INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

52nd International Conference on Parallel Processing (ICPP 2023)

FaST-GShare: Enabling Efficient Spatio-Temporal GPU Sharing in Serverless Computing for Deep Learning Inference

Jianfeng Gu, Yichao Zhu, Puxuan Wang, Mohak Chadha, Michael Gerndt

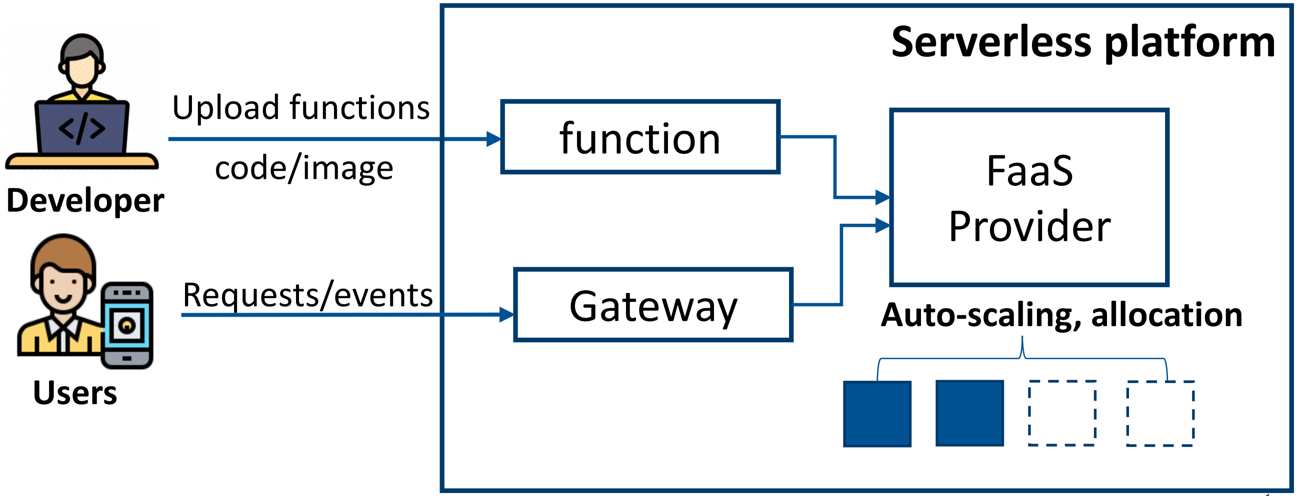
Chair of Computer Architecture and Parallel Systems (CAPS)
Technical University of Munich

Outline

- Background
- Motivation
- Architecture & Implementation
- Evaluation
- Conclusion

Background – Serverless Computing

- Serverless – Function as a Service (FaaS)
- Advantages
  - Easy deployment;
  - Pay-per-use, Cost-effectiveness
  - Auto-scaling, high scalability;
  - Event-driven, fine-grained.



- Popularity
  - Lambda, Azure Functions, Google Cloud Functions;
  - Problem: only CPU-supported.
  - Serverless DL Inference.
    OSDI22 Orin [10], *ATC22 Jie Li, et. al.* [11],
    VLDB22 Ahsan Ali, et. al. [12].

Only CPU

AWS Lambda

Azure Functions

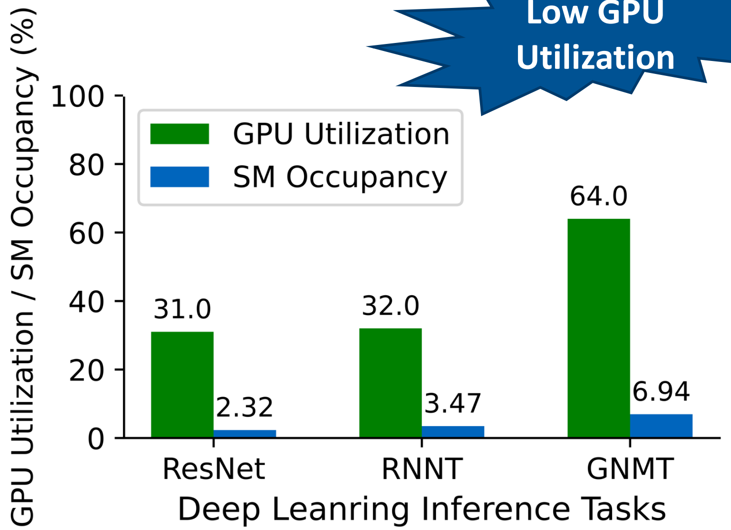
Google Cloud Functions

Motivation – Coarse GPU Usage in Serverless

- GPU Device Plugin for **Kubernetes** [1]
  - Designed for **DL training**, exclusive GPU usage for a container;
  - **GPU under-utilization** in DL inference;

- Time-Slicing GPUs in Kubernetes [2]
  - **Fair time-sharing**, Not guarantee GPU compute power for each pod.
  - Lack of resource **isolation**, Unpredictable interference among FaaS functions.
  - Service Level Objectives (SLOs) violations.

Low GPU Utilization



Kubernetes Device Plugin

NEED!

Both **efficient** and **SLO-aware** GPU sharing mechanism.

Motivation – Fine-grained GPU Sharing

- High GPU utilization == Efficient GPU Usage ?
  - GPU utilization and **SM Occupancy**. (Streaming Multiprocessor)
  - NVIDIA V100 GPU: 80 SM units & A100 HGX GPU: 108 SM units.

