iSingLab HKUST

Towards Optimal Rack-scale µs-level CPU Scheduling through In-Network Workload Shaping

Xudong Liao, Han Tian, Xinchen Wan, Chaoliang Zeng, Hao Wang, Junxue Zhang, Mengyu Ma, Guyue Liu, Kai Chen

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

University of Science and Technology of China 1958

PEKING UNIVERSITY 1898

inspur

Microsecond-scale workloads in datacenter

Datacenter application

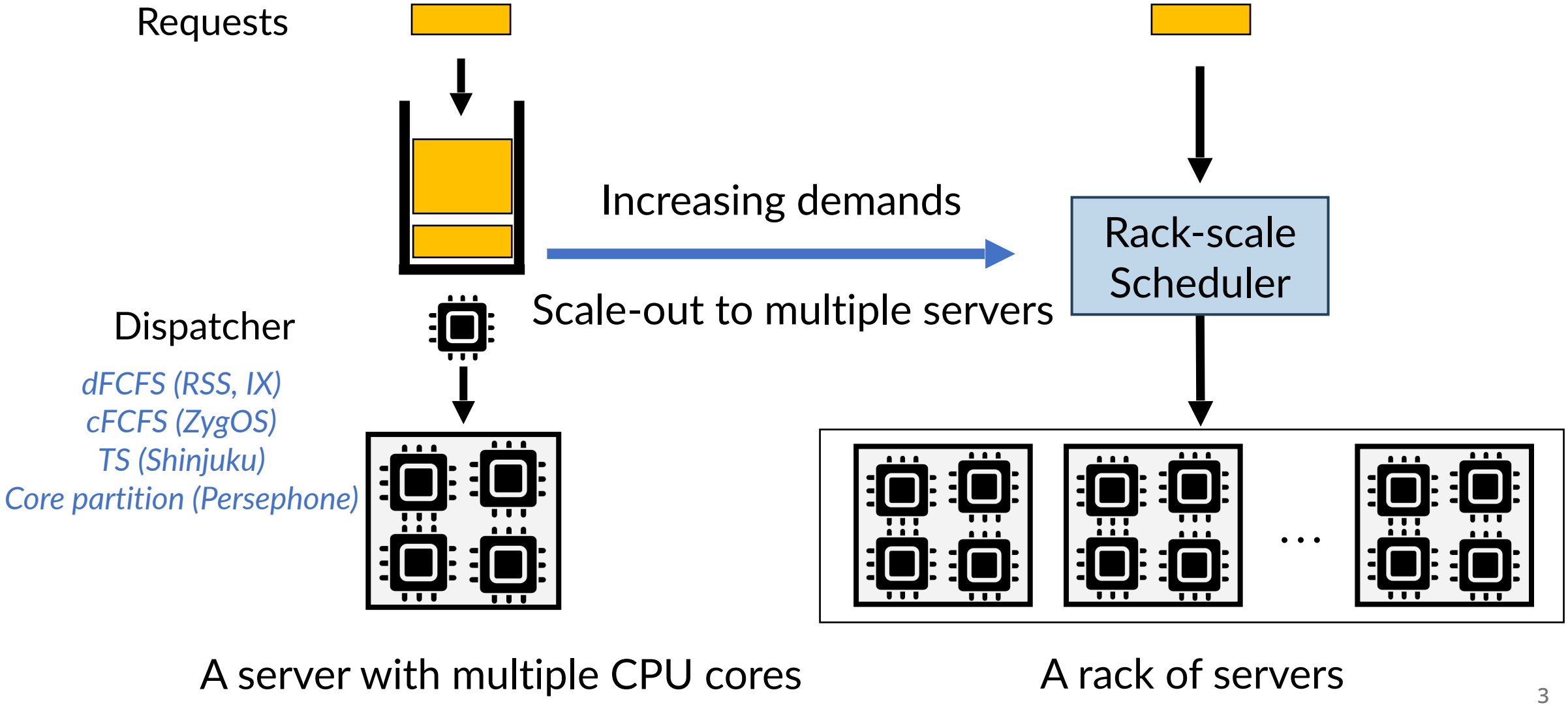
- µs-scale Remote Procedure Calls: 10s ~ 100s of microseconds



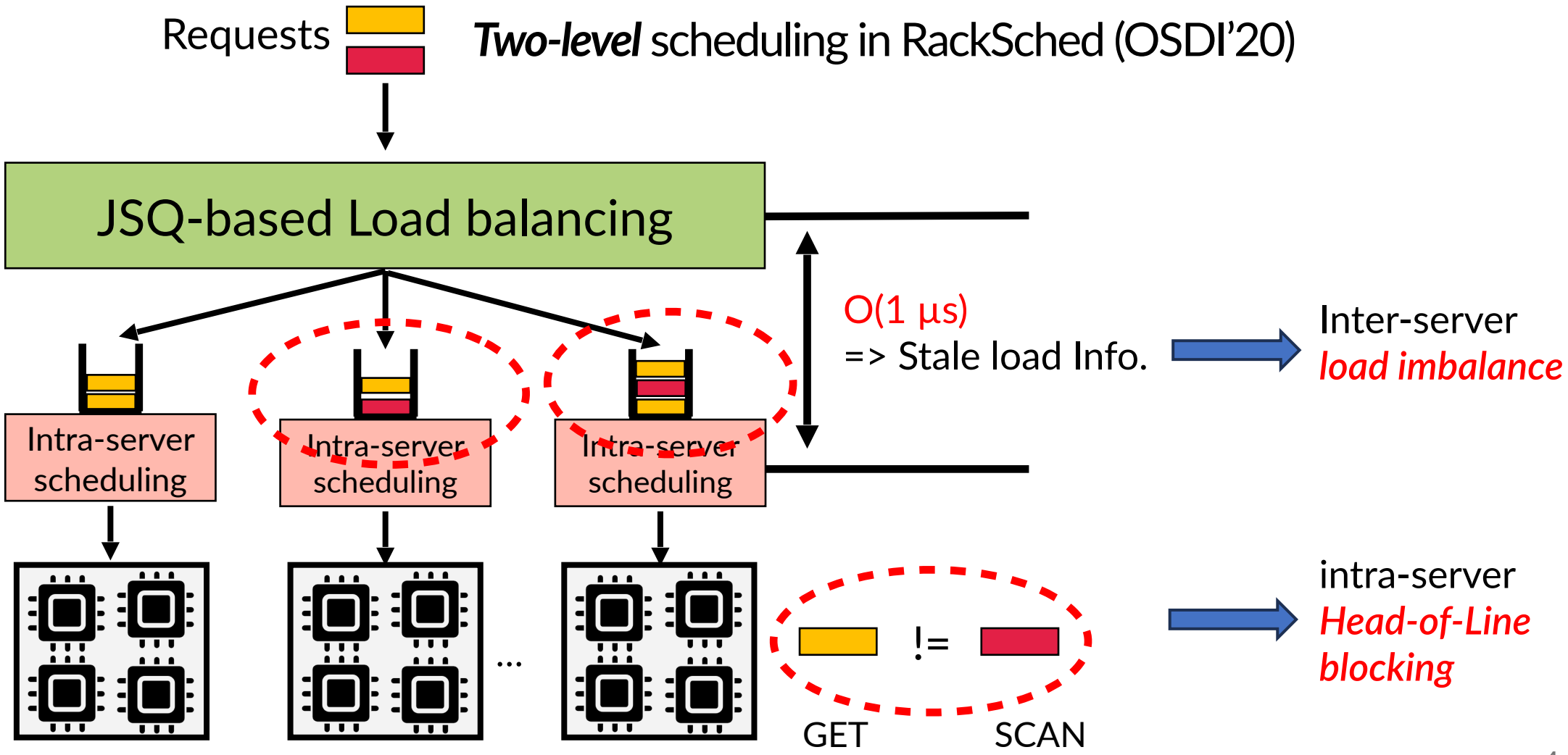
- Tight tail-latency SLOs for online services.

