Transparent GPU Sharing in Container Clouds for Deep Learning Workloads

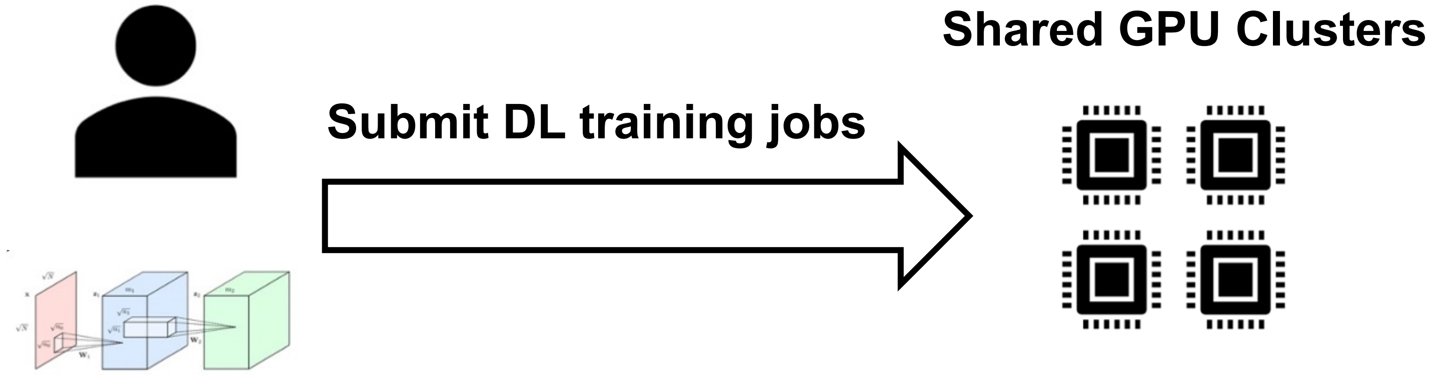
Bingyang Wu, Zili Zhang, Zhihao Bai, Xuanzhe Liu, Xin Jin

