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

(W3) Paper Reading

Nan Lin

Shanghai Jiao Tong University

2024-07-14

Basic Info

- TANKLAB (Tianjin University) and 58.com
- ASPLOS (2022)

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

Yanan Yang
College of Intelligence & Computing (CIC), Tianjin University, Tianjin Key Lab. of Advanced Networking
Tianjin, China
ynyang@tju.edu.cn

Laiping Zhao*
CIC, Tianjin University, TANKLAB
Tianjin, China
laiping@tju.edu.cn

Yiming Li
CIC, Tianjin University, TANKLAB
Tianjin, China
l_ym@tju.edu.cn

Huanyu Zhang
CIC, Tianjin University, TANKLAB
Tianjin, China
3016218159@tju.edu.cn

Jie Li
CIC, Tianjin University, TANKLAB
Tianjin, China
lijie20@tju.edu.cn

Mingyang Zhao
CIC, Tianjin University, TANKLAB
Tianjin, China
mingyang@tju.edu.cn

Xingzhen Chen
58.com
Beijing, China
chenxingzhen@58.com

Keqiu Li
CIC, Tianjin University, TANKLAB
Tianjin, China
keqiu@tju.edu.cn

Outline

Outline

ML Inference

- **Machine learning (ML) inference**
- Typical exploration: Amazon Alexa, Facebook Messenger Bot, OCR

Limitations of existing serverless platforms

They do not address the challenge of

- *providing solutions for guaranteeing latency*
- *resource efficiency at the provider side is very low*

Limitations of existing serverless platforms

1. **High latency**:
   - SLO: required to respond within 200 ms
   - However, large models does not satisfy this requirement

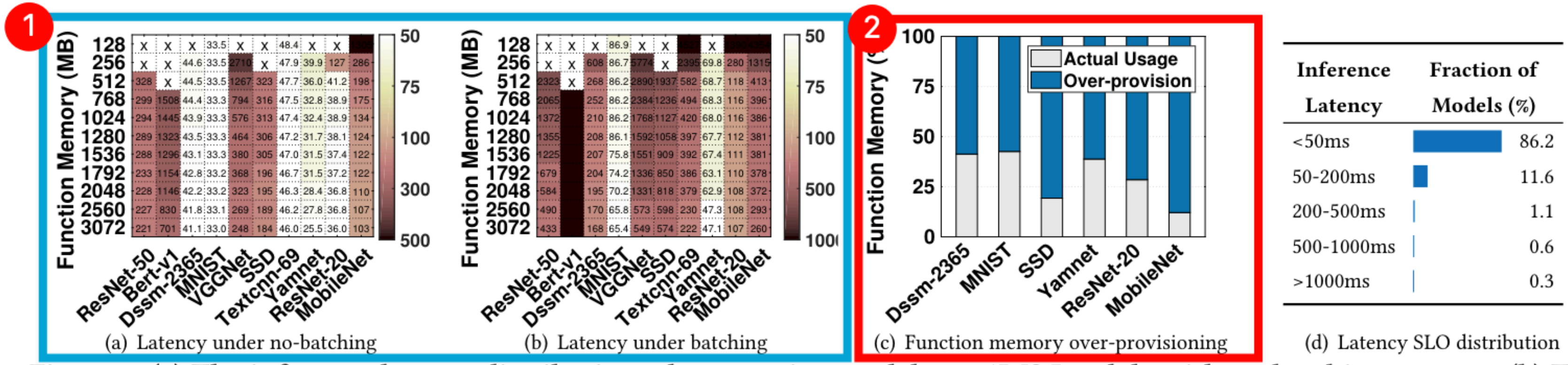


Figure 2: (a) The inference latency distribution when running models on AWS Lambda without batching support. (b) The inference latency distribution when running models on AWS Lambda with OTP-batching support, where × means that the memory size is too small to load the model. (c) The memory over-provisioning for achieving low latency requirement. (4) Real-world latency SLO distribution by the local life service website.

- Reason: Lack the support of accelerators (e.g. GPU)

Limitations of existing serverless platforms

2. **Poor resource utilization**:
   - Resource over-provisioning: *proportional CPU-memory allocation policy* apply for a larger memory to obtain a sufficient CPU quota.