Achieving microsecond-scale tail latency is critical for better user experiences!

How to schedule µs-scale workloads at rack-scale?

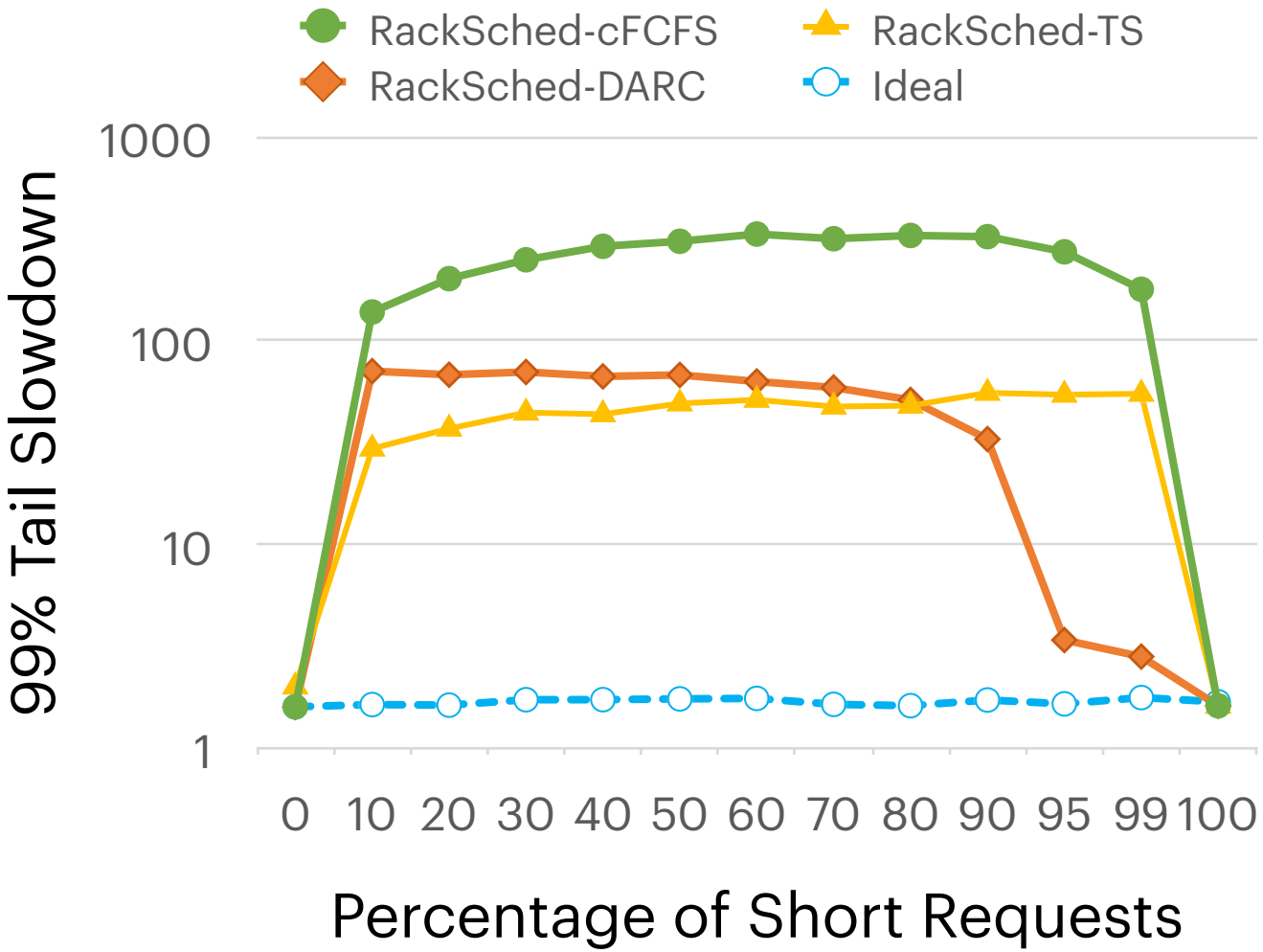


Limitations of current rack-scale scheduling



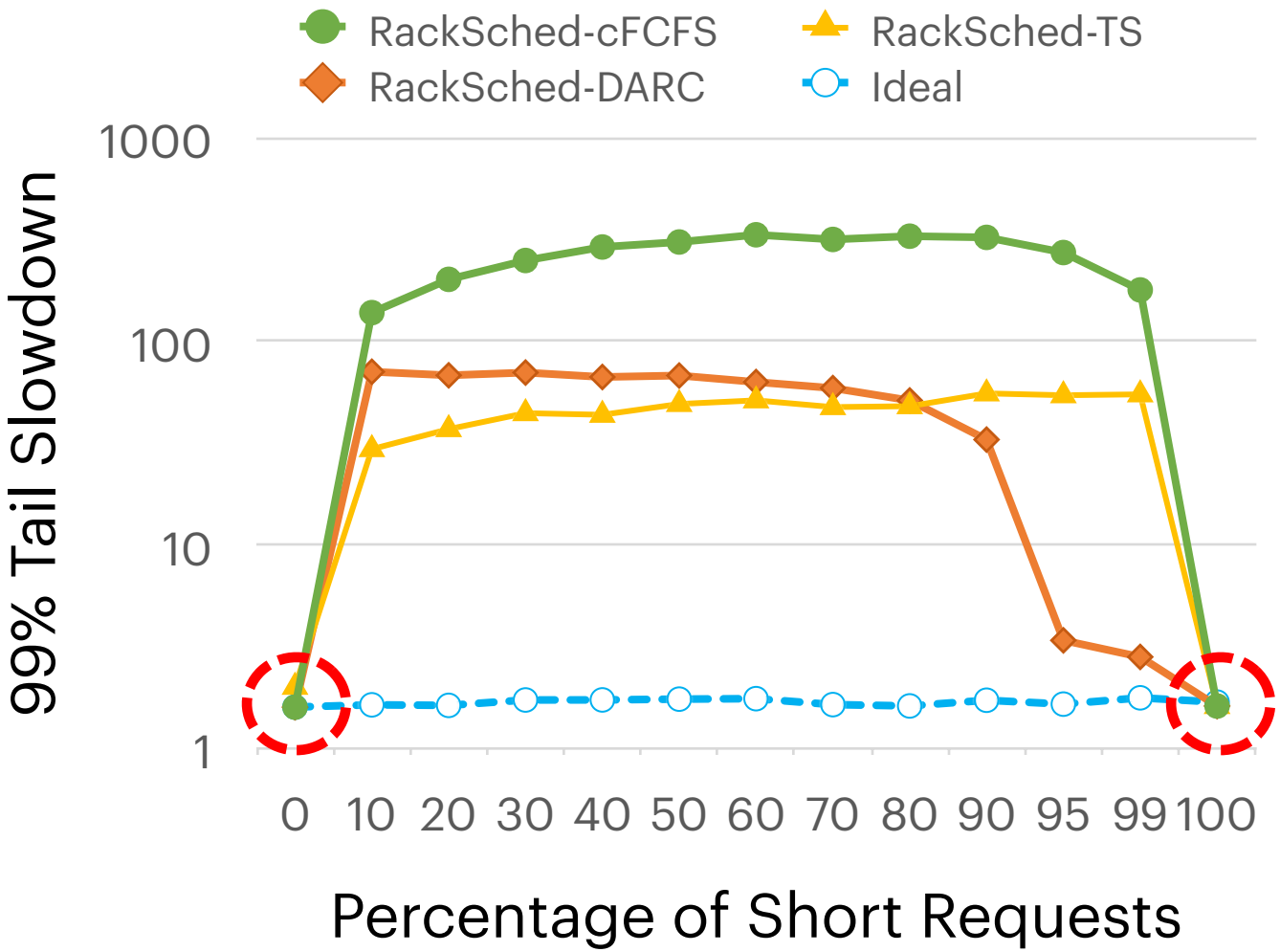
Limitations of current rack-scale scheduling

- Existing work suffers from bad tail latency due to the ***application-agnosticism***.
  - JSQ-based LB leads to inaccurate inter-server load imbalance.
