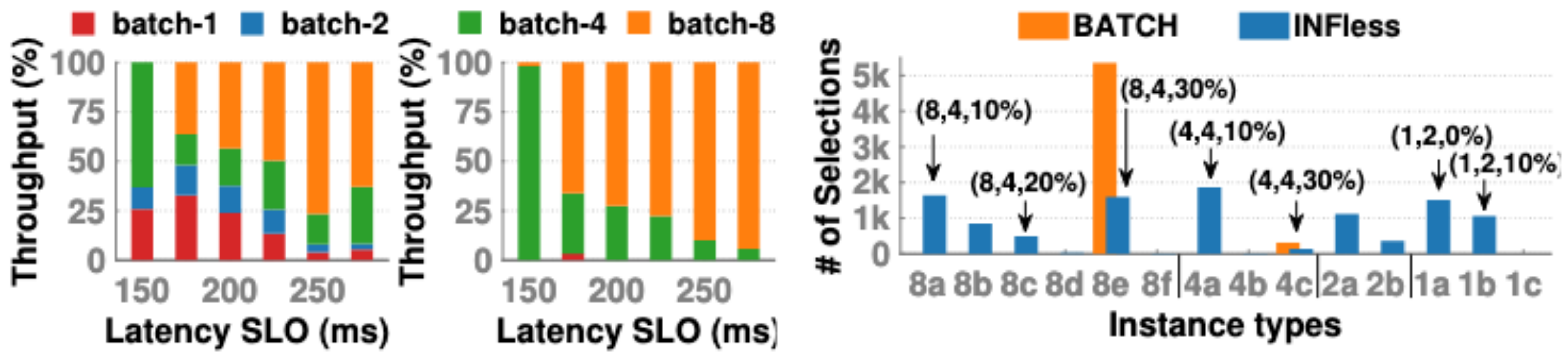
Improvement in throughput

Compared with OpenFaaS+ & BATCH

5.2x & 2.6x

Evaluation

Local Cluster Evaluation — Flexible configurations & Less over-provisioning



(a) INFless (b) BATCH (c) Instance configuration selection

Figure 13: Throughput distribution contributed by different batchsize settings by (a) INFless and (b) BATCH. (c) Resource configuration distribution of instances by INFless and BATCH. (b, c, g) represents the batchsize, CPU core and GPU SM configuration of instances.