- Timing Sharing
  - Previous work: *KubeShare* [38], *Gaia-GPU* [4], *vGPU-scheduler*[5], et. al;
  - **Low SM Occupancy**;

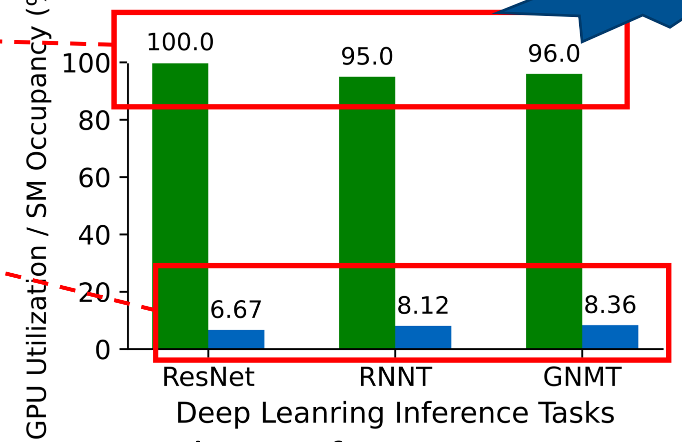
- Spatial Sharing
  - vGPU from **Multi-Instance GPU (MIG);**
    Limited to 7 pre-defined resource configurations;
  - Previous work: *GSLICE* [6], *gpulets* [7], et al.
    Resource-based scaling not compatible with **Instance-based scaling** in FaaS.
  - *Tencent Cloud qGPU* [8] and *Aliyun cGPU* [9]; **Only Spatial Shairng**
    No ***request*** and ***limit*** mechanism**,** not flexible to utilize idle resources.

NVIDIA V100

Low SM Occupancy

GPU Utilization

SM Occupancy



Time sharing from KubeShare[3].

Objectives & Challenges

- Objectives
  - **Spatio-Temporal** GPU Sharing; (Performance Isolation)
  - **SLO-aware**; (Guarantee Function SLOs).
  - **Efficient**; (Throughput, GPU Utilization, and SM Occupancy).

- 4 Challenges
  - How to **coordinate** GPU spatial and temporal multiplexing for performance isolation and efficiency?
  - How to **allocate** the appropriate amount of spatio-temporal resources for each FaaS function?
  - How to **schedule** functions across GPU nodes to ensure its SLO, and improve throughput and GPU occupancy?
  - How to **alleviate** the GPU memory contention with more FaaS functions sharing a GPU?

FaST-GShare

FaaS-oriented Spatio-Temporal GPU Sharing architecture.

Contribution

FaST-GShare

FaST-Manager

- How to **coordinate** GPU spatial and temporal multiplexing to ensure both efficiency and isolation?

FaST-Profiler

- How to **allocate** the appropriate amount of spatio-temporal resources for each FaaS function?

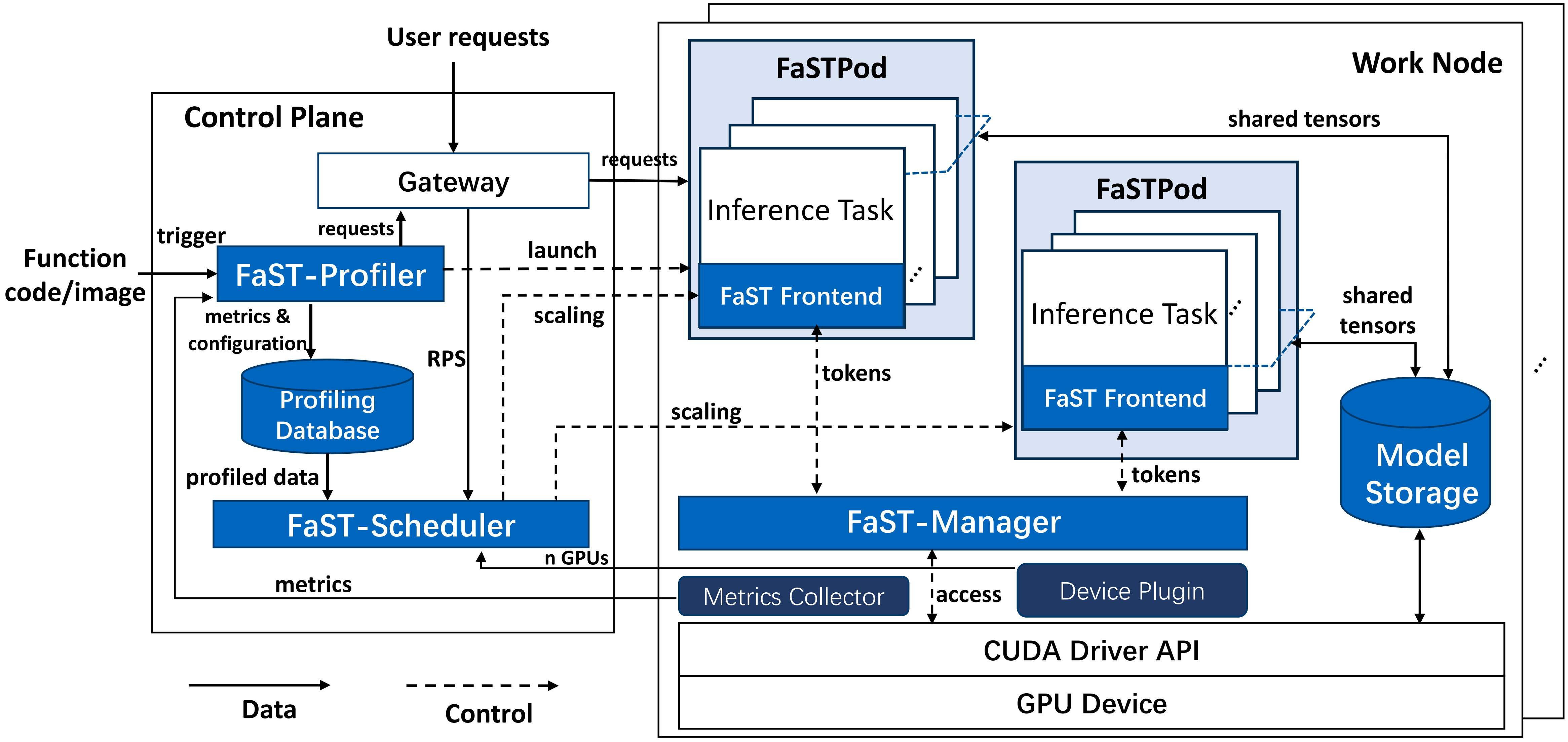
FaST-Scheduler

- How to **schedule** functions across GPU nodes to ensure its SLO and throughput while improving GPU resource usage?