  - Complex intra-server scheduling algorithms cannot mitigates ***HoL*** blocking optimally.



Limitations of current rack-scale scheduling

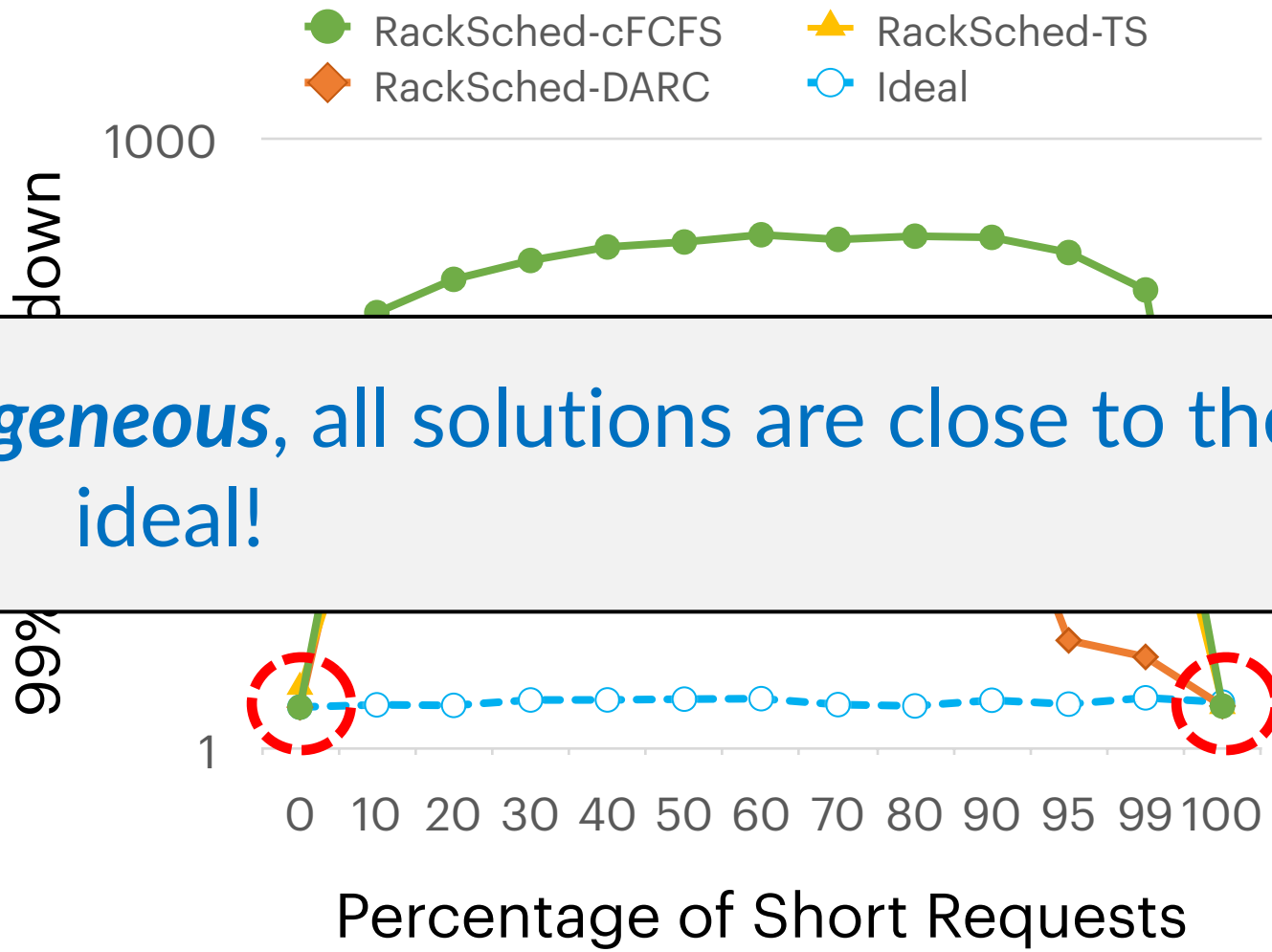
- Existing work suffers from bad tail latency due to the ***application-agnosticism***.
  - JSQ-based LB leads to inaccurate inter-server load imbalance.
  - Complex intra-server scheduling algorithms cannot mitigates ***HoL*** blocking optimally.



