SMIless: Serving DAG-based Inference with Dynamic Invocations under Serverless Computing

C. Lu (Shenzhen Institute of Advanced Technology, CAS; University of Chinese Academy of Sciences; Department of Computer and Information Science, University of Macau), H. Xu, Y. Li, W. Chen (Department of Computer and Information Science, University of Macau), K. Ye (Shenzhen Institute of Advanced Technology, CAS), C. Xu (Department of Computer and Information Science, University of Macau; State Key Lab of Internet of Things for Smart City, University of Macau)

SC, Nov 2024

Presenter: Mausam Basnet, University of North Texas

Date: 2025-03-28

Introduction

Serverless for ML Inference

- Offers elasticity, ease of deployment, and pay-per-use benefits.
- Widely used for ML pipelines involving multiple models.

DAG-Based Workflows

- ML services often form Directed Acyclic Graphs (DAGs) of serverless functions.
- Enables modular, scalable ML inference.

Key Challenges

- **Cold-start latency**, especially on GPUs.
- **Balancing cost vs. latency** with heterogeneous resources.
- **Dynamic workloads** make static provisioning inefficient.

SMIless Motivation

- Existing systems fail to jointly optimize cold-start and resource use.
- SMIless introduces a unified approach for efficient DAG-based ML serving.

Background: ML Serving under Serverless Platform

- **Multiple ML Models in Production**
    - Real-world applications often require several ML models working together.
- **Example: Intelligent Personal Assistant (IPA)**
    - **NLP:** Understanding user requests, **Image Recognition:** Identifying image content, **Question & Answer:** Generating responses, **Text-to-Speech:** Producing audio output.
- **Serverless Platform Structure**
    - Each ML model runs as an independent serverless function.
    - Functions interact to complete complex tasks.
- **DAG-Based Workflow**
    - ML functions form Directed Acyclic Graphs (DAGs).
    - Clearly defines workflow and dependencies among functions.

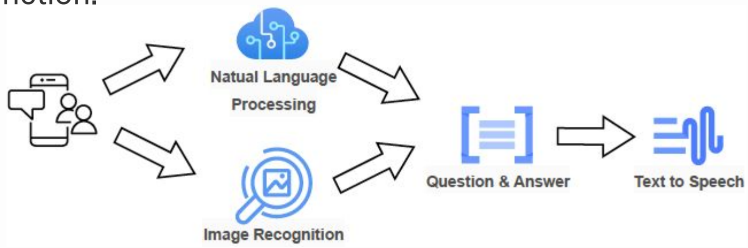

Fig: DAG-based inference

Background: Heterogeneous Serverless Computing

- **Hardware Affects Inference Performance**
    - High-end GPUs vs. low-end CPUs offer distinct latency-cost trade-offs.
- **Latency Example (Warm-start)**
    - GPU (V100) executes Translation (TRS) model ~10× faster than CPU.
    - GPU is ~8× more expensive, yet cost-effective under high usage.
- **Cold-start Overhead**
    - GPU initialization significantly increases initial latency.
    - During cold-start, GPU performance advantage diminishes.

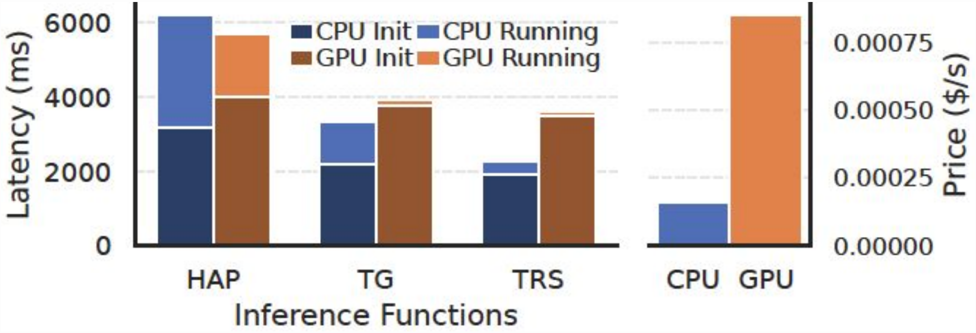

Fig: Inference latency and cost under different hardware.