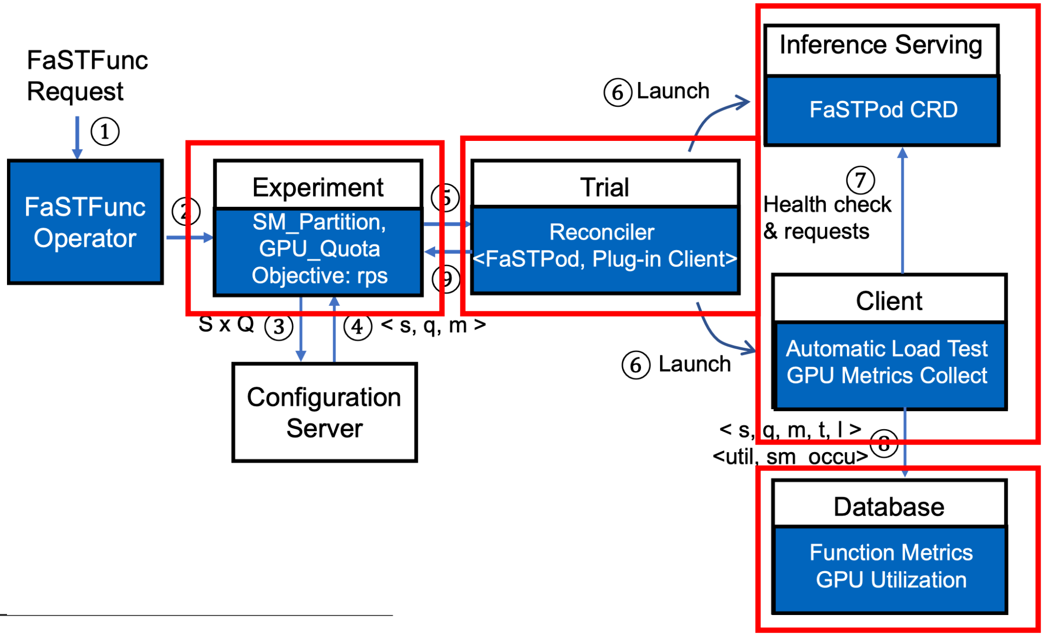
Model Sharing

- How to **alleviate** the increased contension of GPU memory with more FaaS functions sharing a GPU?

Architecture – FaST-GShare Overview



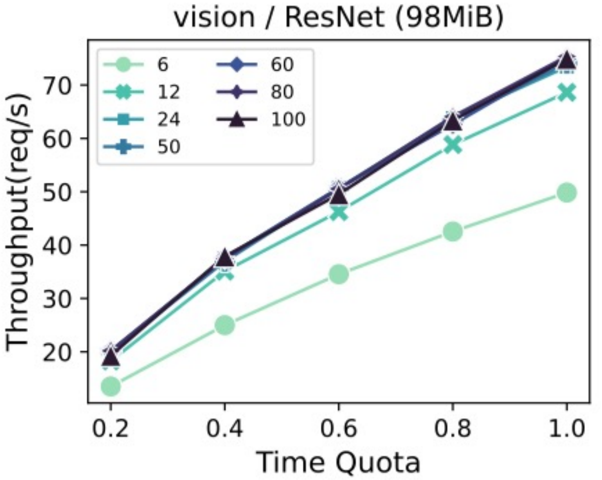
Architecture – FaST-Profiler



- Profiler
  - **Automatic**; **Profile** the function **throughput** under **various spatio-temporal** GPU resource allocations; (Function Performance Curves)
  - **Automatic** Experiment-Trial workflow;

- FaSTPod
  - Basic Deployment Unit;
  - **Any granularity** of spatial and temporal GPU resources.