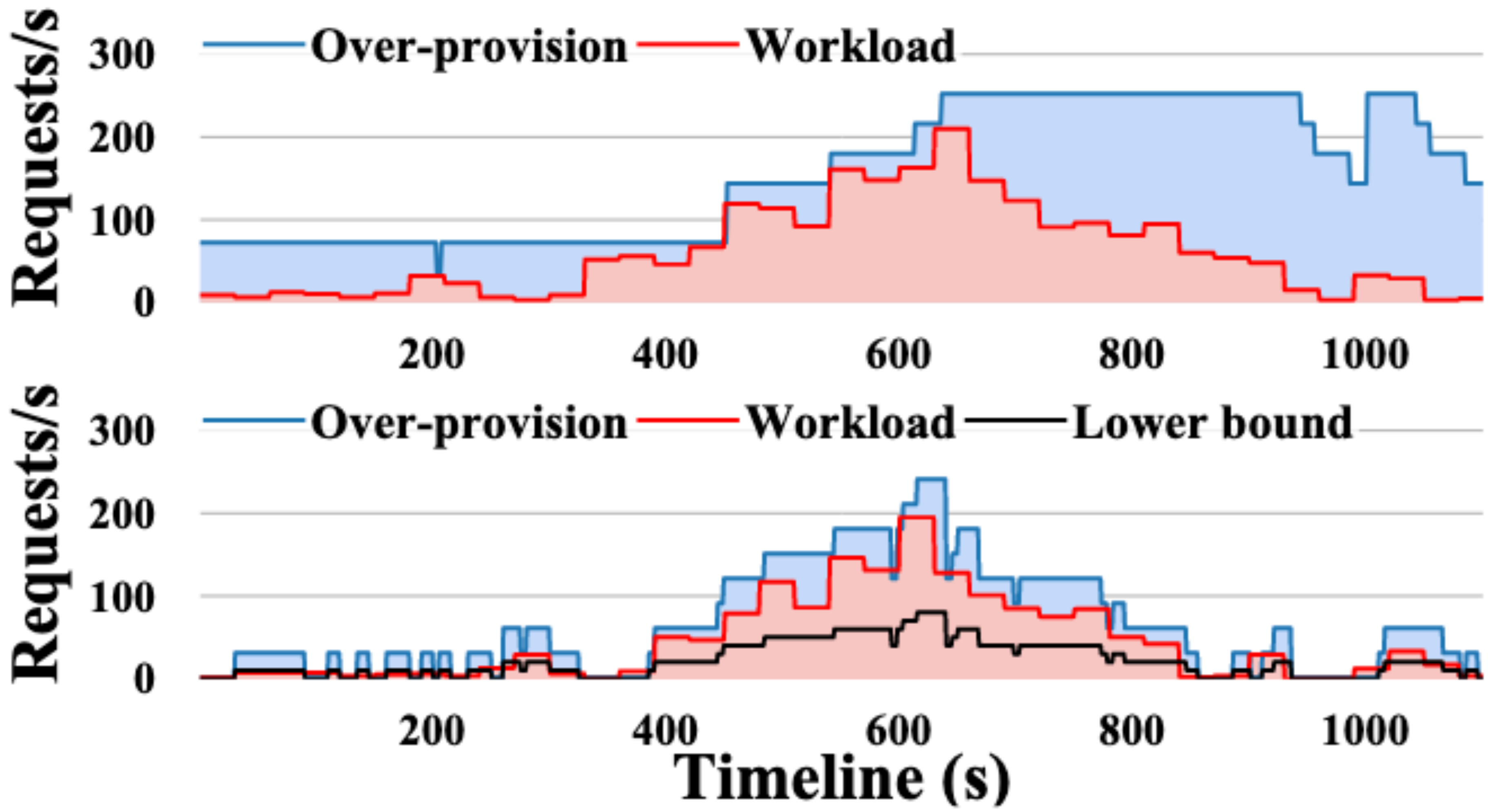


Figure 14: Resource provisioning by BATCH (top) and INFless (bottom).

Flexible configurations : more various batchsize settings & resource allocations

Less over-provisioning : reduce resource by 60%

Evaluation

Local Cluster Evaluation — SLO violation & Cold start

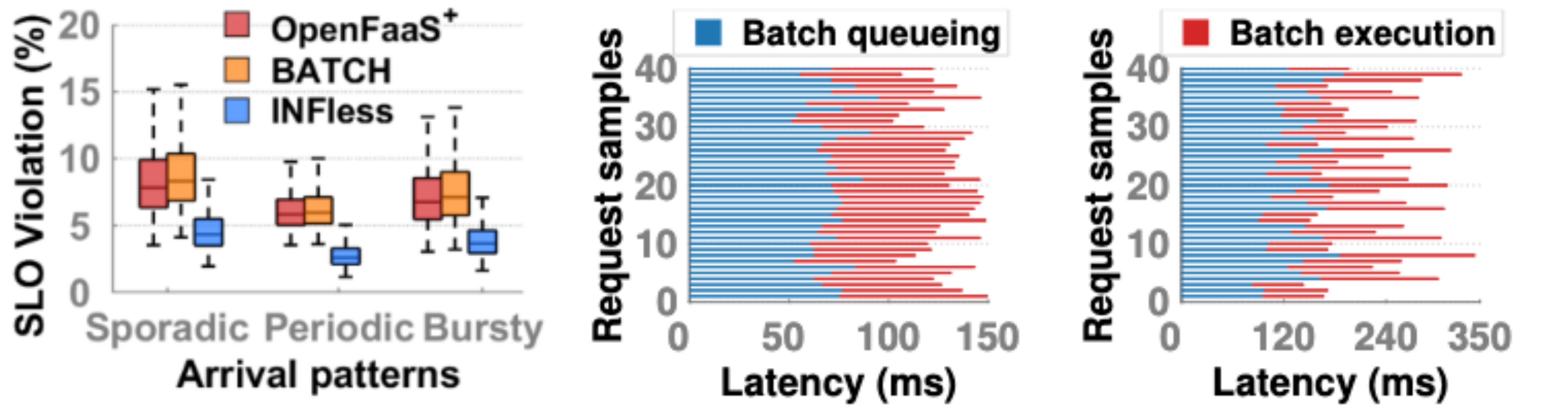


Figure 15: (a): SLO violation comparison of INFless with baselines and (b): latency breakdown of INFless under different latency SLO settings.

