Proteus: A High-Throughput Inference-Serving System with Accuracy Scaling

ASPLOS' 24

Sohaib Ahmad,* Hui Guan,* Brian D. Friedman,* Thomas Williams,* Ramesh K. Sitaraman, Thomas Woo

Presenter: Yunpeng Xu
2025.5.16

Author

研究方向：系统与机器学习的交叉优化

研究核心聚焦于**如何提升机器学习推理的资源效率**，特别是在**推理服务系统**（Inference Serving Systems）中实现**高吞吐、低延迟和精度可调**的运行机制。

- **高效机器学习系统与模型推理服务系统**
  - Proteus（ASPLOS 2024）
  - Loki（HPDC 2024）
- **云边协同的模型部署与资源调度优化**
  - Proteus（边缘推理）
  - Bell Labs（Edge-Cloud Scheduling）
- **多模型协同执行、精度与延迟动态权衡**
  - DiffServe（MLSys 2025）
  - Proteus（模型级精度伸缩机制）
- **异构硬件与推理流水线的联合伸缩与调度**
  - Loki（硬件与模型精度“双伸缩”）

Sohaib Ahmad

Research Scientist

Meta

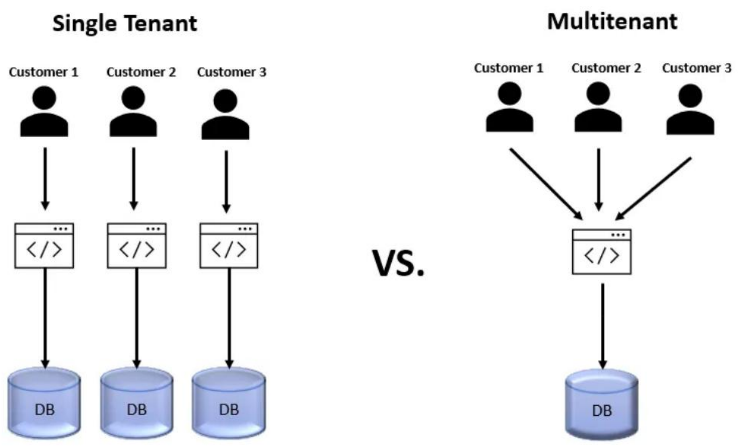
sohaibahmad759.github.io

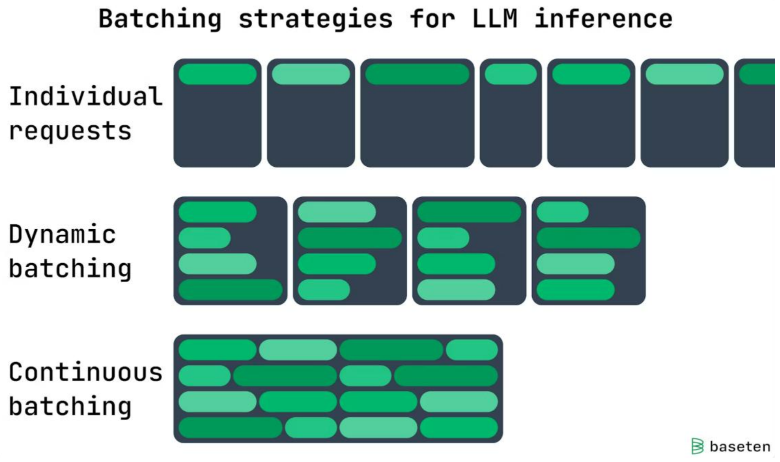
Contents

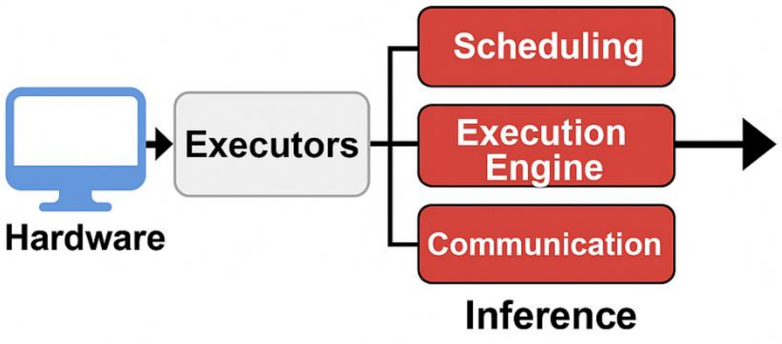
- Background & Motivation
- Challenges
- Design
- Evaluation
- Thinking

Background & Motivation

Tranditional inference-serving systems:







Multi-Tenancy **Batching** **Low-Level Optimizations**

hardware scaling: more devices and more powerful accelerators ?