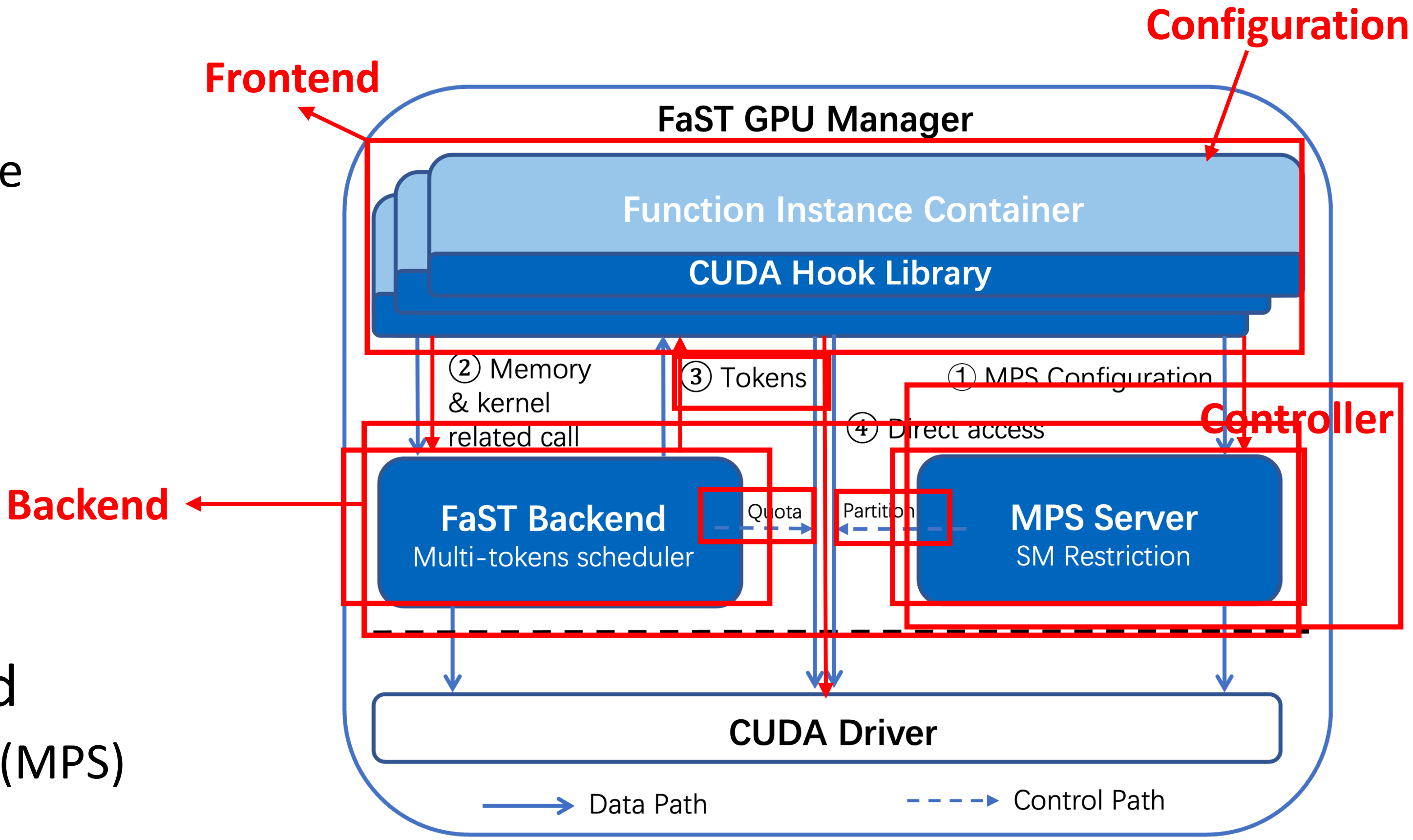
```yaml
apiVersion: faasshare.com/v1
kind: FaSTPod
metadata:
  annotations:
    faasshare/sm_partition: "12"
    faasshare/quota_limit: "0.8"
    faasshare/quota_request: "0.3"
    faasshare/gpu_mem: "1073741824"
  name: fastsvc-rnnt-q30-p12
spec:
  podSpec:
    containers:
    - env:
      - name: MODEL_NAME
        value: MLPerf-FaaS-rnnt
      image: xxxx/mlperf-faas-rnnt:latest
  replicas: 2
```



Function Performance Curves

Architecture – FaST-Manager

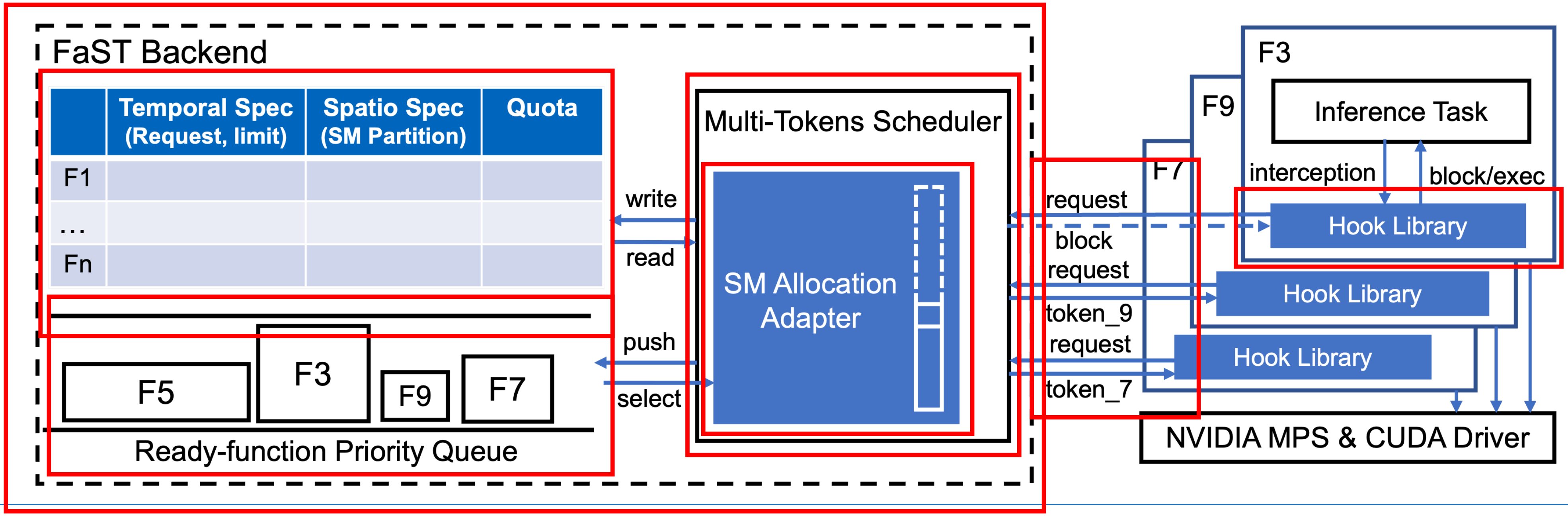
- FaST-Manager
  - **Coordinate** and **Isolate** GPU resource in spatial and temporal dimensions.
  - **Frontend-Backend** mechanism;
  - Tokens: Time slices;



- MPS-based Spatial Sharing Backend
  - Controller for Multi-Process Service (MPS)
  - Exposes the IPC namespace.
  - *CUDA_MPS_ACTIVE_THREAD_PERCENTAGE;*

Architecture – FaST-Manager

- FaST Time Sharing Backend
  - CUDA Driver API **Interception,** Linux *LD_PRELOAD*.
  - Multi-tokens Scheduler;
  - Control Spatial multiplexing through timing tokens → the spatio-temporal limitation collaboration;



Architecture – FaST-Scheduler

- FaST-Scheduler
  - Scale and schedule functions across GPU nodes to guarantee SLOs and improve GPU efficiency.
  - 1. Heuristic Auto-Scaling;
  - 2. Maximal Rectangles Algorithm for node selection;

- Heuristic Auto-Scaling
  - Utilize Profiling data. Decide the number of function pods and corresponding resource allocations;
  - Scaling-up and Scaling-down;
  - A metric RPR;
  - Priority queue in the ascending order by RPR.