Challenges

Resource Management

- **Balancing Latency and Cost**
  - Optimal trade-off is difficult due to heterogeneous resources (CPU/GPU).
  - Requires global co-optimization across all DAG functions.

- **Key Issues in DAG-based ML Serving**
  - Cascading Effects: Resource decisions for one function impact subsequent functions.
  - Dynamic Invocation Patterns: Resource configuration optimal for one invocation might not suit subsequent ones.

Challenges

Cascading Effects

- **Resource Choices Affect Subsequent Functions**
  - High inference latency increases potential overlap with the next function's initialization.
  - Proper overlap reduces idle (keep-alive) periods, thus reducing cost.

- **Complex Interdependencies**
  - Overlap duration depends on next function's initialization time.
  - Initialization time itself depends on its resource configuration, further affecting downstream functions.

Challenges

Dynamic Invocation Patterns & Existing Limitations

- **Invocation Pattern Variability**
  - High invocation rate complicates optimal long-term resource decisions.
  - Initial optimal settings might quickly become inefficient under dynamic workloads.
- **Limitations of Existing Solutions**
  - Orion: Assumes perfect overlap, Struggles with frequent invocations, **Cost:** 37.7% above optimal
  - IceBreaker: Manages functions separately, Inefficient global optimization, Cost: 33% above

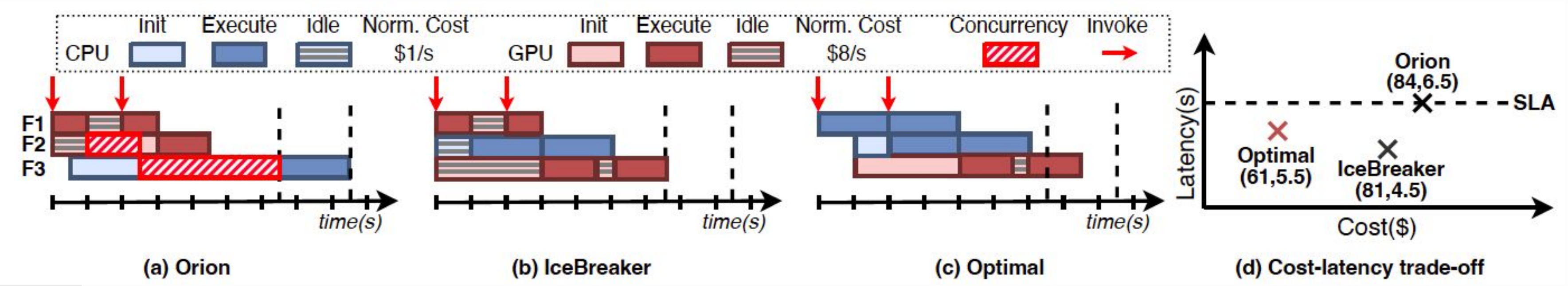

Fig: Limitations of existing approach

SMILESS ARCHITECTURE

System Overview

- Developer submits ML application to the serverless platform.
- Profiles each function's initialization and inference times under various configurations.
- Predicts request arrival patterns and invocation counts or dynamic workload handling.
- Optimizer Engine computes optimal execution strategy considering DAG workflow.
- Auto-scaler dynamically adjusts resources based on predicted workloads and hardware configuration.
- Serverless Container Manager deploys and executes the functions efficiently based on optimized strategies.