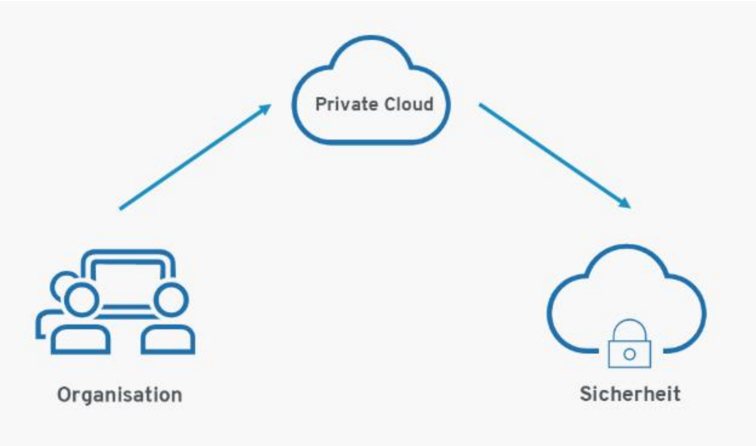
Background & Motivation

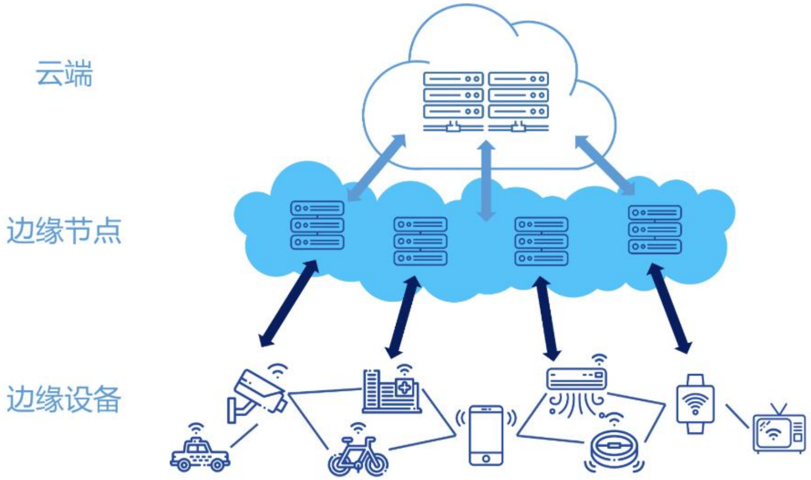
Problems:

- **the limited availability** of hardware resources.
- purchasing and maintaining more devices to cater to peak demands can **be expensive and cost-ineffective.**

Private Cloud

Edge Cluster





Fixed-size

vs

Increasing query demands

How To Handle?