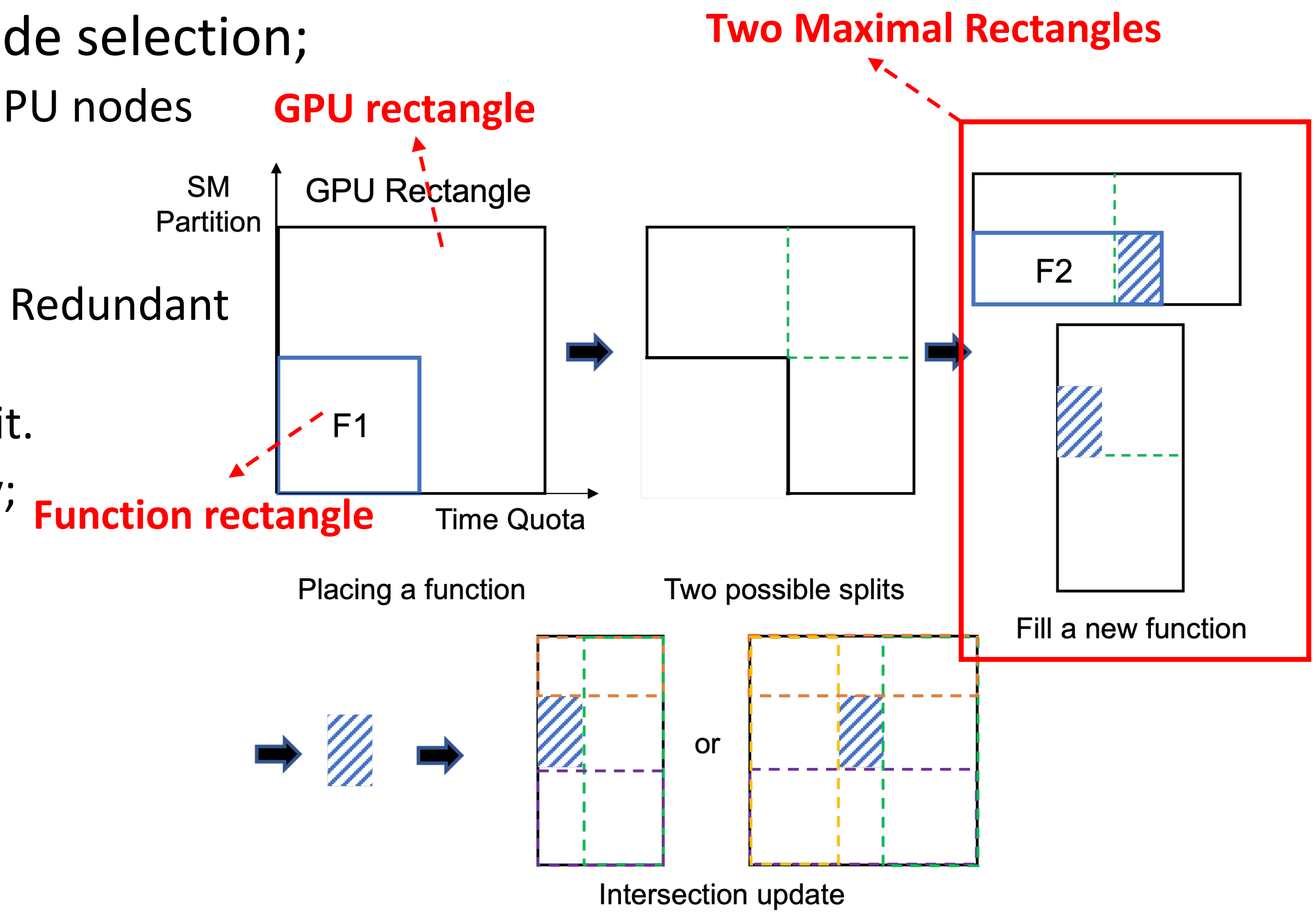
Processing gap

$$\boxed{\Delta RPS_j} = R_j - \left(\sum_{J_i \in F_j} T_{j,i}\right)$$

$$\boxed{RPR \text{ (RPS per Resource)}} = \frac{T_{j,p}}{S_{j,p} * Q_{j,p}}$$

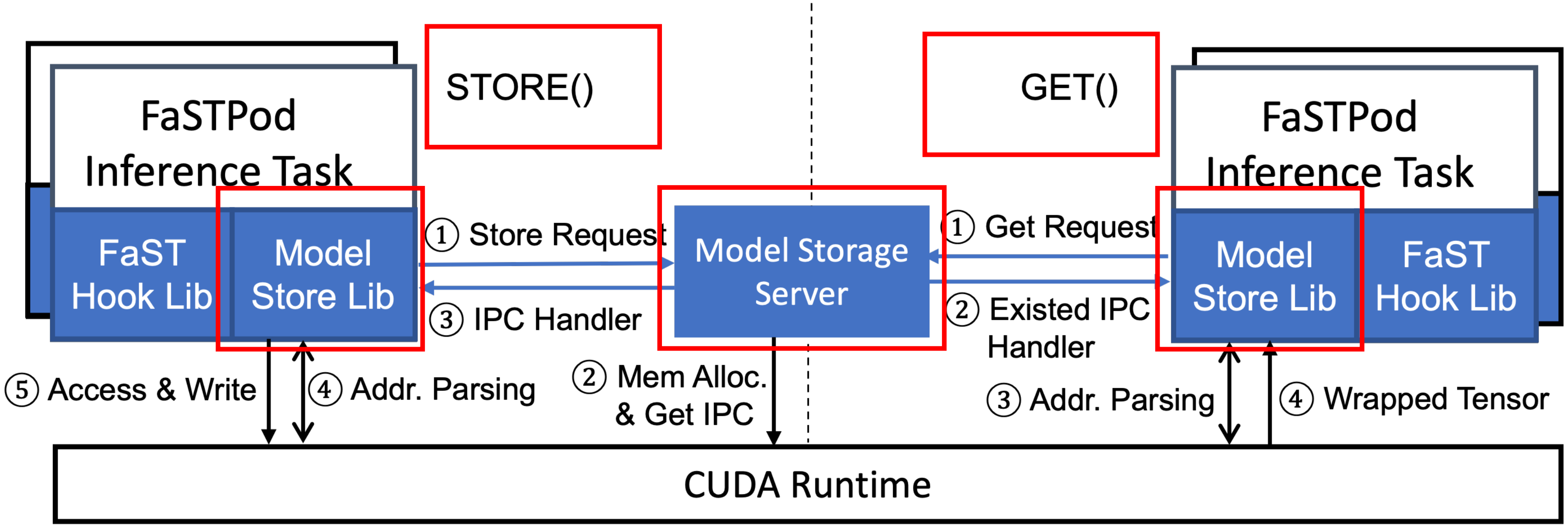
Architecture – FaST-Scheduler

- Maximal Rectangles Algorithm for node selection;
  - Effectively schedule function pods to GPU nodes
  - Dynamic 2D resource **restructuring**.
  - **GPU rectangle** and **function rectangle**.
  - 3 Steps: Split, Intersection update, and Redundant rectangle removing;
  - Two maximal rectangles during the split.
  - Reclamation: "keep-restructure" policy;



Architecture – Model Sharing

- Model Sharing Storage
  - Mitigate memory contention.
  - Model Store Library and Model Storage Server.
  - STORE() and GET().



Experiment Setup

- Testbed
  - Serverless platform: **OpenFaaS** with **faas-netes**.