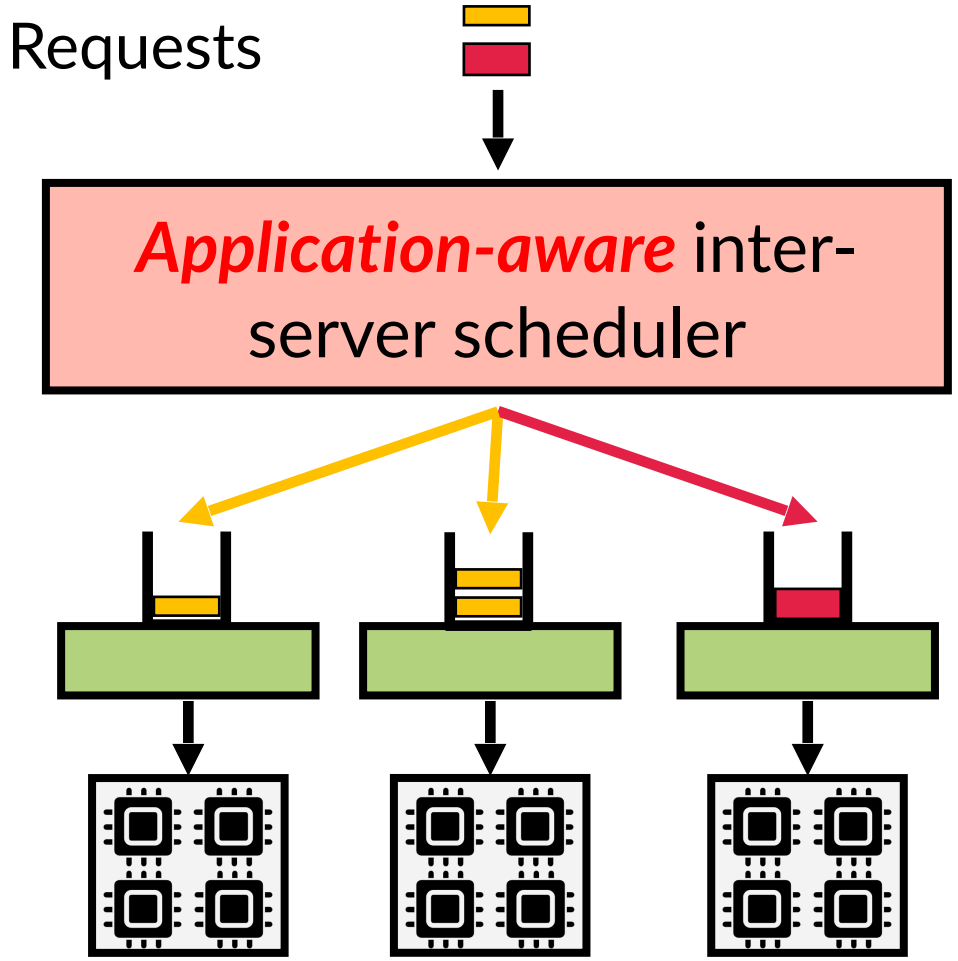
Limitations of current rack-scale scheduling

- Existing work suffers from bad tail latency due to the ***application-agnosticism***.
  - cannot mitigates ***HoL*** blocking optimally.

When workloads are ***homogeneous***, all solutions are close to the ideal!