Background & Motivation

Accuracy Scaling

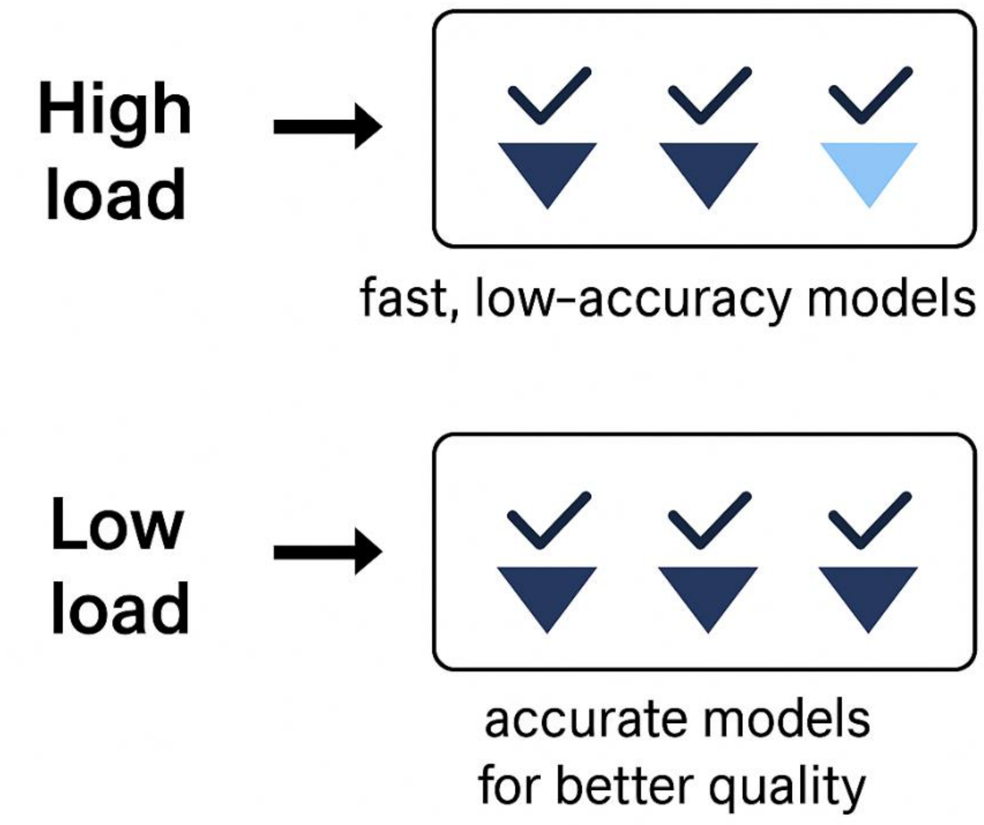
Dynamically adjust model **precision** to trade off **accuracy for throughput.**

Minimal SLO violations

&

Maximal inference accuracy

→ **High Throughput**



Challenges

◆ Challenge 1： Determine the “**right**” amount of accuracy scaling

- Many model variants
- Heterogeneous devices
- Many applications

- Model selection
- Model placement
- Query assignment

◆ Challenge 2： Adaptive batching under **micro-scale** arrival fluctuations

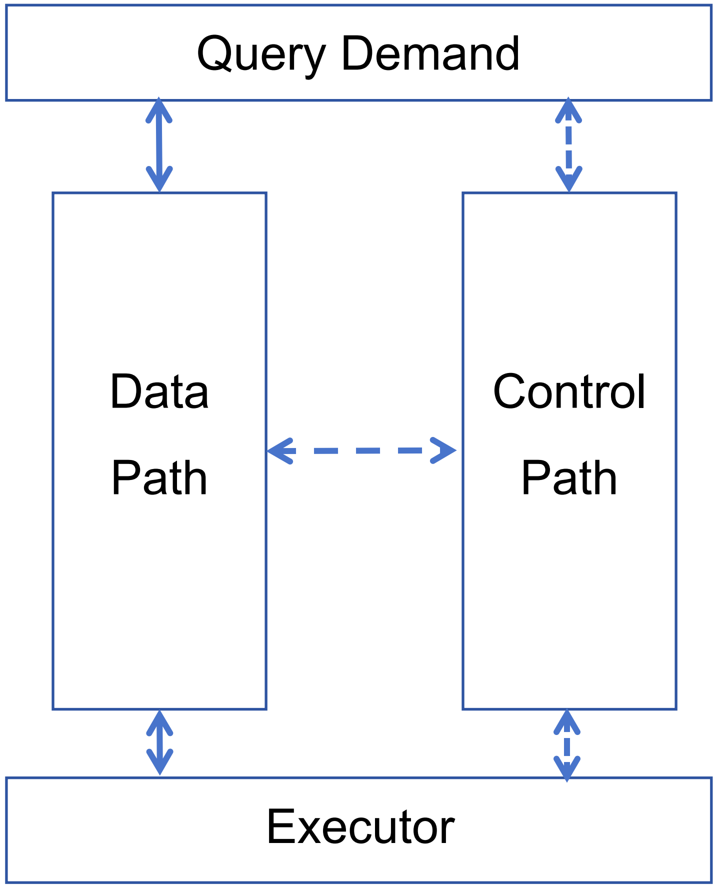
a non-work-conserving approach **vs**

- a work-conserving approach
- adaption based on whether timeouts or not
- significant changes to the underlying ML framework

Challenges

★ **Decoupling Control and Data Paths**

resource allocation problems
vs
serving queries



Data Path:

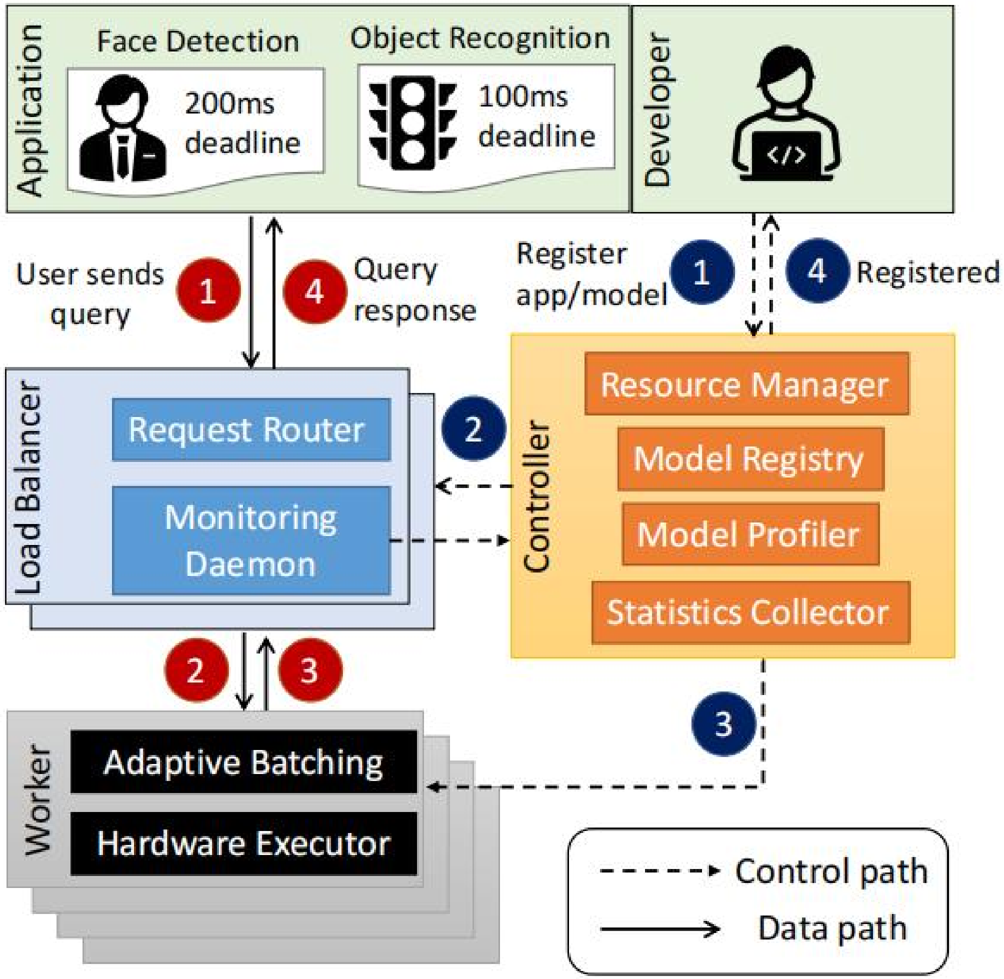
Execute incoming queries along **pre-planned** paths with no real-time decisions.

Control Path:

Periodically optimize model selection and placement strategies with **MILP.**

Design

System architecture of Proteus



- **Controller:**

1. receive the registration of the application and model variants
2. confirm the registration status

- **Load Balancer:**

1. receive inference queries from its designated application
2. respond with model execution results

- **Worker:**

1. determine the suitable batch size
2. execute its hosted model variant to serve inference queries

Design

Core modules 1： Resource Management

→ responds to **macro-scale** changes about **QPS**

- **Model Selection**
- **Model Placement**
- **Query Assignment**

a **MILP** optimization

Optimization variables:

$x_{d,m}$	true if model variant m is hosted on device d; false otherwise
$y_{d,q}$	percentage of queries of type q routed to device d

↓

$$\max_{\{x_{d,m}\},\{y_{d,q}\}} \sum_q a_q \quad s.t. \text{ Constraints Eqs. 1-6}$$

Constraints

1. Device capacity limits
2. Target query throughput
3. Latency (SLO) limits

Design

To solve the **MILP** problem, we have to estimate **the SLO-aware throughput** capacity $P_{d,m,q}$.

$$P_{d,m,q} = \frac{\text{Maximum allowed batch size for } d, m, q}{\text{Profiled latency (seconds)}}$$

the maximum inference latency for any model ≤ ½ (its latency SLO)

- Meet the query's latency SLO
- Fit into device memory

Maximum batch size = **min** (batch that meets SLO and the device can support)