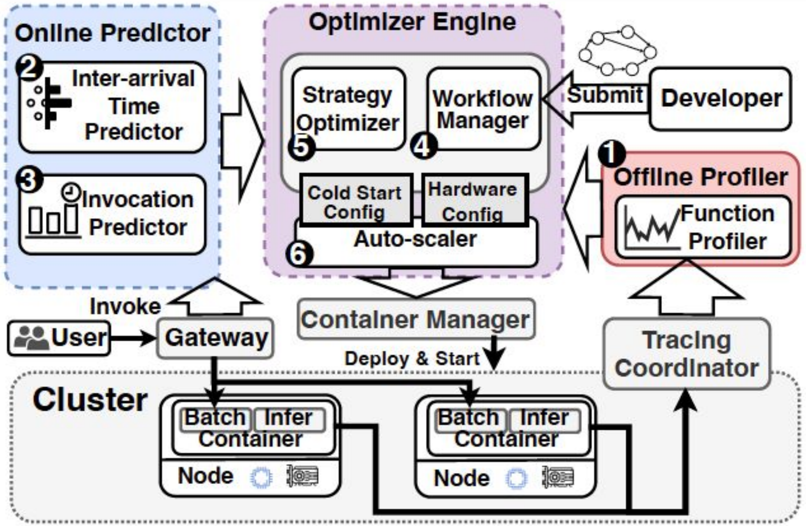

Fig: System architecture of SMIless

SMILESS : Offline Profiling

Prometheus-based event tracking: SMIless accurately records function execution times, hardware configurations, and batch sizes

1. **Profiling initialization time:**

- **Container Initialization:** Downloads container images from remote repositories and initialize on suitable hosts
- **Factors Affecting Initialization Time**
  - Network, PCIe, and memory bandwidth contention cause fluctuations.
- **GPU-specific Overheads**
  - CUDA context setup, GPU memory allocation, model loading.
  - Results in longer initialization time compared to CPUs.

Offline Profiler estimates initialization time robustly using average and standard deviation: (avg + n × std dev).

SMILESS : Offline Profiling

2. **Profiling inference time:**

- **Factors Impacting Inference Time:** Hardware configuration (CPU cores or GPU allocation), Input batch size (B), Container memory allocation

- **Memory Optimization ("Knee Point")**
  - Beyond a specific memory capacity ("knee point"), additional memory yields minimal performance benefits.
  - SMIless allocates memory just above this knee point to prevent wasteful resource allocation

$$\text{Inference time} = \lambda_c \times B \times \left(\frac{\alpha_c}{\#\text{ of CPU cores}} + \beta_c \right) + \gamma_c.$$

$$\text{Inference time} = \lambda_g \times B \times \left(\frac{\alpha_g}{\%\text{ of GPU}} + \beta_g \right) + \gamma_g.$$

SMIless : Online Prediction

SMIless Gateway forwards invocation request to the Online Predictor for counting invocations per application in that time window (1 sec)

1. **Predicting the invocation number:**

- Forecasts future invocation counts using LSTM classification (avoids SLA violations).
- Employs classification rather than regression to prevent underestimation.
- Utilizes past invocation patterns with tailored sequence lengths per application.

2. **Predicting the inter-arrival time:**

- Predicts intervals between invocation requests separately to enhance accuracy.
- Uses dual-input LSTM (past inter-arrival times and invocation counts) to avoid overestimation.
- Combines two data streams, providing precise inter-arrival predictions to effectively manage SLAs.

SMILESS : SMIless Resource Optimization

Co-optimization Framework

- **Objective:** Minimize total cost (initialization, inference, keep-alive) while meeting SLA.
- Optimizes two aspects for each function:
  - Hardware configuration
  - Cold-start management policy
- Formulated as a combinatorial optimization problem (NP-hard).

$$\min_{\{\vec{\chi},\vec{\varphi}\}} \sum_{k=1}^{N} C_k(\star_k, \triangle_k), \quad \text{subject to, } \mathcal{L}(\vec{\chi}, \vec{\varphi}) \leq \text{SLA}.$$