The case for application-aware scheduling



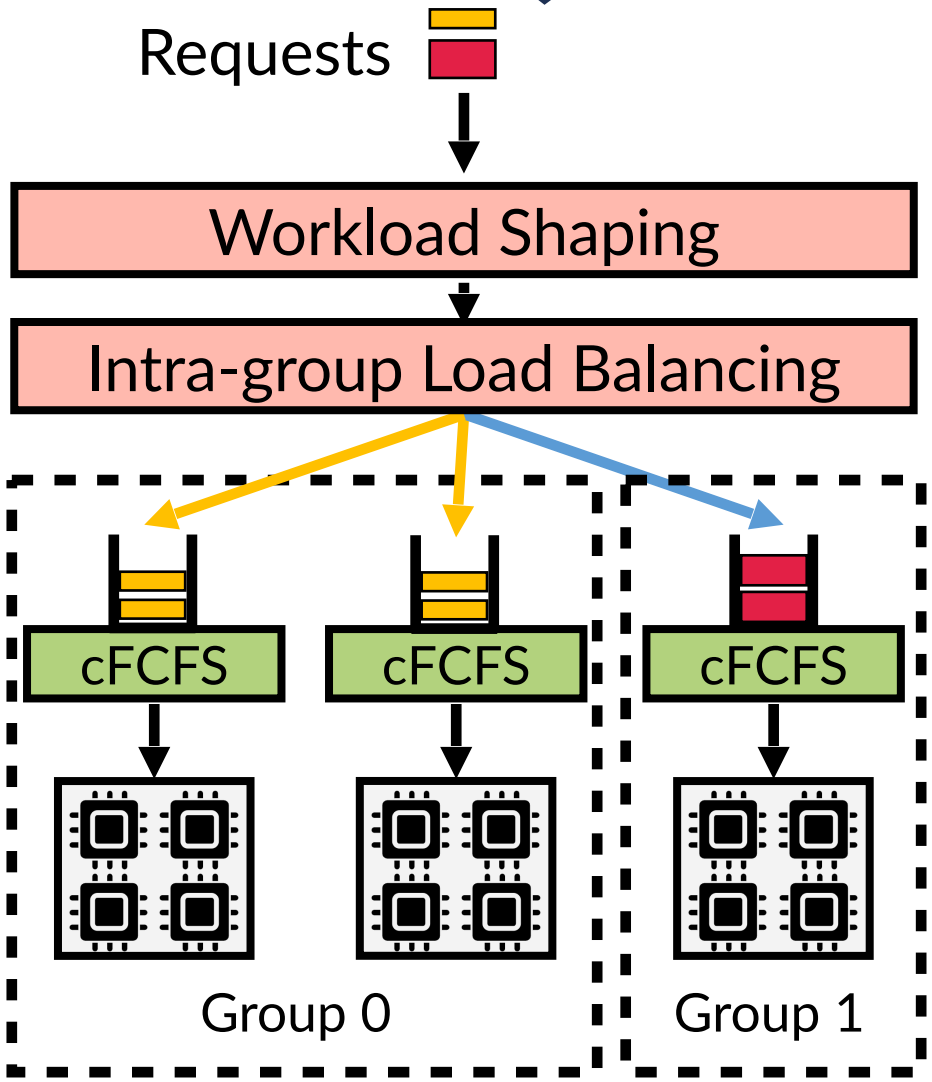
Opportunities:

- Request execution time can be inferred from request headers
  - Monitor and estimate the request load in the *application-aware manner*
- Datacenters are equipped with programmable switches
  - Categorize and schedule requests within the network

Pallas architecture

See the optimality analysis in the paper!

- ***In-network workload shaping***
  - Separate requests into different groups with homogeneity
- ***Accurate*** Intra-group load balancing
  - Req num → cumulative load
- ***Near-optimal*** intra-server scheduling
  - Most servers only face homogeneous workloads, entirely eliminates the HoL blocking.

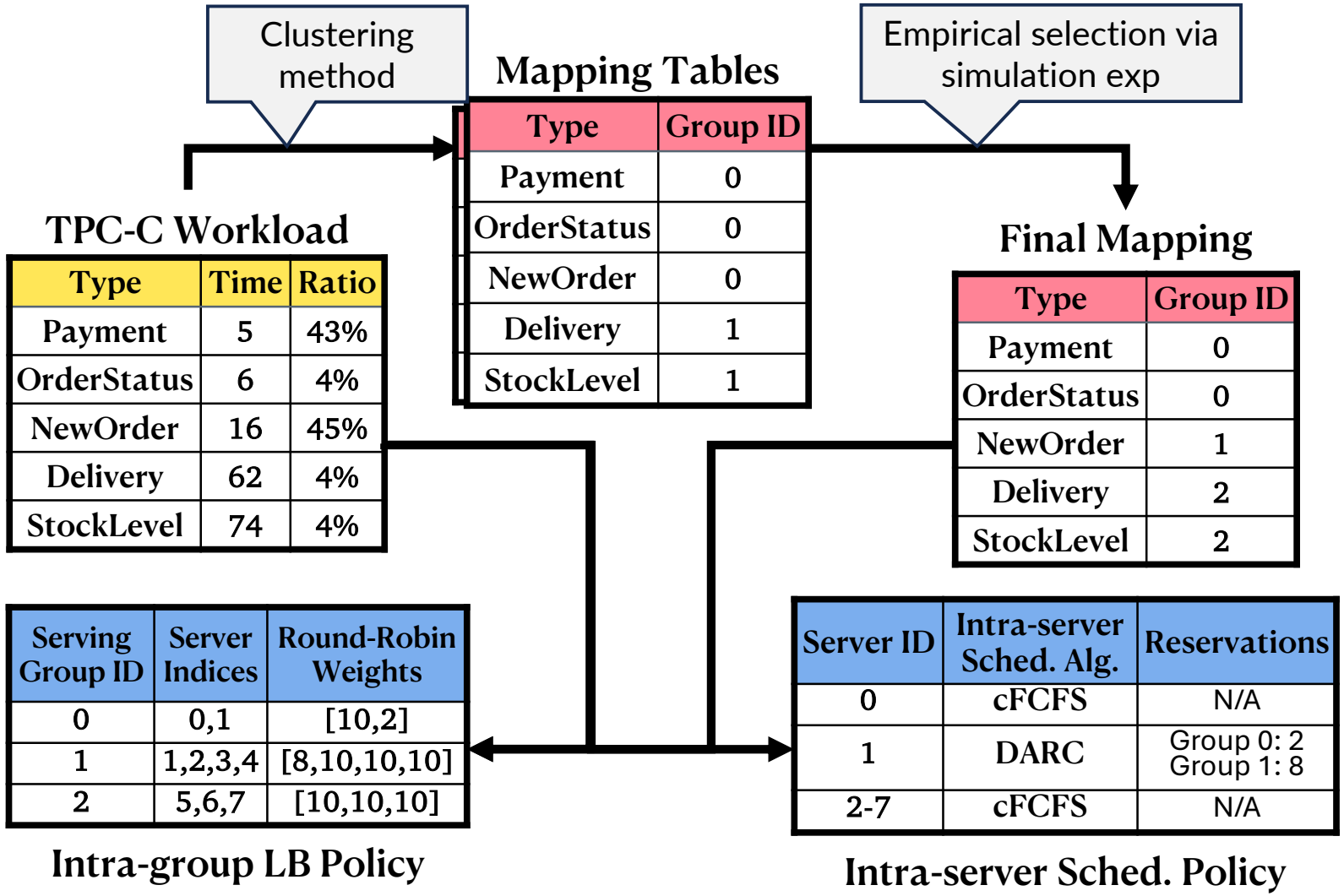


Key challenges in designing Pallas

- How to efficiently shape the workload?
  - More groups leads to more homogeneity but scarifies resource efficiency.
- How to gracefully handle *long-term workload changes*?
  - Request distribution may change over time, simply replacing the scheduling policy is insufficient.
- How to mitigate *short-term workload bursts*?
  - Coarse-grained control granularity is hard to address us-level transient bursts.