PEKING UNIVERSITY

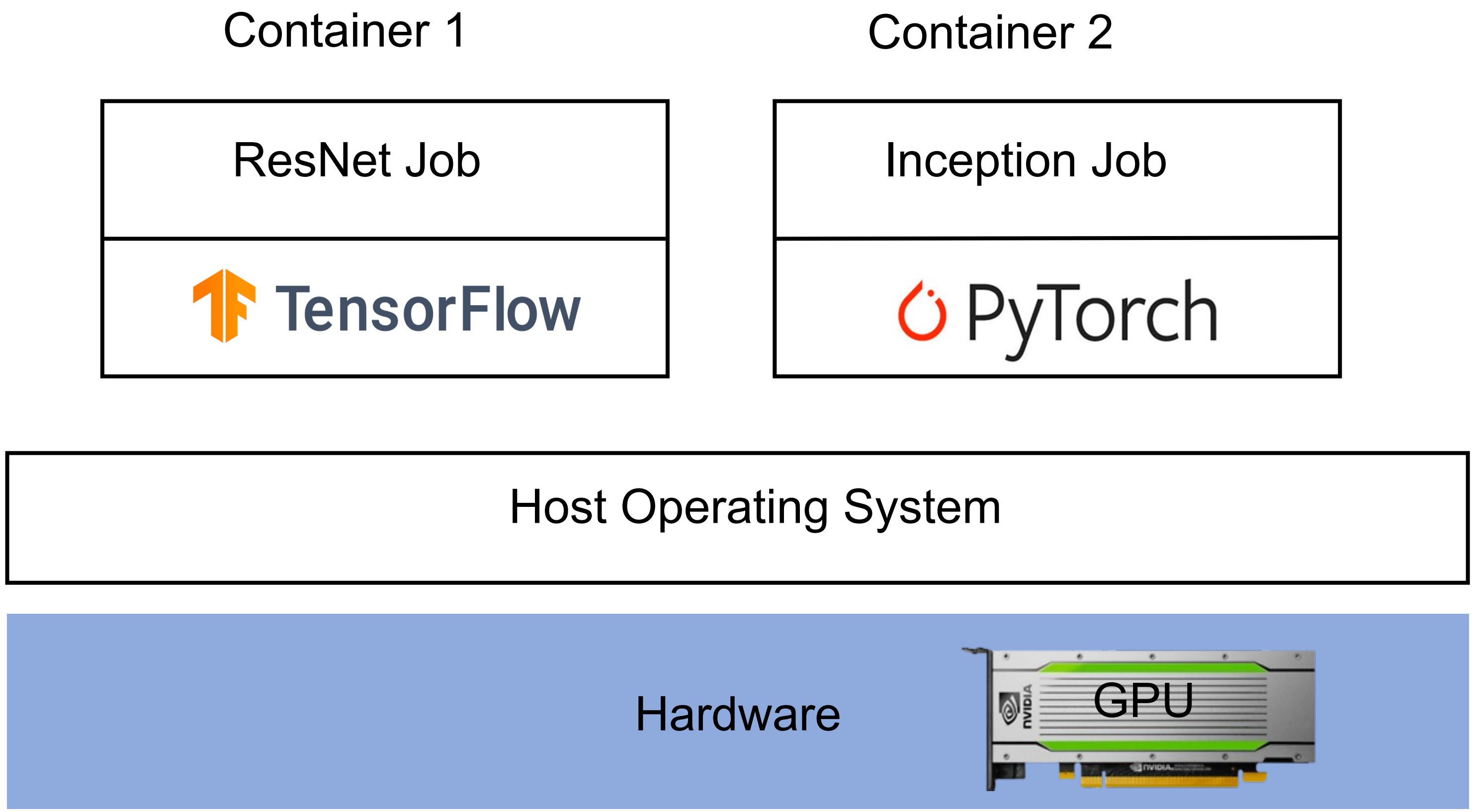
JOHNS HOPKINS UNIVERSITY

Deep learning training jobs: important workloads in datacenters

- Deep learning is widely used in many applications
  - Recommendation
  - Machine Translation
  - Voice Assistant
  - ……
- Deep learning models are often trained in shared GPU clusters



Deep learning training jobs in container clouds



Low GPU utilization in production

- Microsoft [1]: the average GPU utilization is only 52%
- Alibaba [2]: the median GPU utilization is no more than 10%
- Low GPU utilization is bad
  - Container clouds: idle GPUs are a huge waste
  - Users: longer queueing delay, longer job completion time

- **Root cause:** Each GPU is statically assigned to a single container

[1] M. Jeon, et al., "Analysis of large-scale multitenant GPU clusters for DNN training workloads," in *USENIX ATC 2019*.
[2] W. Xiao, et al., "Antman: Dynamic scaling on GPU clusters for deep learning," in *USENIX OSDI 2020*.

Existing GPU sharing solutions

- **Key idea:** Share GPUs to improve GPU utilization
- Classify DLT jobs into two classes
  - Production job: Run without performance degradation
  - Opportunistic job: Utilize spare GPU resources to execute

- SOTA solutions:
  - Application-layer solution：AntMan [OSDI’ 20]
  - OS-layer solution: NVIDIA MPS, NVIDIA MIG

Application-layer solution: AntMan

- Custom DL framework
  - Modify TensorFlow (~4000 LoC) or PyTorch (~2000 LoC)
- Support GPU compute sharing and GPU memory oversubscription

- **Limitations:** Lack of Transparency
  - **Limited use cases:** restricts users to use particular frameworks
  - **Huge operation overhead:** need to maintain custom frameworks

OS-layer solution: NVIDIA MPS

- A software solution for GPU sharing provided by NVIDIA
- Limitations:
  - Low GPU utilization
    - Does not support GPU memory oversubscription
    - Requires application knowledge to properly set the resource limit
  - Weak fault isolation
    - When a job fails, other jobs may be affected and even fails