SMILESS : SMIless Resource Optimization

Adaptive Cold-Start Management

- Dynamically adjusts "keep-alive" time based on invocation predictions.
- Balances initialization overhead with resource efficiency.
- Ensures functions remain ready just long enough to handle incoming requests without excessive costs

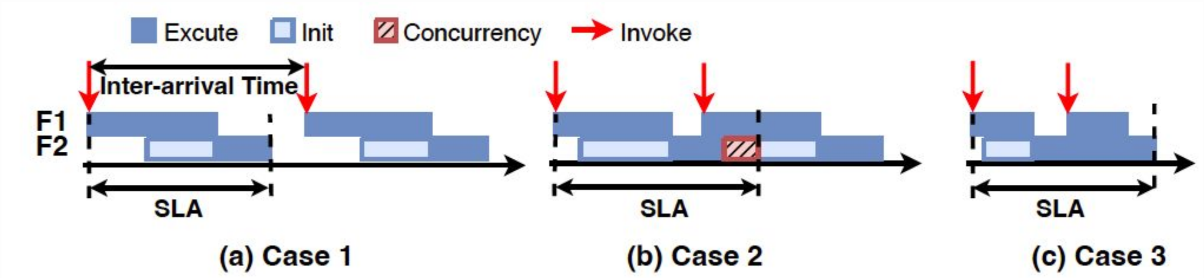

FIg: The pre-warming policies vary in different settings based on the inter-arrival time between successive invocations. The functions F1 and F2 are executed in a pipeline.

SMILESS : SMIless Resource Optimization

Co-optimization Algorithm Design

- **Joint Optimization**
  - Simultaneously selects optimal hardware and cold-start strategies for each function.
- **Efficient Path Search**
  - Utilizes a multi-way tree to explore configuration combinations systematically.
- **Top-K Strategy Selection**
  - Quickly identifies near-optimal solutions by pruning less promising options.

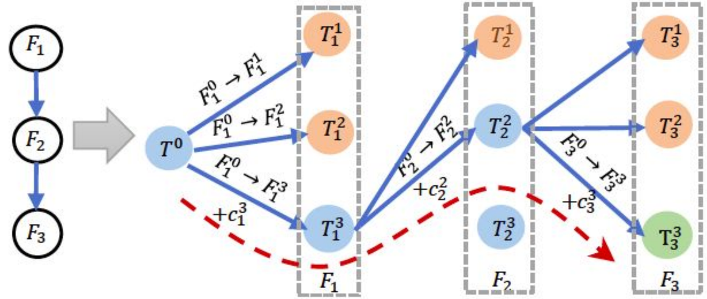

Fig: The path-search process where the orange (green) node indicates the E2E latency violates (meets) SLA.

SMILESS : SMIless Resource Optimization

Container Autoscaling

- **Dynamic Resource Adjustment**
  - Scales containers based on predicted workloads and resource usage.
- **Responsive to Burst Loads**
  - Quickly adapts resources to handle sudden increases in invocation requests.
- **Cost-efficient Scaling**
  - Balances scaling decisions to maintain SLA compliance without excessive costs.

System Implementation

SMIless Implementation

- Built on OpenFaaS with Kubernetes orchestration.

Testing Environment

- 8-node cluster setup.
- Each node: High-performance CPU, NVIDIA GPU (RTX 3090).

Resource Management

- NVIDIA Multi-Process Service (MPS) enables GPU sharing.
- Containers managed efficiently via Kubernetes.

Workload Simulation

- Realistic workloads derived from Azure Function traces.