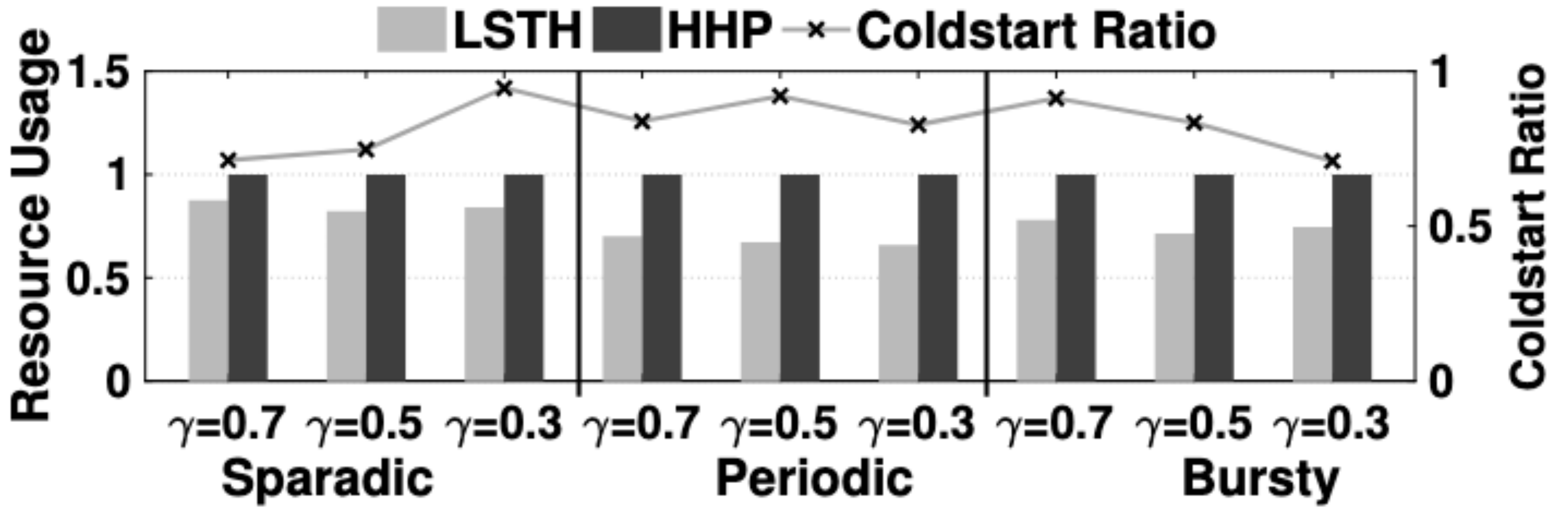


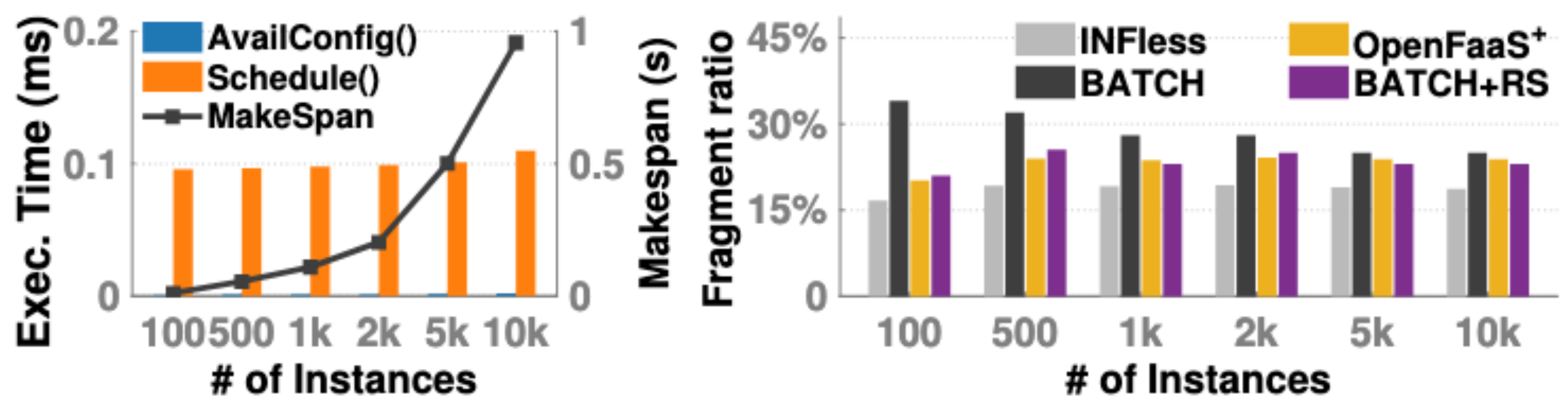
Figure 16: Cold start rate comparison.