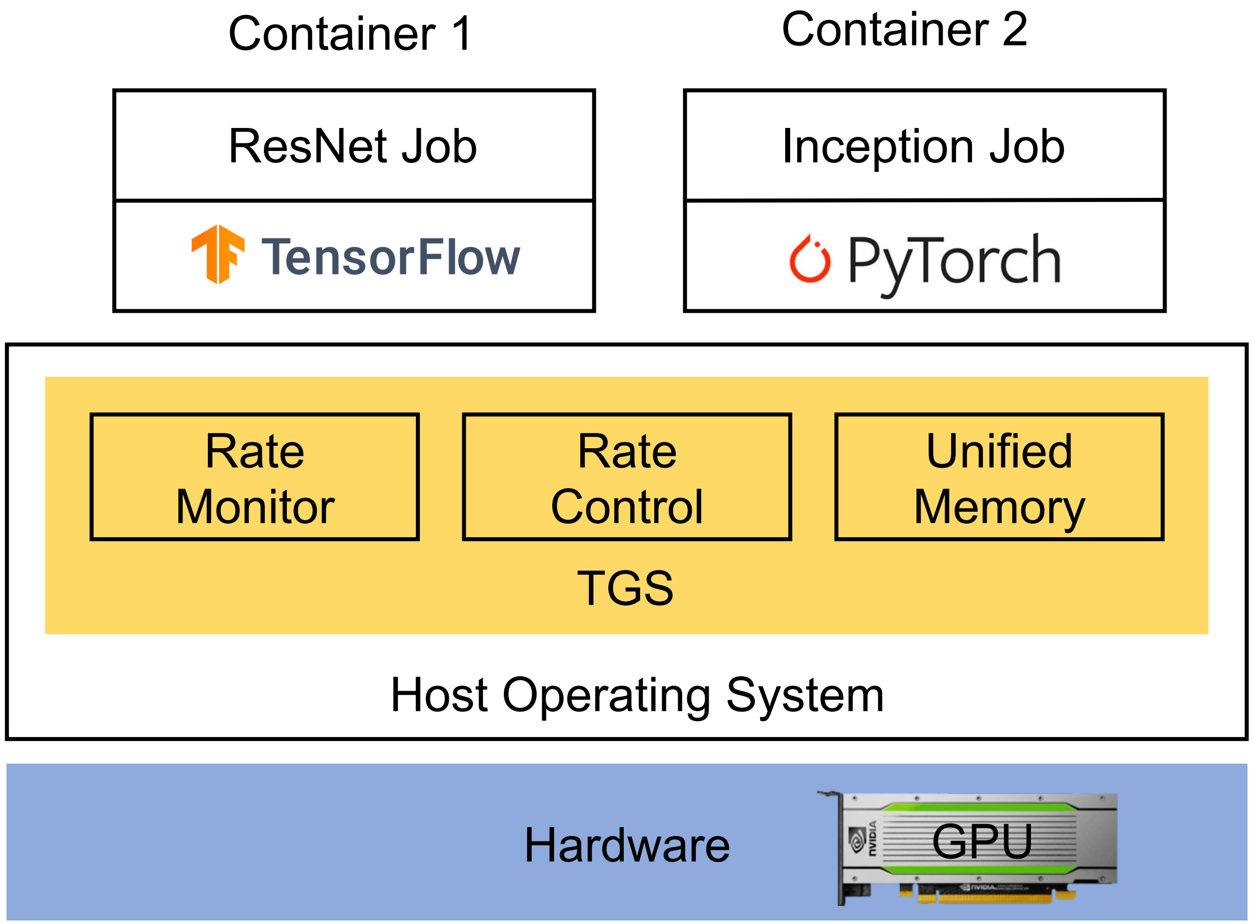
OS-layer solution: NVIDIA MIG

- A recent hardware solution for GPU sharing provided by NVIDIA
- Limitations:
  - Performance isolation
    - Cannot arbitrarily partition a GPU
    - Cannot dynamically change GPU resources
  - Compatibility
    - Only available on a few high-end GPUs
    - Does not support GPU sharing for the multi-GPU instance

A more practical solution: TGS

	AntMan	MPS	MIG	TGS
Transparency		✓	✓	✓
High utilization	✓			✓
Performance isolation	✓	✓	✓	✓
Fault isolation	✓		✓	✓