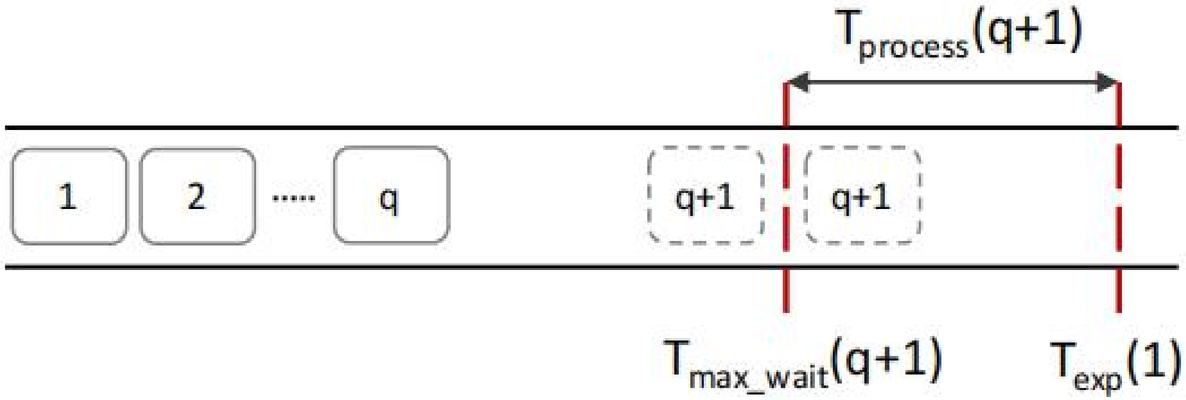
Design

Core modules 2：Adaptive Batching

→ responds to **micro-scale** changes in terms of varying query **inter-arrival times.**

Two key ideas:

- Proactive
- Non-work-conserving



$$T_{\text{max wait}(q+1)} = T_{\exp(1)} - T_{\text{process}(q+1)}$$

Case1：We do not receive any query until $T_{max\_wait}(q+1)$

Case2：We receive the $q+1^{st}$ query before $T_{max\_wait}(q+1)$

Evaluation

1. Workloads

① Real-world trace: a public trace from Twitter

- Use a **Poisson process** to simulate **inter-arrival times** for queries.
- Use a **Zipf distribution** to distribute querys to different model families.

② Synthetic traces: made for the **stress-test** in response to burstiness

- Evaluate resource allocation on the **macro-scale**.
- Evaluate adaptive batching on the **micro-scale**.

docker + kubernetes

The cluster contains:

a. 20 Intel(R) Xeon(R) Gold 6126 @ 2.60GHz CPU workers
b. 10 NVIDIA GeForce GTX 1080 Ti GPU workers
c. 10 NVIDIA V100 GPU workers

Evaluation

2. Model Variants

Model Family	Model Variants
ResNet (classification) [20]	18, 34, 50, 101, 152
DenseNet (classification) [22]	121, 161, 169, 201
ResNest (classification) [47]	14, 26, 50, 269
EfficientNet (classification) [40]	b0-b7
MobileNet (classification) [21]	1.0, 0.75, 0.5, 0.25
YOLOv5 (object detection) [25]	n, s, m, l, x
BERT (sentiment analysis) [11]	RoBERTa-base, large; [29], ALBERT-base, large, xlarge, xxlarge [26]; BERT-base, tiny, mini, small, medium, large [41]
T5 (translation) [34]	small, base, large, 3b, 11b
GPT-2 (question answering) [33]	base, medium, large, xl

◆ **Notices**

① the setting of latency SLO for each model family

② normalize the accuracy of each model variant

Evaluation

3. Baselines

- Fully static： Clipper
- Partially dynamic： Sommelier
- Mostly dynamic： INFaaS

Control Variables

① Clipper-HT

② Clipper-HA

③ Sommelier

④ INFaaS-Accuracy

Feature	Clipper	Sommelier	INFaaS	Proteus
Model placement	Static	Static	Heuristic	MILP
Model selection	Static	Heuristic	Heuristic	MILP
Accuracy scaling	No	Limited[3]	No[4]	Yes
Adaptive batching	Yes	No	Yes	Yes