  - The master node: Intel(R) Xeon(R) CPU @ 2.00GHz with 48 cores and 60GB RAM.
  - **4 work nodes**, each with a **NVIDIA Tesla V100 GPU** with 80 SM units, and **16GB device memory**.

- Workload
  - **MLPerf Benchmark** [14] from NVIDIA, including **ResNet**, **BERT**, **RNNT**, and **GNMT**;
  - Larger transformer models: **ResNeXt** [37] (vision) and **ViT_huge**[16];
  - ML framework: **PyTorch** and **TensorFlow**;

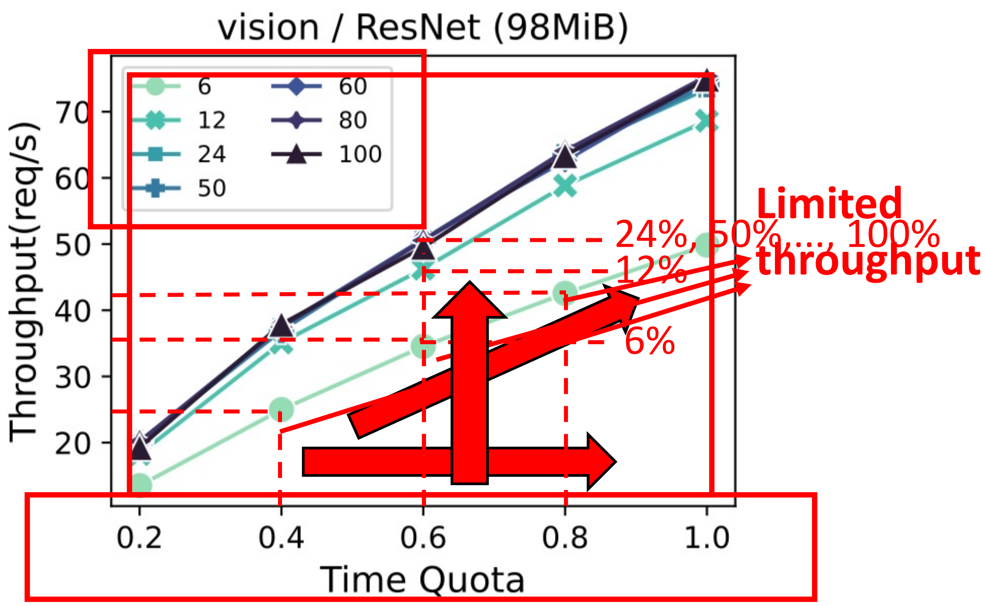
Evaluation – Performance Isolation

- Profiling points
  - Temporal dimension: 0.2, 0.4, 0.6, 0.8, 1.0.
  - Spatial dimension: 6%, 12%, 24%, 50%, 60%, 80%, 100%;

- Temporal dimension
  - The throughput of the model increases proportionally.
  - **Limited throughput →** **Effective temporal resource isolation.**

- Spatial dimension
  - At a certain point, the throughput will no longer increase.
  - → **A model cannot fully occupy all SMs**;
  - → **Effective spatial resource isolation;**