TGS architecture

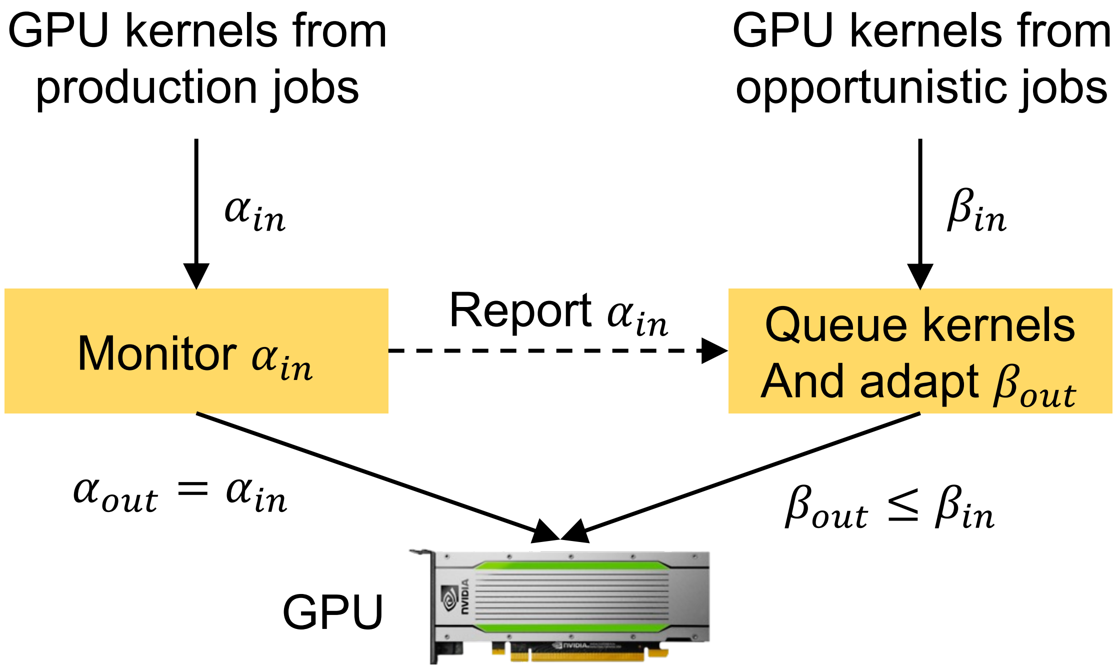


Sharing GPU compute resources

- Strawman solution: priority scheduling
  - Control the opportunistic job based on the GPU kernel queues

- Low GPU utilization:
  - The state of queues do not reflect the remaining GPU resources

Adaptive rate control of TGS



Sharing GPU memory resources

- Weak Fault isolation: total GPU memory consumption may exceed GPU memory capacity and cause OOM
- Low GPU utilization: some jobs always claim all GPU memory

- Application-layer technique cannot be used in the OS layer
  - Cannot directly ask DL framework to release unused GPU memory
  - Cannot directly change pointer address from GPU memory to host memory

Transparent unified memory of TGS

- **Key ideas:** leverage CUDA unified memory to transparently unify GPU memory and host memory

- High GPU utilization: The actual physical GPU memory is allocated when jobs first access to them
- Fault isolation: When GPU memory is oversubscribed, TGS changes virtual memory mapping to evict GPU memory of opportunistic job to host memory

Evaluation setup

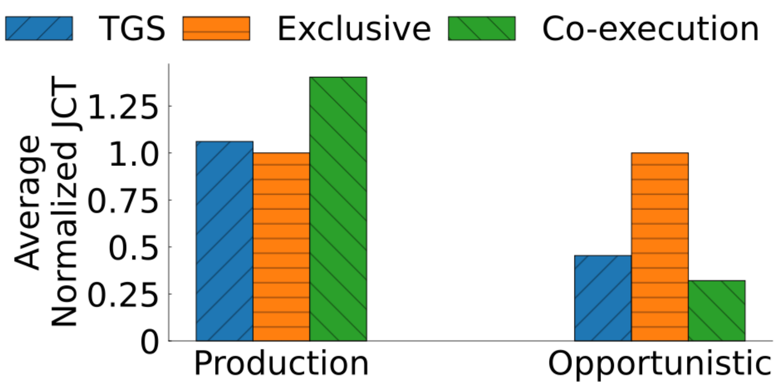
- Implementation: ~3000 LoC C++ & Python
  - Integration with Docker and Kubernetes
- Testbed: NVIDIA A100 GPUs and NVIDIA V100 GPUS
- Trace: Philly Trace from Microsoft [Jeon et al. 2019]
- Models
  - CV: ResNet, ShuffleNet, MobileNet
  - Graph: GCN
  - NLP: Bert, GPT-2
  - Recommendation: DLRM