Scheduling policy generation

1) Find grouping policies
2) Select candidate
3) Provision resource & generate LB policy & generate Intra-server scheduling policy



Type	Time	Ratio
Payment	5	43%
OrderStatus	6	4%
NewOrder	16	45%
Delivery	62	4%
StockLevel	74	4%

Type	Group ID
Payment	0
OrderStatus	0
NewOrder	0
Delivery	1
StockLevel	1

Type	Group ID
Payment	0
OrderStatus	0
NewOrder	1
Delivery	2
StockLevel	2

Serving Group ID	Server Indices	Round-Robin Weights
0	0,1	[10,2]
1	1,2,3,4	[8,10,10,10]
2	5,6,7	[10,10,10]

Server ID	Intra-server Sched. Alg.	Reservations
0	cFCFS	N/A
1	DARC	Group 0: 2 Group 1: 8
2-7	cFCFS	N/A

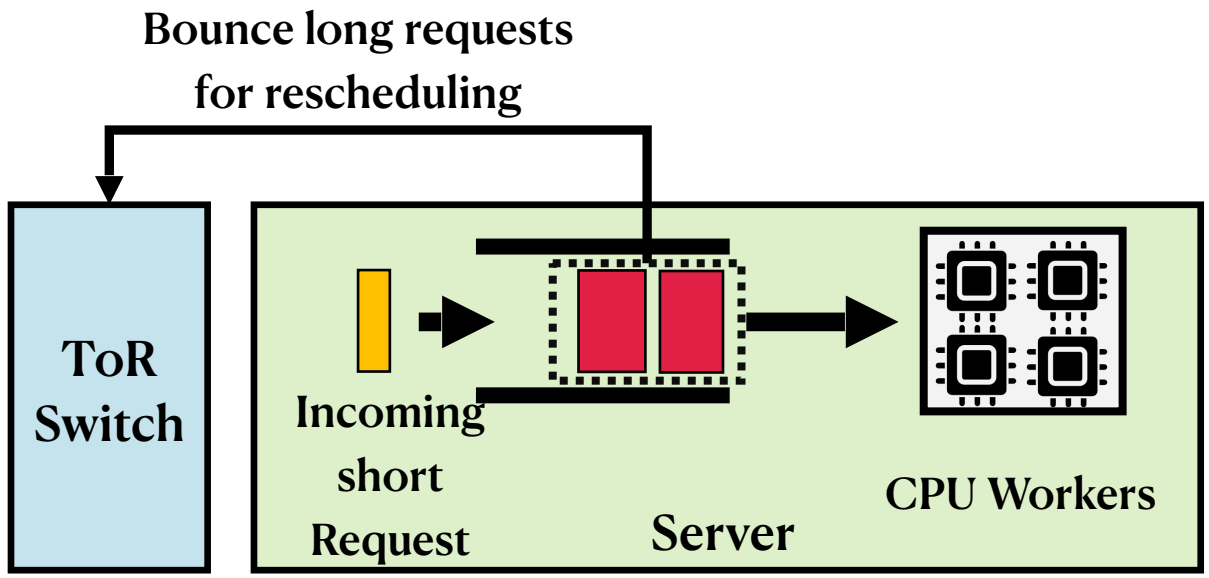
Long-term workload adaptation

➢Workload change detection

- Record request num and exec. time inside switch dataplane registers
- Trigger actions when real time demand deviates from last monitored one by δ

➢Reaction – policy re-generation

- Re-trigger the workload estimation and update the scheduling policy
- *Graceful request bouncing*