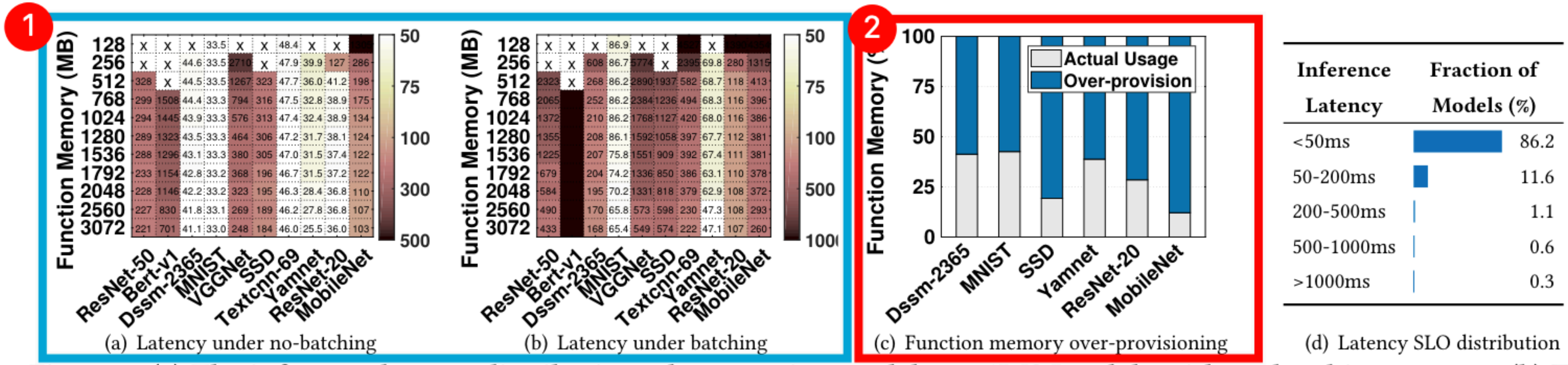


Figure 2: (a) The inference latency distribution when running models on AWS Lambda without batching support. (b) The inference latency distribution when running models on AWS Lambda with OTP-batching support, where × means that the memory size is too small to load the model. (c) The memory over-provisioning for achieving low latency requirement. (4) Real-world latency SLO distribution by the local life service website.

Limitations of existing serverless platforms

3. **Low Throughput**:
   - (Revision) *One-to-one mapping*; batching could increase throughput
4. **Shortages of *On-Top-of-Platform (OTP)* design**:
   - Another dedicated serverless
   - Unaware of the situation inside the serverless platform
   - Lacking the codesign of configurations, instance scheduling and resource allocation, therefore limited improvement

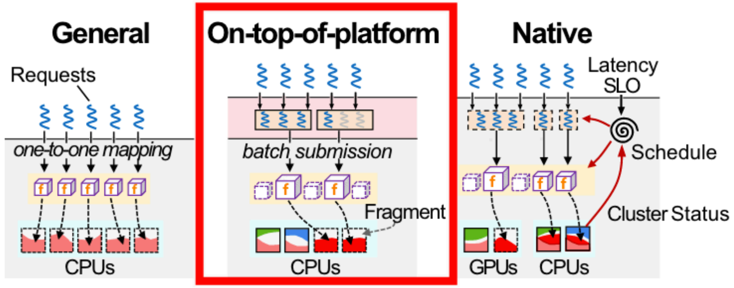


Figure 1: Schematic overview of serverless inference systems.

Limitations of existing serverless platforms

Summary:

1. High Latency, especcailly for batch-enabled inference
2. Resource over-provisioning
3. Low throughput caused by *OTP*
4. *OTP batching* lack codesign

Limitations of existing serverless platforms

Summary:

1. High Latency, especailly for batch-enabled inference
2. Resource over-provisioning
3. Low throughput caused by *OTP*
4. *OTP batching* lack codesign

Motivation:

1. Support hybrid CPU/accelerations
2. Producing resource-efficient scheduling
4. Support built-in batching

Outline

High-level overview

- Native implementation
  - Uniform scaling → non-uniform scaling

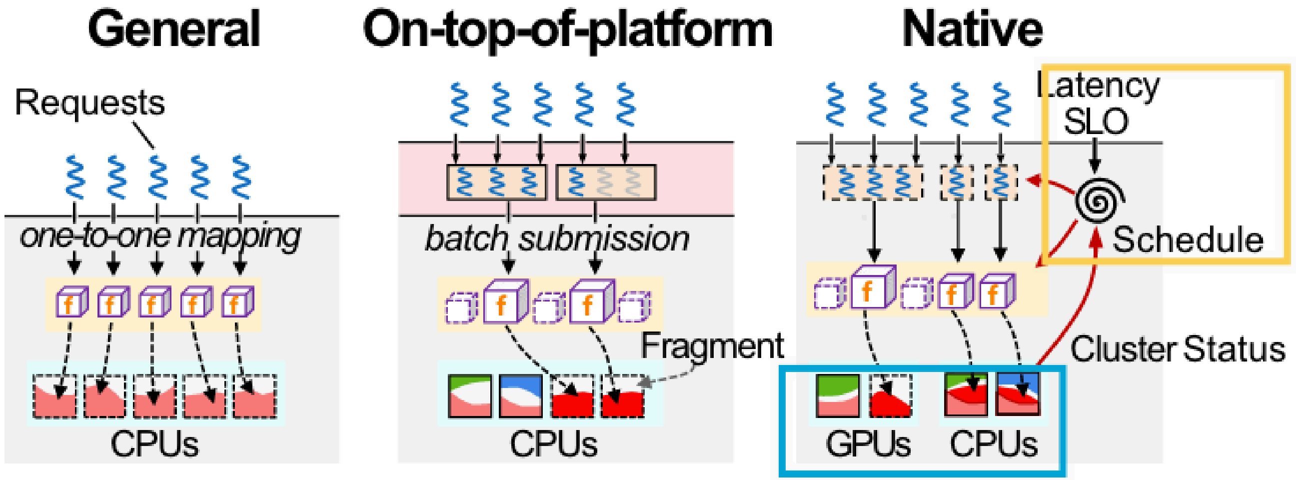


Figure 1: Schematic overview of serverless inference systems.

Challenges

Challengable work: Achieving both low latency and high resource efficiency.

- Application layer
  - Function profiling cost
  - Request forwarding, decision making, scheduling time
- Decision layer (Optimal scheduling decisions)
  - Batchsize
  - Instance placement
  - Workload dispatching rate
- Resource layer
  - Device collaboration
  - Hardware affinity