Evaluation baselines

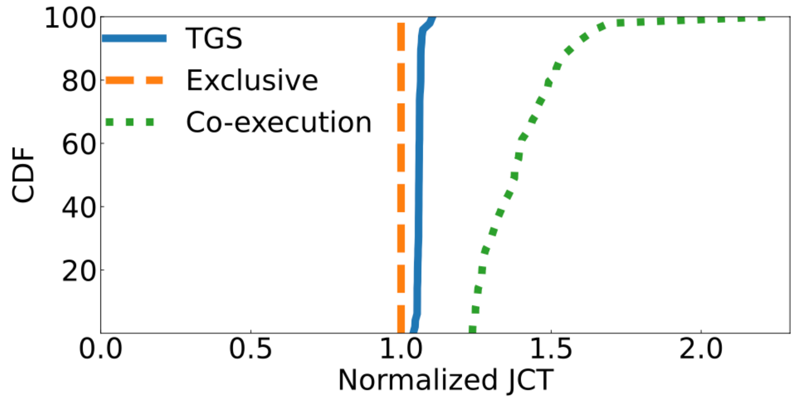
- TGS: our work
- AntMan: the state-of-the-art application-layer solution
- MPS: manually set appropriate limit
- MIG: manually set best configuration
- Exclusive: give exclusive access to a GPU
- Co-execution: share a GPU without any control

Mixed workload job stream

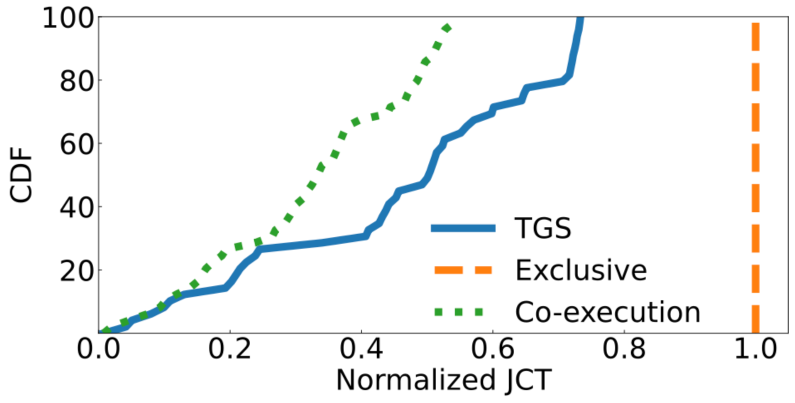
- A job stream contains 50 production jobs and 50 opportunistic jobs
- Opportunistic jobs: 52% JCT reduction compared to Exclusive
- Production jobs: 21% JCT reduction compared to Co-execution



(a) Average JCT.