SLO : over 95% in three modes

Cold start : reduce by 20% compared with HHP

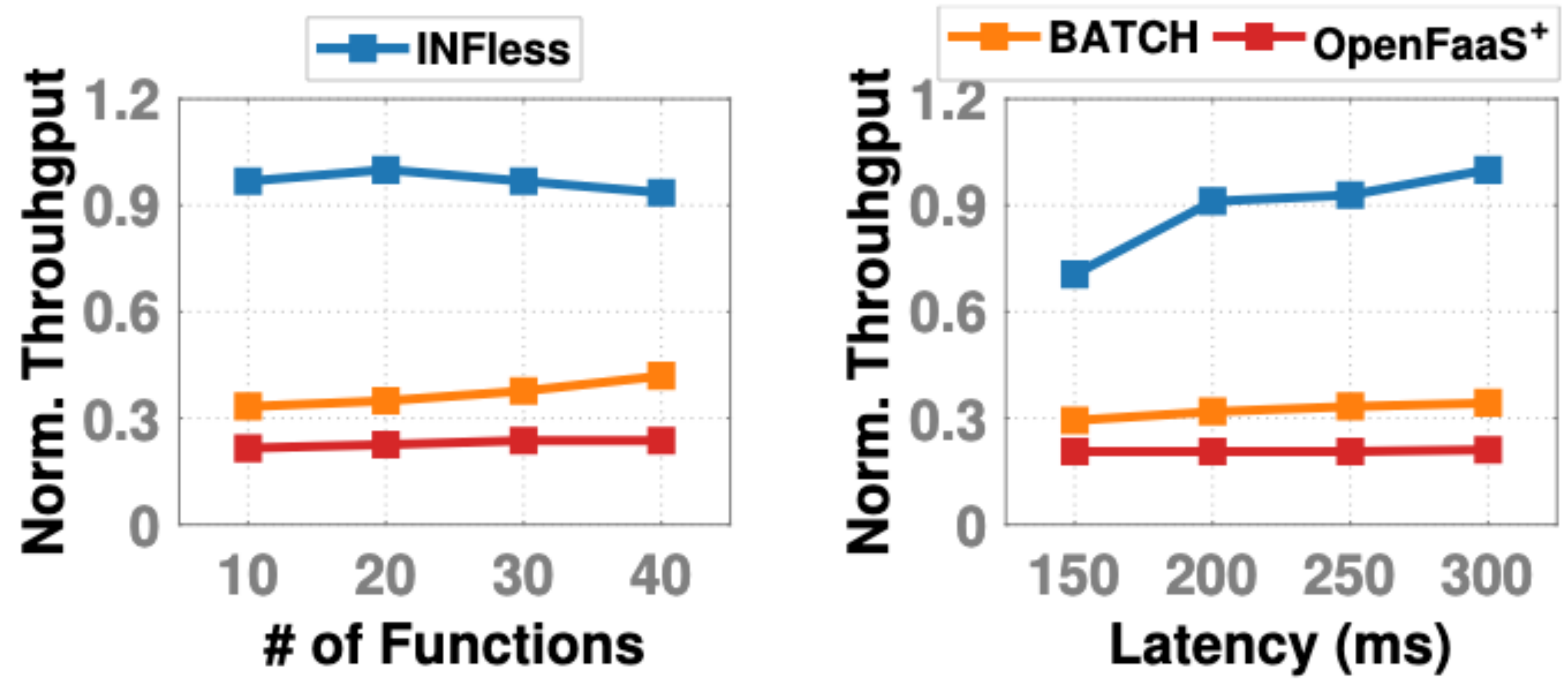
Evaluation

Large Scale Simulation — Scalability & Resource fragments & Cost efficiency



(a) Scheduling overhead (b) Resource fragments

Figure 17: Scheduling overhead and resource fragments of INFless in large-scale simulations.



(a) Thp. under different # of funcs (b) Thp. under different SLOs

Figure 18: Throughput evaluation in the large-scale simulation.

Table 4: Computation cost comparison.

	AWS EC2	OpenFaas+	BATCH	INFless
CPUs per 100RPS	49.42	55.63	41.45	13.91
GPUs per 100RPS	2.47	2.13	1.34	0.51
Cost per request [$]	2.23×10^{-5}	2×10^{-5}	1.32×10^{-5}	1.6×10^{-6}

Conclusion

- Develop a novel serverless computing platforms which satisfy the **SLO latency** & **high throughput** requirements of inference services :
  - **Built-in, non-uniform batching -> high throughput**
  - **Combined operator porfiling & LSTH -> meet SLO requirement**
- **Co-design** the batch management and heterogeneous resource allocation
- **High resource efficiency (low cost for users), release resource over-provisioning**

https://github.com/TankLabTJU/INFless/

Thanks

2023-6-15

Presented by Guangtong Li