- → Profiling performance curves for FaST-Scheduler



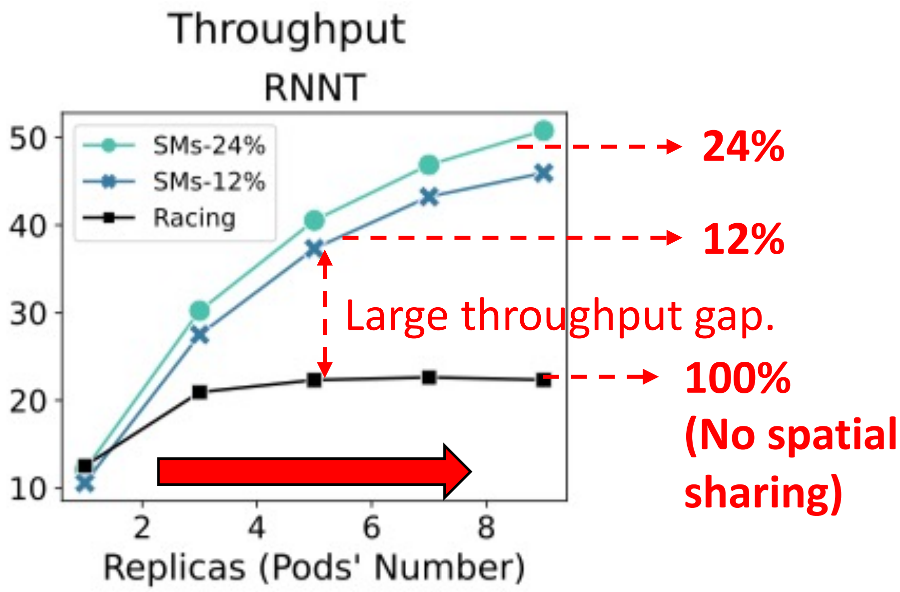
Function throughput of ResNet from FaST-Profiler.

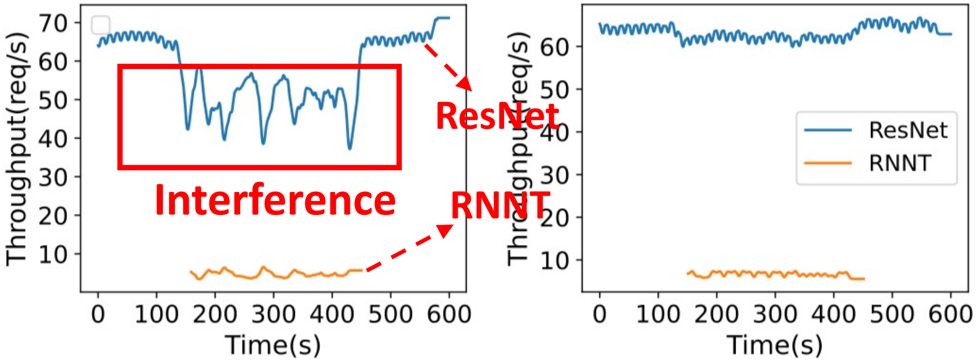
Evaluation – Spatial Sharing Performance

- 100% time quota allocation
  - No spatial sharing (racing, 100% partition).
  - 12% partition;
  - 24% partition;

- **Throughput Improvement:**
  - **Improve throughput by x3.15.**

- **Avoid Interference**
  - Time quotas (limit):
ResNet: 80%, RNNT: 50%.   >100%
  - **Effectiveness of spatial sharing in FaST-Manger.**



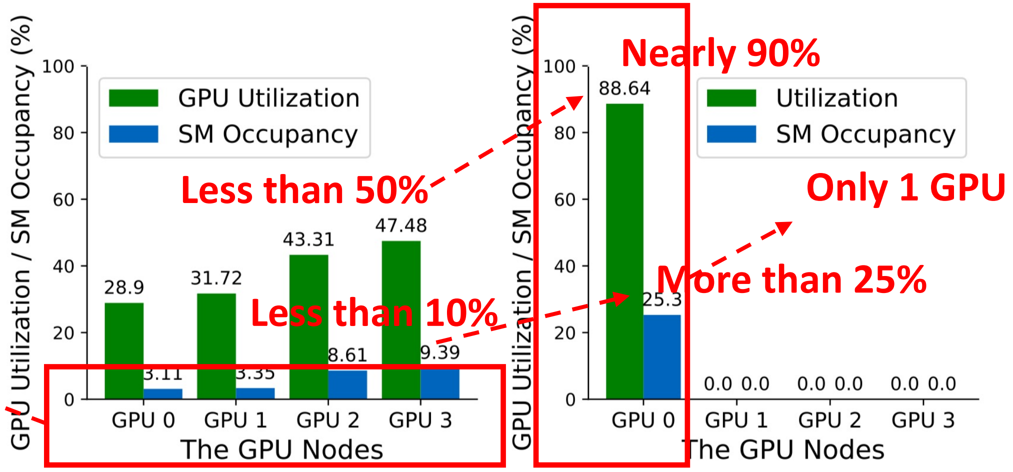


(a) Only time sharing. **(b) Spatio-Temporal sharing.**

Evaluation – Efficient FaST-Scheduler

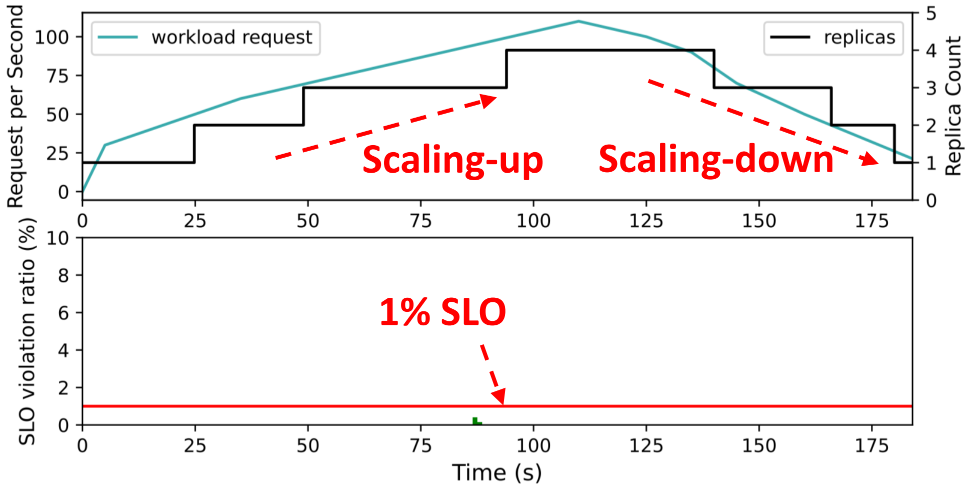
- Workload (SM partitions, time quotas)
  - 4 ResNet pods, (12%, 40%)
  - 2 RNNT pods, (24%, 40%)
  - 2 BERT pods, (50%, 60%)

4 GPUs



(a) Only time sharing [38]. (b) FaST-Scheduler.