System Architecture overview

- Design overview

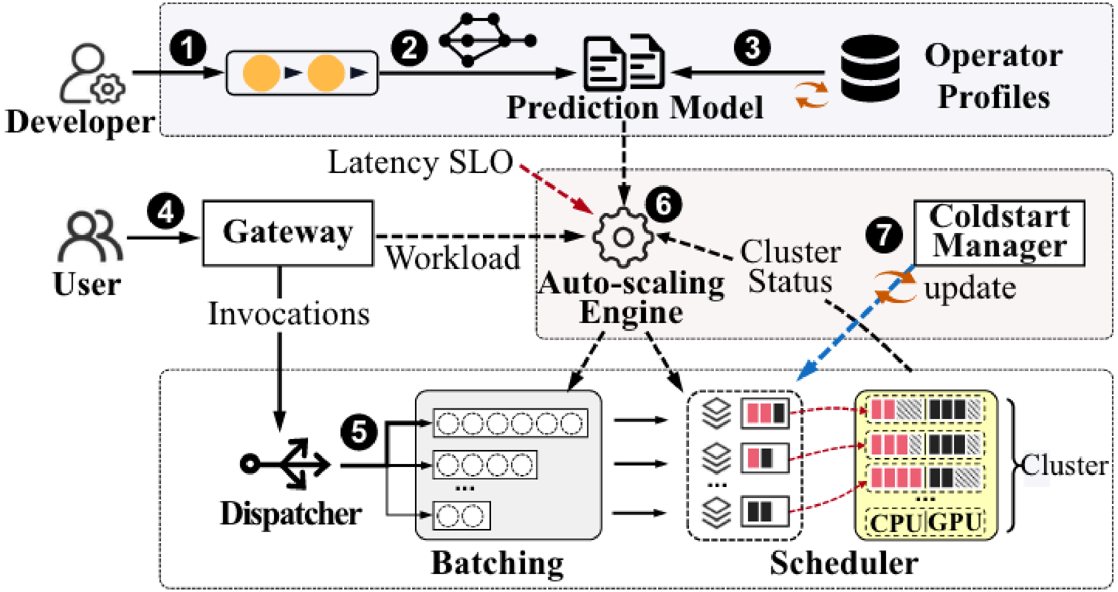
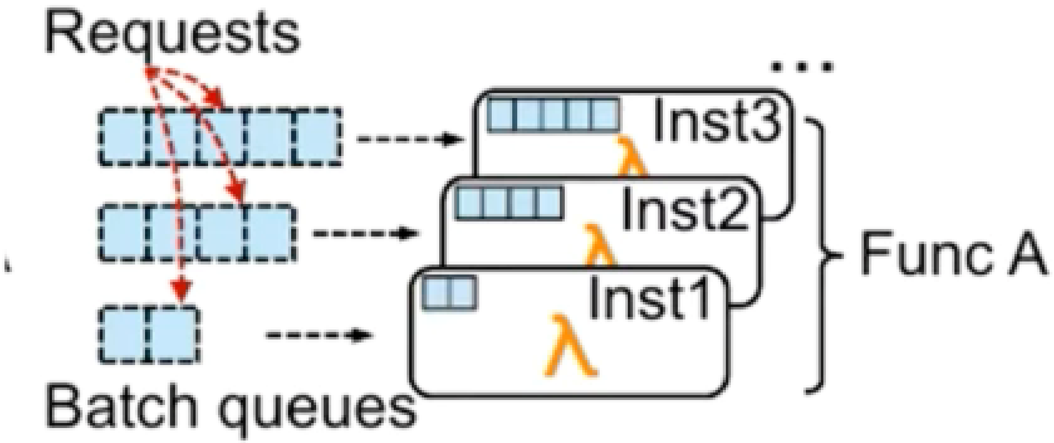


Figure 4: The design overview of *INFless*.

Built-in, non-uniform batching

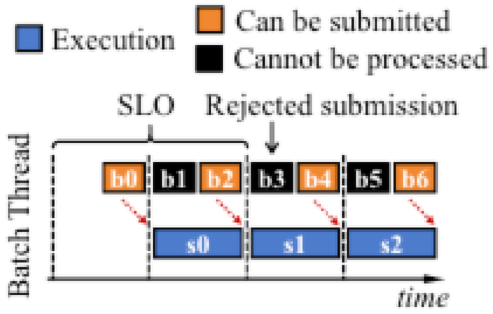
- Non-uniform: individual batch queue, different configuration
- Inner integration



Built-in, non-uniform batching

- Lower and upper bound of workload can be processed in one instance (denoted resp. r_{up} and r_{low}, b is the batchsize)
  - SLO
  - Over-submission

$$r_{\text{up}} = \frac{1}{t_{\text{exe}}} \times b, \quad r_{\text{low}} = \frac{1}{t_{\text{slo}} - t_{\text{exe}}} \times b$$