Evaluation

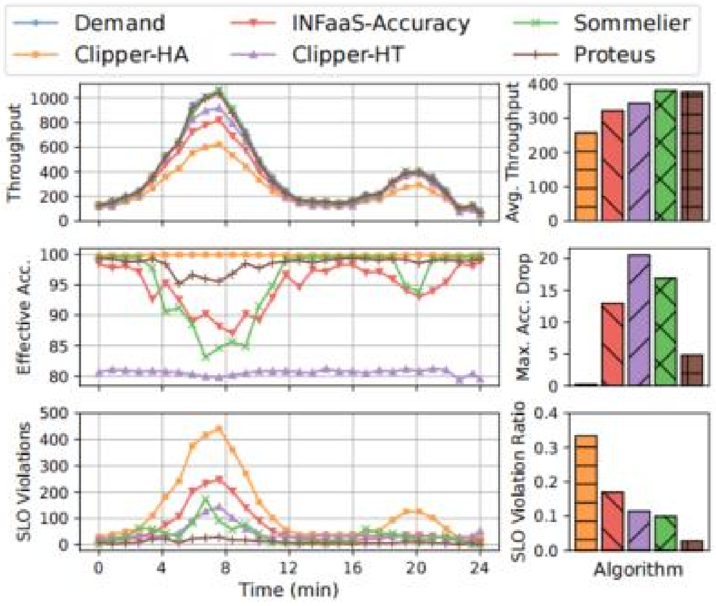
4. Evaluation Metrics

① Throughput

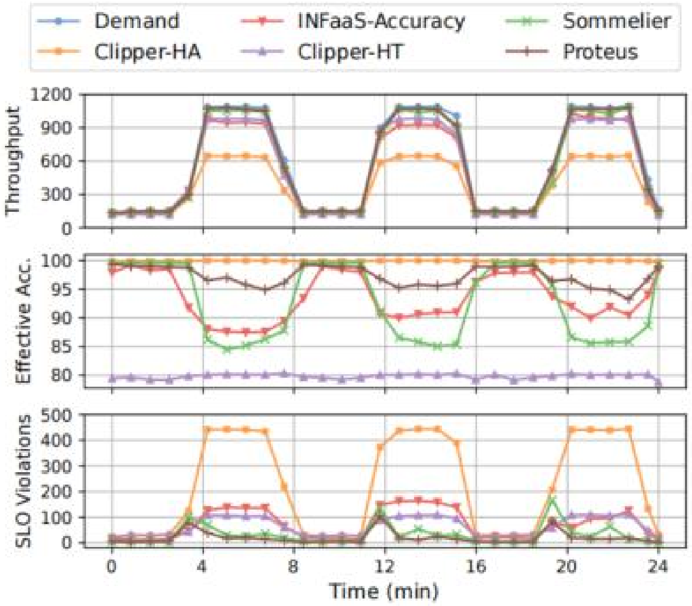
② Effective Accuracy

③ Maximum Accuracy Drop

④ SLO Violation Ratio



End-to-End Performance Comparison



Bursty Workload Test

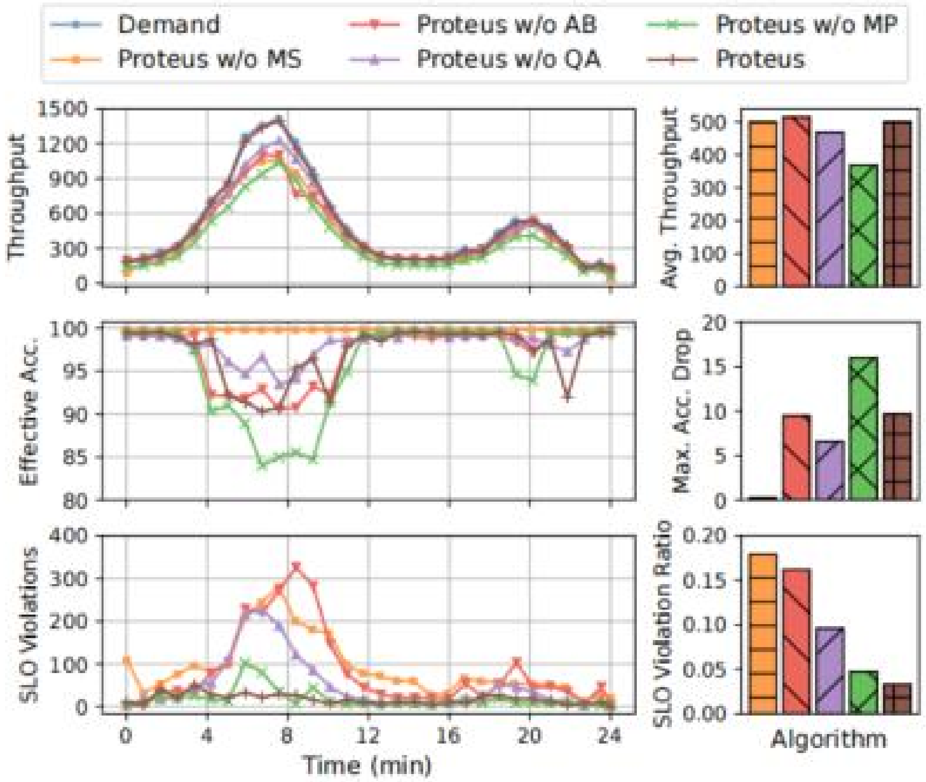
High Throughput

Low Accuracy Drop ← MILP Allocation Strategy

Low SLO Violation Ratio ← Adaptive Batching Algorithm

Evaluation

5. Ablation Study



① Effective accuracy

Model Placement > Adaptive Batching > Query Assignment > Model Selection

② SLO violations

Model Selection > Adaptive Batching > Query Assignment > Model Placement

Evaluation

6. Decision Overhead

4.2 seconds to solve the MILP is this case!

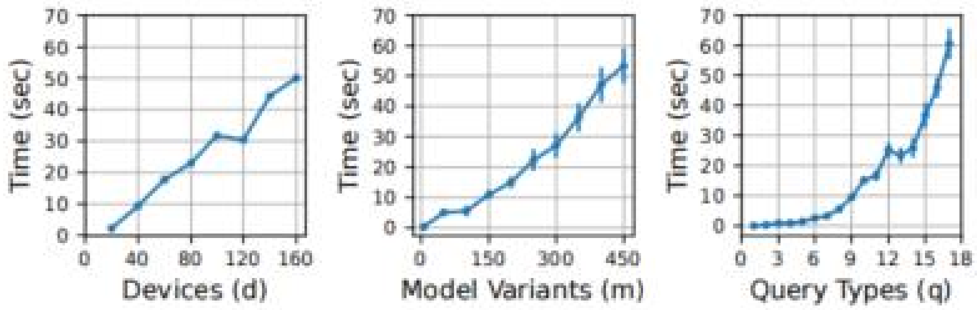
The overhead of Proteus reflects in two ways：

① overhead of the **Request Router** on the critical path of queries.

The latency of searching tables < 1ms

② overhead of the **Resource Manager**.