- **Applications**

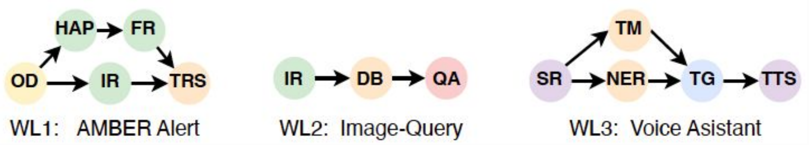

Fig: ML serving applications with DAG workflows

- **Baselines**
  - GrandSLAm
  - IceBreaker
  - Orion
  - Aquatope

Evaluation : E2E Performance

Cost and Latency

- **Cost Efficiency:**
  - SMIless achieves near-optimal execution costs.
  - Up to 5.73× cost reduction compared to IceBreaker.
- **Latency & SLA Compliance**
  - Consistently meets SLA (0% violations).
  - Outperforms other methods significantly.

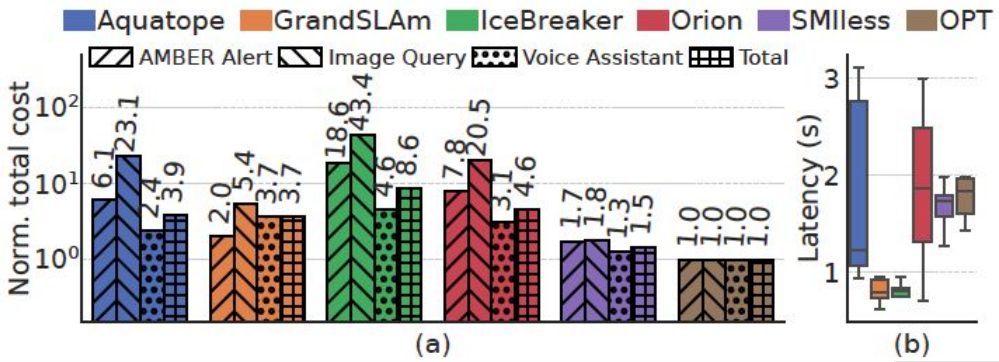

Fig: (a) Overall execution cost. (b) Distribution of the E2E latency (*OPT denotes optimal policy*)

Evaluation : E2E Performance

Hardware and Cold-start Management Comparison

- **Resource Usage:**
  - SMIless balances CPU-GPU usage effectively, optimizing cost-performance.
- **Container Re-initialization:**
  - Aquatope and Orion have high reinitialization rates (~40%), causing SLA violations.
  - SMIless significantly reduces unnecessary reinitializations, maintaining lower cost and SLA compliance.

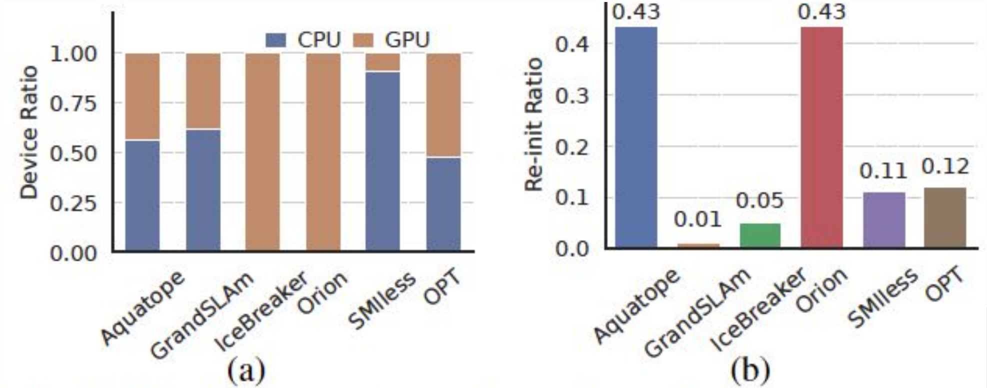

Fig: Hardware configuration and cold-start management across different systems.
(a) The ratio of CPU to GPU usage. (b) Fractions of container reinitialization.