(a) Over-submission

Profiling

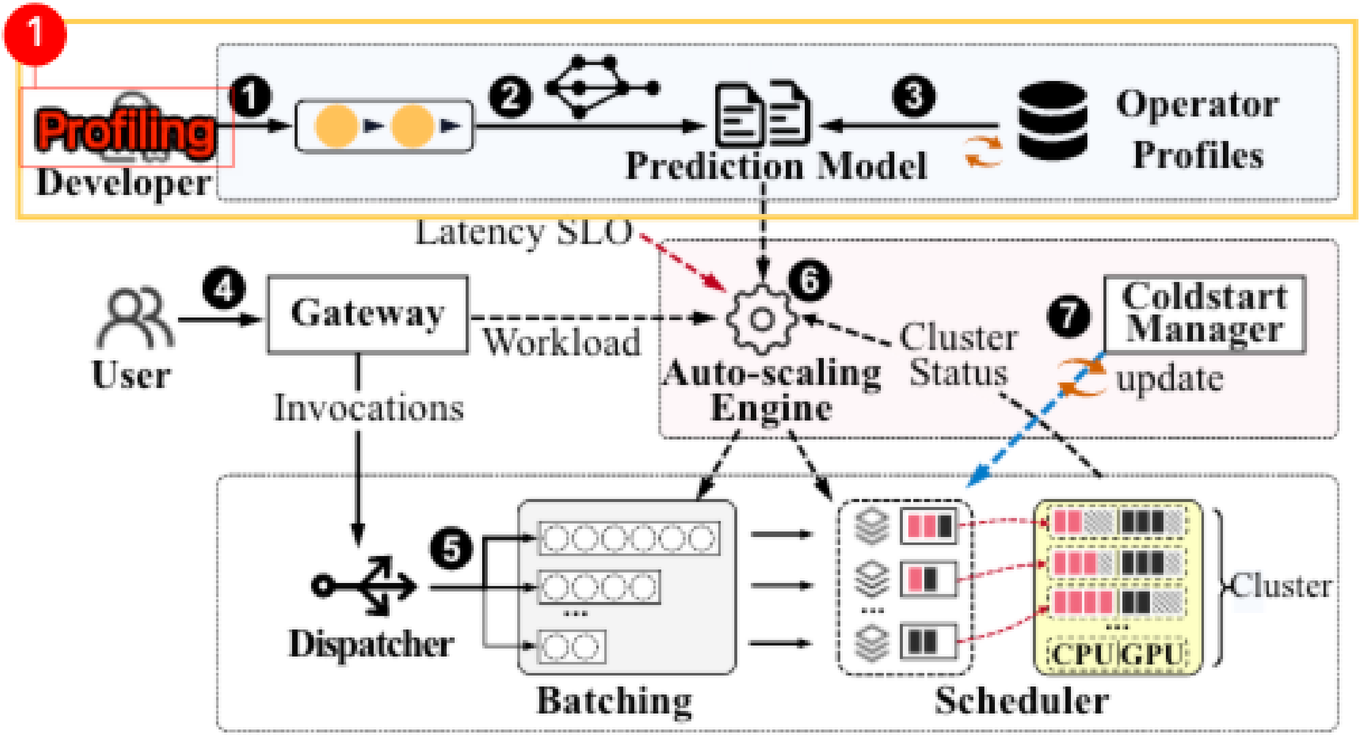
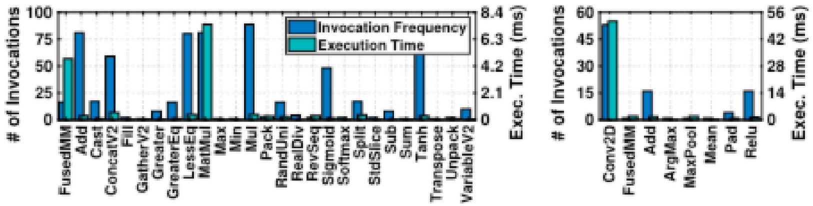


Figure 4: The design overview of *INFless*.

Profiling

- Offline profiling each function is costly
- **Combined operator profiling (COP)** method
  - Operators: *Matmul*, *Sum*, …
  - Inference functions share a common set of operators
  - Exec time dominated by a small subset of operators



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

Figure 7: Calling frequency and execution time of the DNN operators. (a) LSTM-maxclass contains 27 operators, and *MatMul* takes more than 95% of the overall execution time; (b) ResNet-50 contains 8 operators, and most of the execution time is spent on *Conv2D*.

Profiling

1. Given an operator o_i,

$$i_i = (p_i, b_i, c_i, g_i, t_i)$$

- p_i size of each piece of input data (e.g. size of image)
- b_i batchsize
- c_i CPU-related resources
- g_i GPU-related resources
- $t_i = t_i(p_i, b_i, c_i, g_i)$ corresponding execution time

2. Collect peach operators' profiles and store them in a database

- Consider discrete values: $b_i \in \{2^0, 2^1, ..., 2^{i_{\max}}\}$

Profiling

3. Calculate the overall latency

- Sequence chain

$$t_{\text{chain}} = \sum_{c \in \text{ chain}} t_c$$

- Parallel chain

$$t_{\text{branch}} = \max_{b \in \text{ branch}} t_b$$

Auto-scaling

Induce number of instances/batchsize and configs for each queue from function profiles and workload