Triggered periodically （needed to set)



MILP can be solved **in 60 seconds：**

a. 160 devices at most

b. 450 model variants at most

c. 17 query types at most

Thinking

- **文章有什么问题，基于这篇 paper 还能做什么优化?**

1. 输入尺寸变化的问题
    - 当前 MILP 优化时，并**未考虑查询输入大小的变化**对决策的影响，尤其在NLP的任务中。
    - Adaptive Batching 会**根据实时排队状况调整 batch size**，但系统整体并未对输入变长进行建模。
2. 公平性问题（trade-off）
    - Proteus 进行精度缩放时是**系统级优化**，即仅关注全局的平均精度。
    - 可能导致**一些请求总是分配到低精度模型**，致使部分“用户”可能感受到不公平。
3. 与硬件协同扩展的问题
    - 当前工作是为了**避免硬件扩展**而提出的精度缩放。
    - 可在短时间内使用精度缩放**吸收突发负载**，同时启动新硬件资源，**待硬件上线后恢复高精度模型。**

Thinking

- **这篇 paper 的 idea 能不能应用在自己的工作上面?**
  - 文中提出了一种MILP的资源管理策略，该方法可以协助在有限资源环境下进行异构计算资源的分配优化，从而加速模型的加速推理过程。
  - 另外，文中在批量处理的过程中采用了一种非工作保持的方法，在不违反SLO的前提下最大化系统吞吐量，这也可以作为边缘端模型推理加速的优化角度之一。
- **这篇 paper 能不能泛化?**
  - 对于多模型推理系统，都几乎可以引入accuracy scaling来替代硬件扩展从而达到可接受的推理吞吐量和模型推理精度。
  - 论文中数据路径和控制路径的解耦分离方法可迁移至对查询响应速度要求较高的推理系统中。

Q & A

Presenter: Yunpeng Xu
2025.5.16