Evaluation : E2E Performance

Impact of SLA Settings on Performance

- **Execution Cost:**
  - SMIless consistently achieves the lowest execution cost across SLA settings.
- **SLA Violation:**
  - Orion and Aquatope incur high SLA violations under tight SLA conditions.
  - SMIless maintains 0% violations, ensuring stable performance and cost.

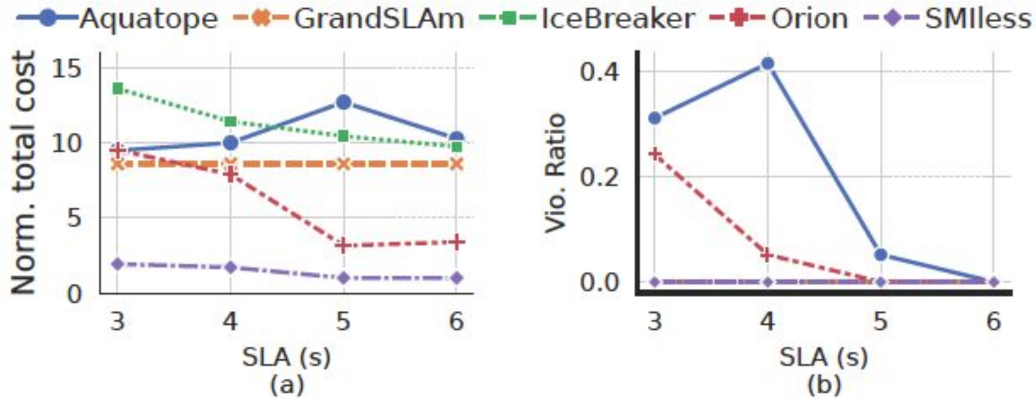

Fig: The E2E performance under different SLA settings. (a) Total execution cost. (b) SLA violation ratio

Evaluation : Source of Improvement

- Impact of Offline profiling

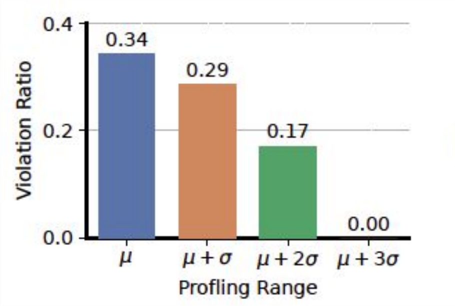

Fig: Offline profiling results under SMIless

- Impact of Co-optimization

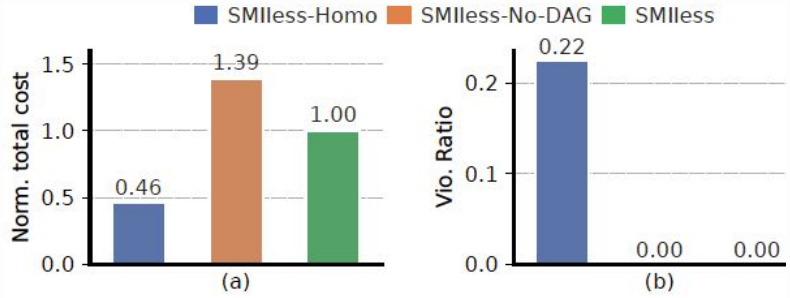

Fig: The advantage of co-optimization. SMIless-Homo only launches containers with CPU backend while SMIless-No-DAG starts all functions simultaneously

Discussion & Conclusion

- **Effective DAG Optimization:**
  - Jointly optimized hardware configuration and adaptive cold-start management.
- **Cost and Performance:**
  - Achieved significant cost reduction (up to 5.73×) compared to existing solutions.
  - Consistently maintained SLA compliance (0% violations).
- **Stable and Scalable:**
  - Demonstrated stable performance under varied SLA settings.
  - Effectively handled dynamic invocation workloads.
- **Future Directions:**
  - Explore multi-tenant scenarios and stateful workflows.
  - Extend support to other hardware accelerators for broader applicability.

Thanks!

Do you have any questions?