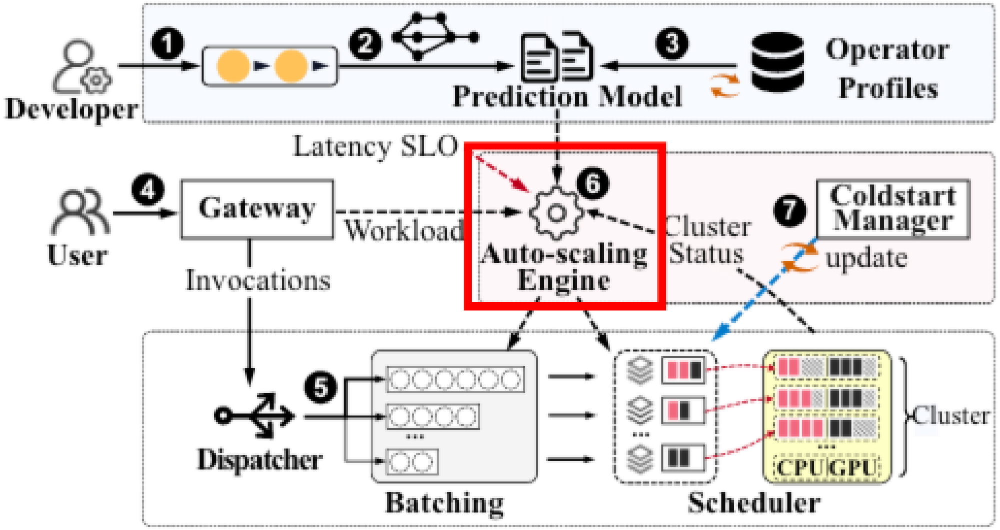


Figure 4: The design overview of *INFless*.

Auto-scaling

$$l = t_{\text{batch\_queue}} + t_{\text{batch\_exec}} + t_{\text{cold\_start}}$$

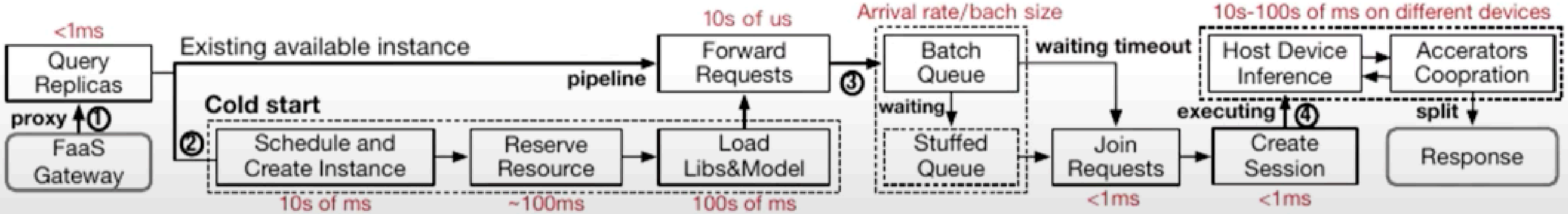


Figure 1: Serverless inference latency breakdown. The red numbers are the duration time of each step in inference process. The **proxy** and **pipeline** path are the process of receiving a request and forwarding to the service end-point in Gateway, and **join** and **split** are the fusion and parsing of requests for batching process, respectively.

Auto-scaling

Formalization:

- Schedule $x_{ij} = \delta_i \delta_j$, instance i on server j
- $y_j = \delta_j$ if server k is used for instance deployment
- C_j, G_j available CPU and GPU
- β conversion factor, comparing FLOPS
- R_k arrived requests toward function k

Auto-scaling

$$\text{minimize}: \quad \sum_{j}^{m}(\beta C_j + G_j)y_j \tag{2}$$

$$t^i_{wait} + t^i_{exec} \le t^i_{slo}, \quad \forall i \in [1, .., n] \tag{3}$$