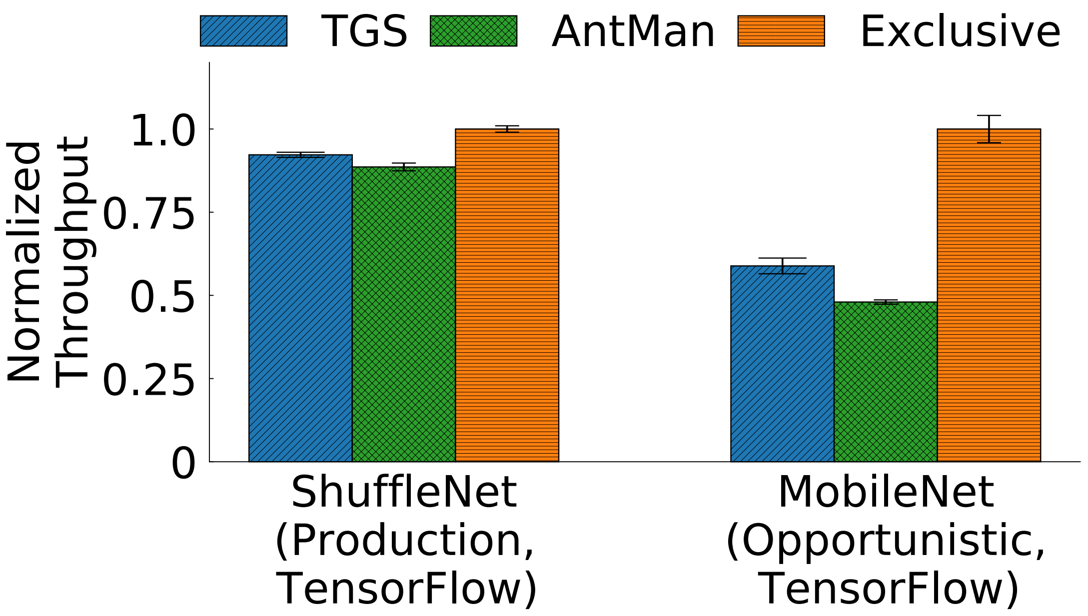
(b) CDF of production jobs.



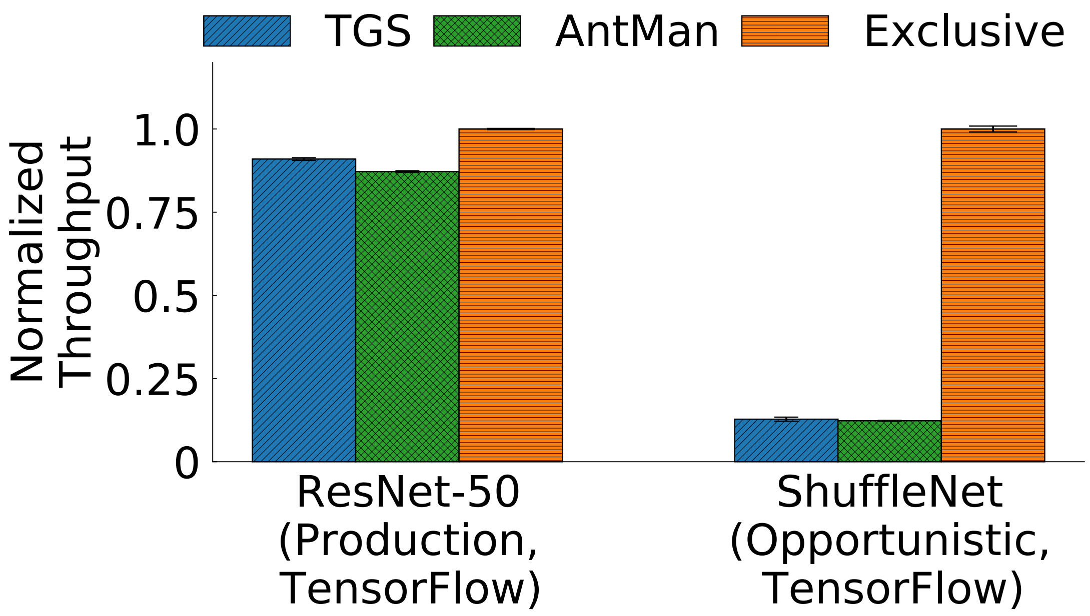
(c) CDF of opportunistic jobs.