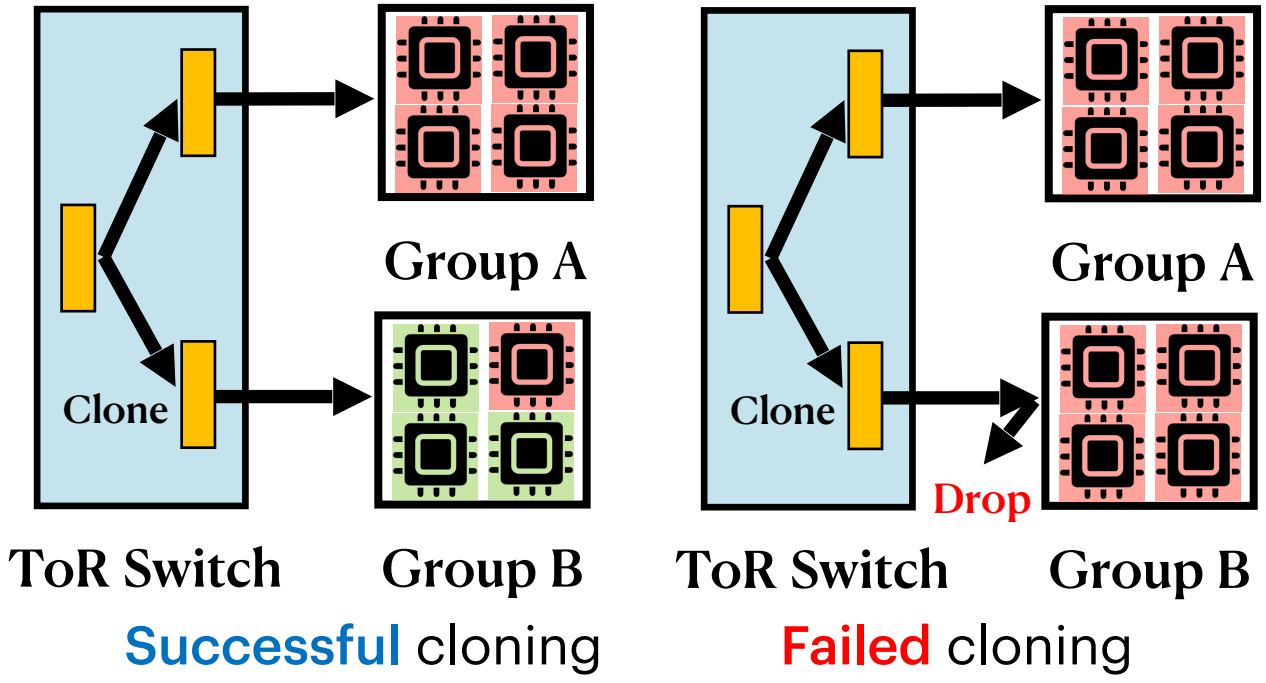
Short-term burst handling

➢Burst detection

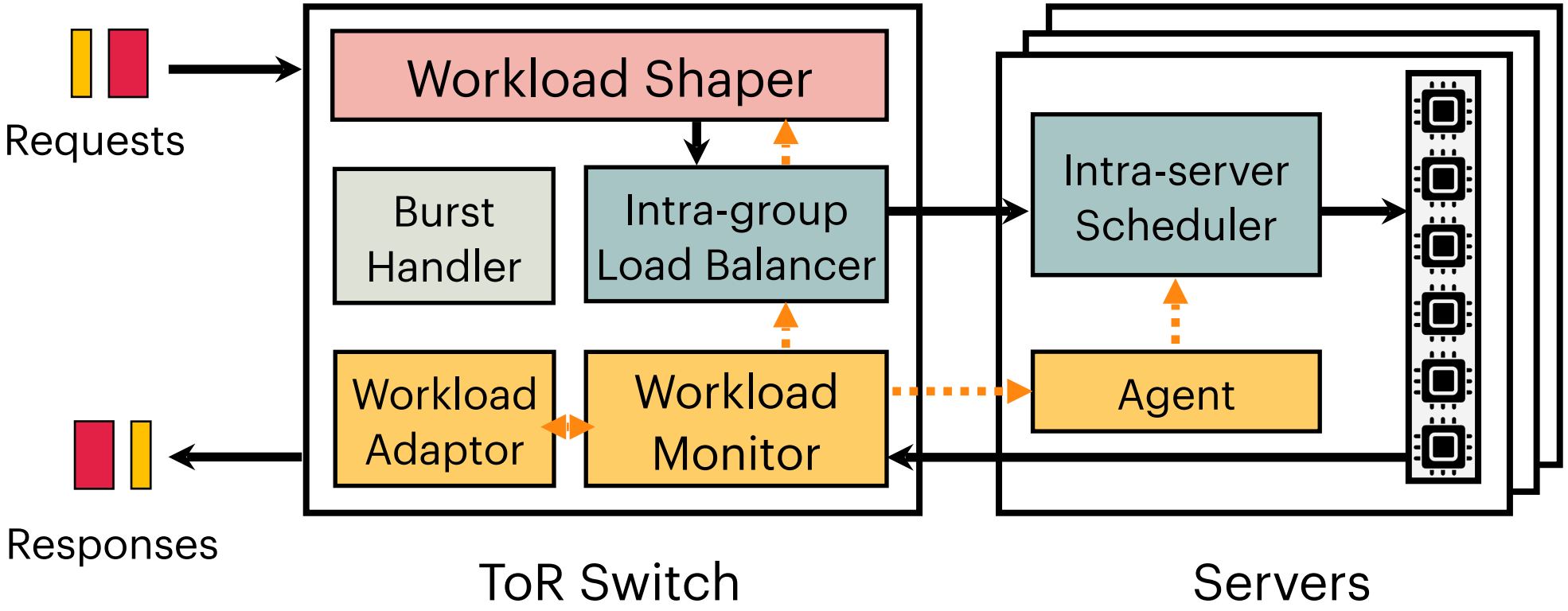
- Real-time observed request num. $\mathbb{C}_{g,t}$ exceeds $\mathbb{C}_{thresh}$ during interval τ
- $\mathbb{C}_{thresh}$ represents upper limit of requests from group g

➢Reaction – Conditional request cloning

- Clone the *bursty portion of requests* to under-utilized groups



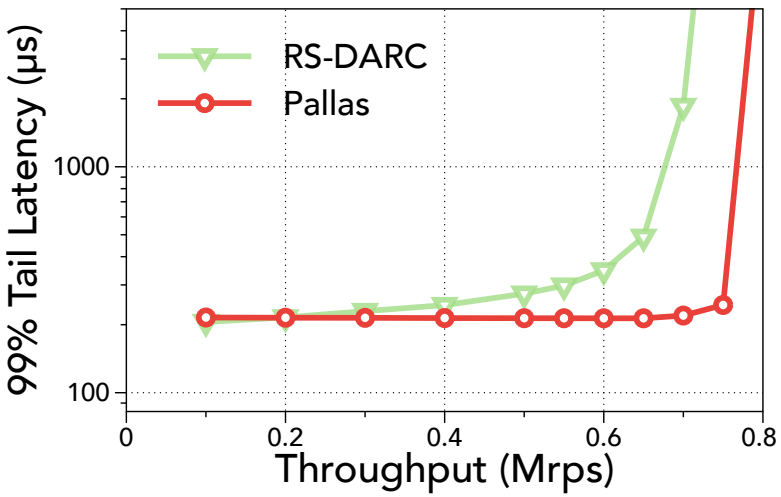
Pallas system overview and workflow



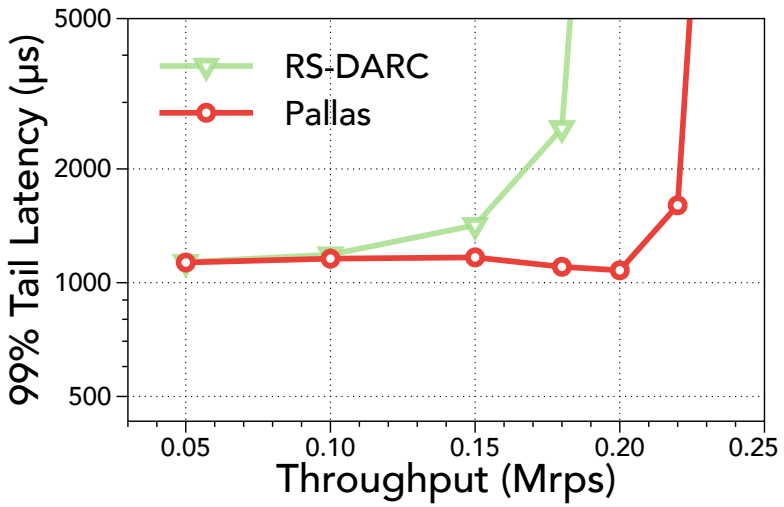
Implementation and evaluation setup

- Implementation
  - Switch Dataplane: workload shaping and WRR intra-group scheduling; request numbering; burst detection
  - Switch control plane: policy enforcement, workload monitoring and change handling
  - Server: simple ***cFCFS*** or ***DARC***
- Testbed
  - Intel Tofino switch
  - 10 servers with Nvidia ConnectX-4 100G NIC and 12 cores
- Workloads
  - Synthetic: exponential, bimodal, trimodal, multi-modal distributions (spin-loop the CPU)
  - Real-world applications: RocksDB with GET and SCAN requests