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

- SLO constraint

$$\sum_{i}^{n} c_i x_{ij} \le C_j y_j, \quad \forall j \in [1, .., m] \tag{5}$$

$$\sum_{i}^{n} g_i x_{ij} \le G_j y_j, \quad \forall j \in [1, .., m] \tag{6}$$

- Resource capacity constraint

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

- Workload constraint

$$x_{ij} \in \{0, 1\}, \quad y_j \in \{0, 1\} \tag{8}$$

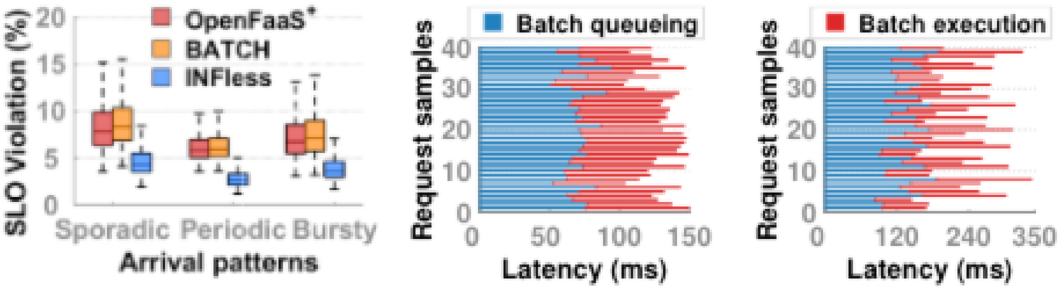
$$b_i, c_i \in Z_+, \quad g_i \in Z \tag{9}$$

- Variables

Outline

SLO

violation rate $\leq 3.1\%$ on average

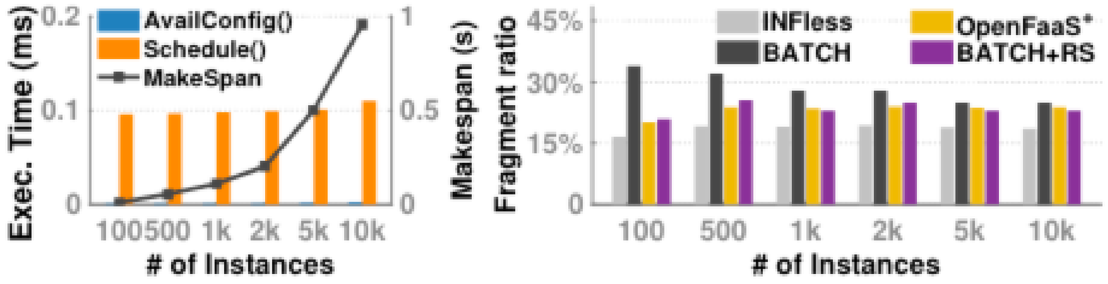


(a) SLO violations (b) SLO=150ms (c) SLO=350ms

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

Resource fragment

resource fragment ratio $< 15\%$ on average



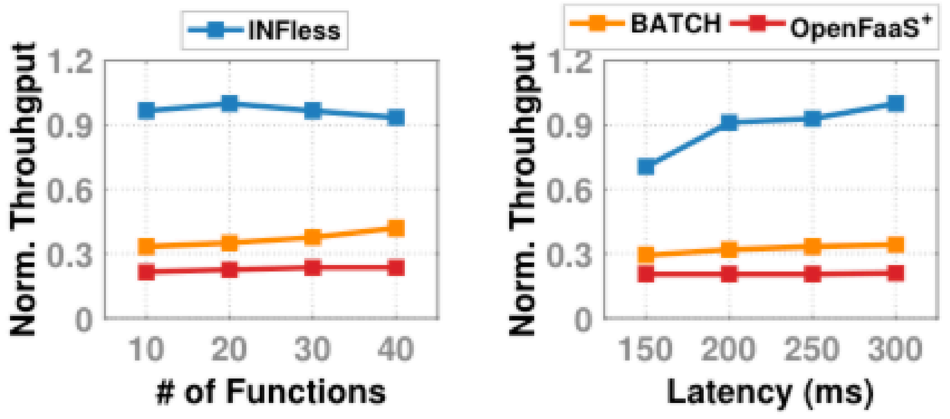
(a) Scheduling overhead (b) Resource fragments

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

Throughput

throughput $\times$ 2.6 and $\times$ 4.2 resp.

Throughput



(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}

Outline

- A solution for achieving both low latency and high resource efficiency
- “Just a start”