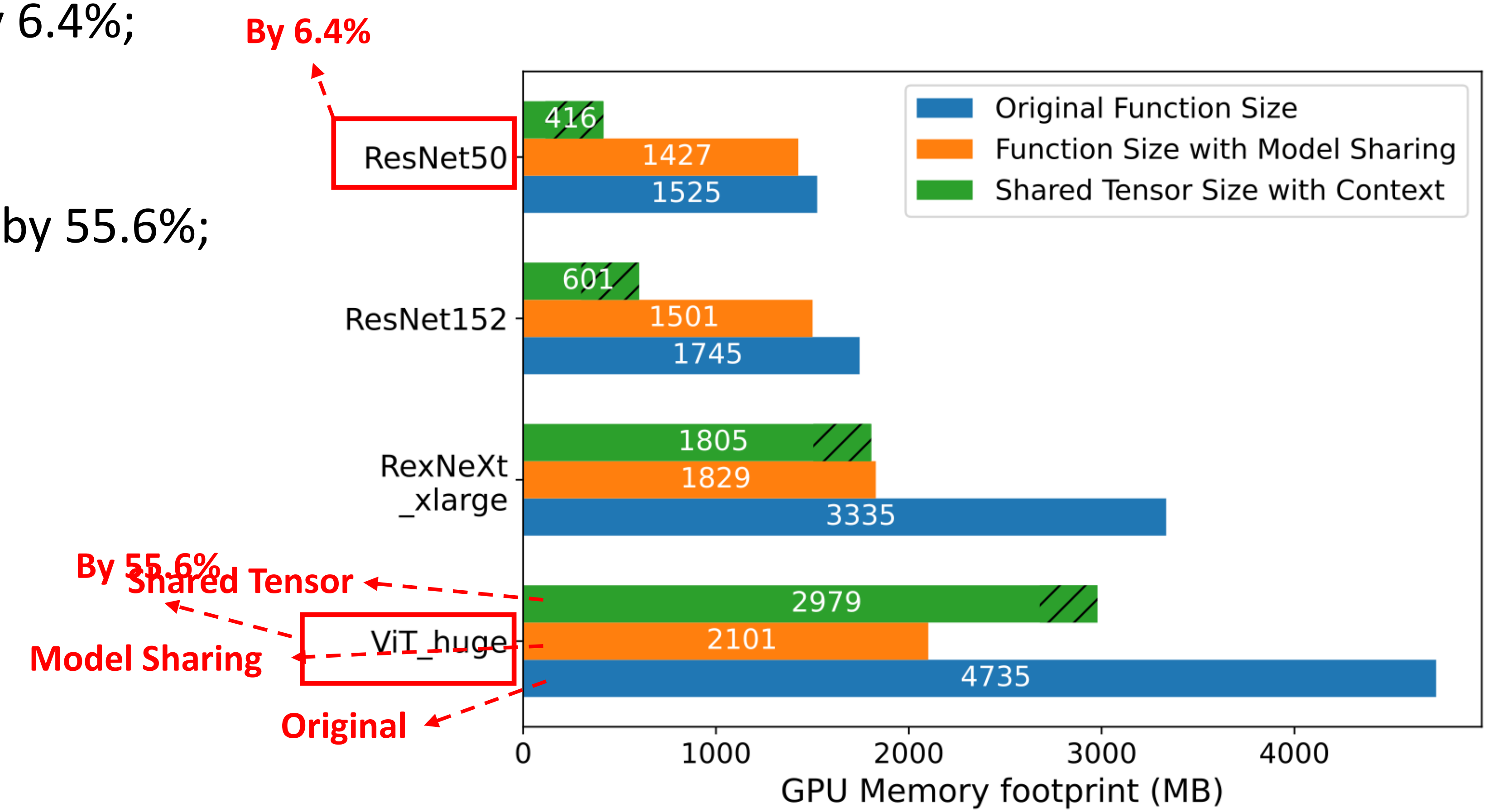
- **GPU utilization and SM occupancy Improvement**
  - **4 GPUs → 1 GPU**
  - **GPU utilization by x1.34,**
    SM occupancy by x3.13 on average;



- **SLO-Aware**
  - Effective Auto-scaling;
  - ResNet SLO of 69ms, **no SLO violation** exceeding 1% .

Evaluation – Model Sharing

- ResNet model decreased by 6.4%;
  - 1525M to 1427M
- ViT_huge model decreased by 55.6%;
  - 4735M --> 2101M



Conclusion

- **FaST-GShare**
  - An efficient FaaS-oriented Spatio-Temporal GPU Sharing architecture for deep learning inferences;
- **FaST-Manager**
  - Limit and isolate spatio-temporal resources for GPU multiplexing.
- **FaST-Profiler** & **FaST-Scheduler,**
  - Guarantee function SLOs and maximize GPU usage;
- **Model Sharing**
  - Alleviate Memory Contention;
- **Performance**
  - Improve throughput by 3.15x, GPU utilization by 1.34x, and SM occupancy by 3.13x on average.