Comparison with AntMan

- Achieve comparable performance in different contention scenarios
- Provide transparency without sacrificing performance



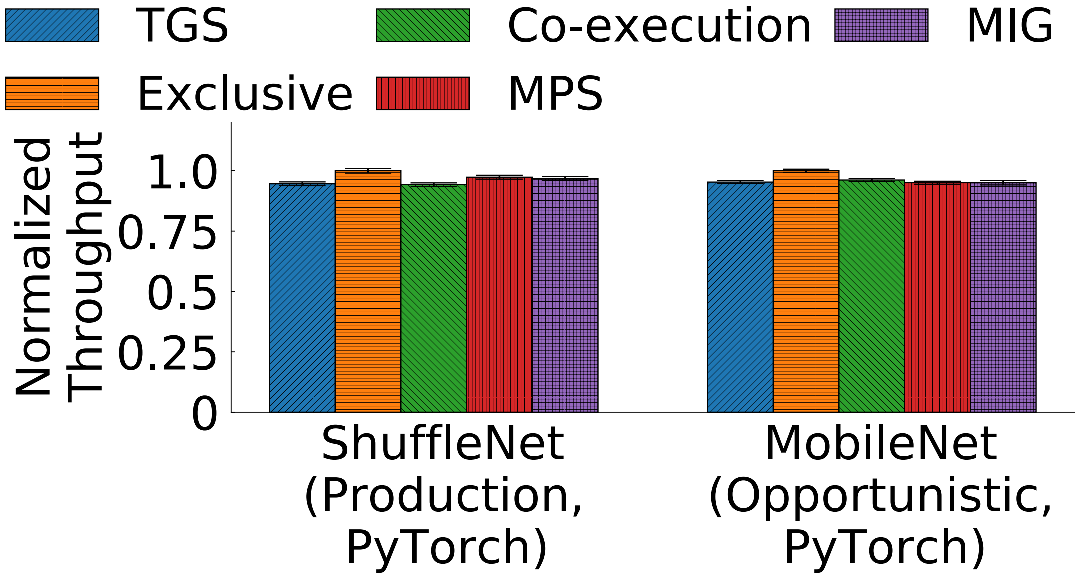
(a). Low-contention scenario



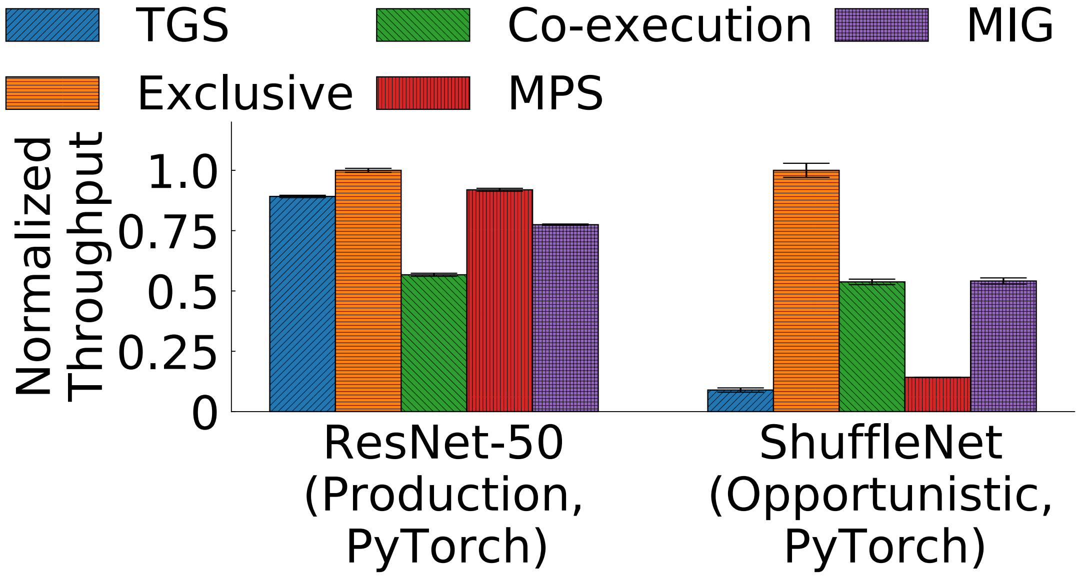
(b).High-contention scenario

Adaptive rate control of TGS

- TGS protects productions job with little overhead, while providing remaining GPU resources to opportunistic jobs



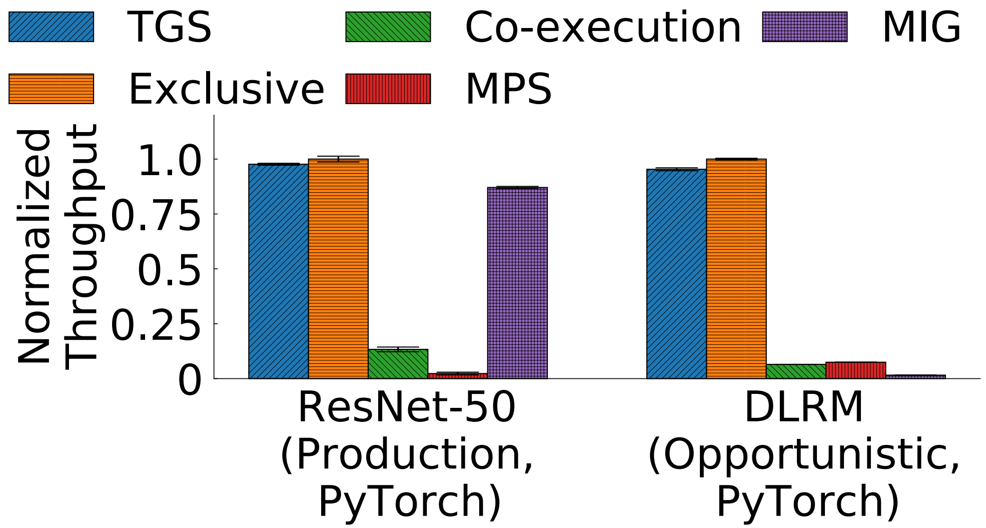
(a). Low-contention scenario



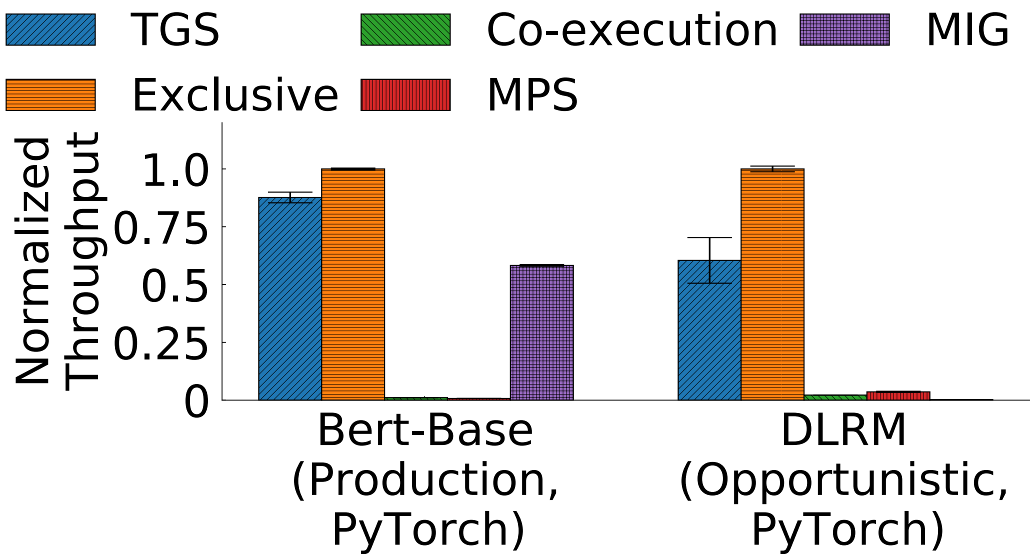
(b).High-contention scenario

Transparent unified memory of TGS

- TGS protects production jobs under GPU memory oversubscription
- 15× throughput improvement compared to MPS



(a). Low-contention scenario



(b).High-contention scenario

More experiments in our paper

- System overhead
- Convergence of TGS in different scenarios
  - Convergence of the rate control under dynamic job arrival
  - Convergence of the rate control under dynamic resource usage
- Supporting different DL frameworks
- GPU sharing for large model training

Conclusion

- TGS provides transparent GPU sharing to DL training in container clouds with four important properties:
  - Transparency
  - Performance isolation
  - High GPU utilization
  - Fault isolation
- TGS improves the throughput of the opportunistic job by up to 15× compared to the existing OS-layer solution MPS

bingyangwu@pku.edu.cn

Thanks!