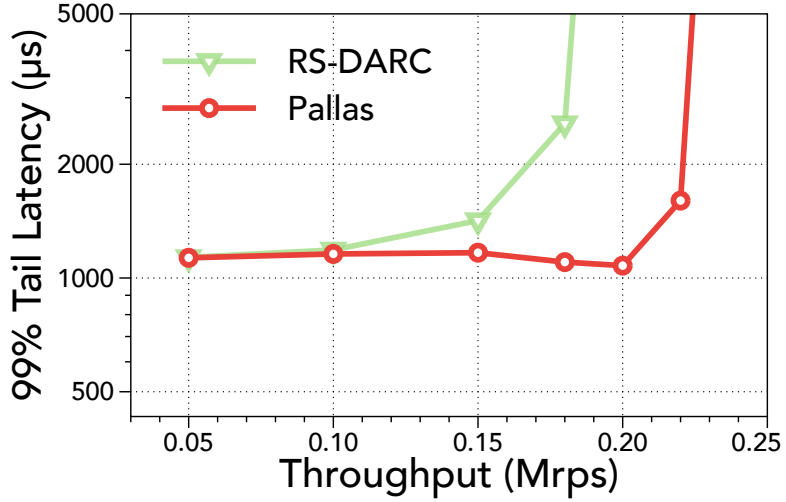
Throughput vs. tail latency in synthetic workloads



Bimodal
(50%-10us, 50%-100us)



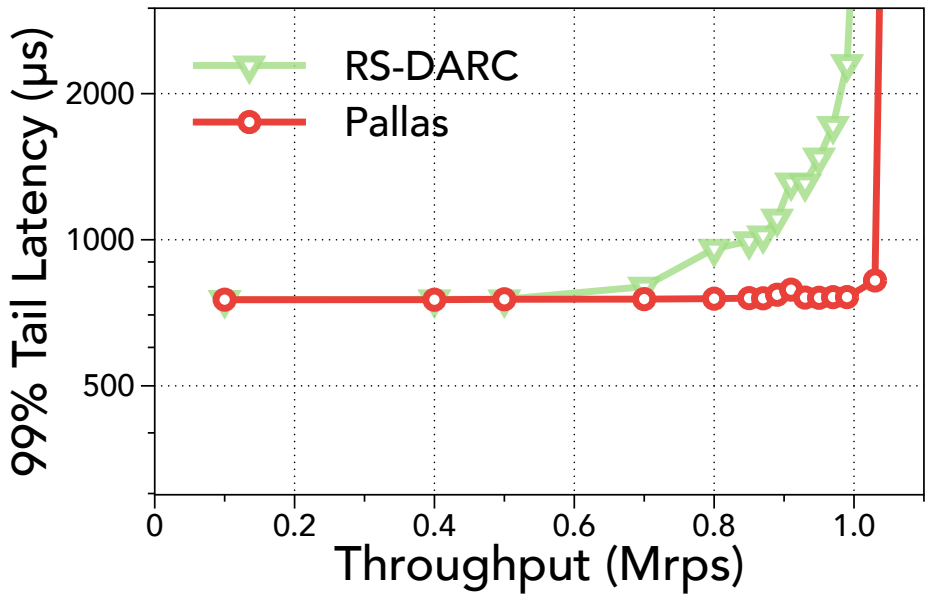
Trimodal
(33%-5us, 33%-50us, 33%-500us)



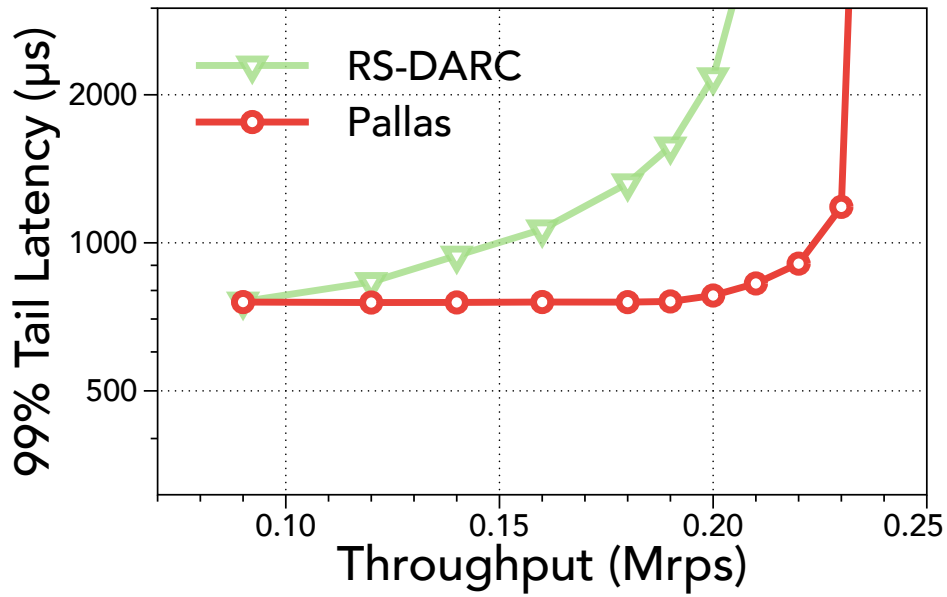
Multi-model from TPC-C

Pallas reduces the tail latency significantly and improves the SLO-target sustainable throughput by up to 2.1x.

Application: RocksDB



Bimodal
(90%-GET, 10%-SCAN)

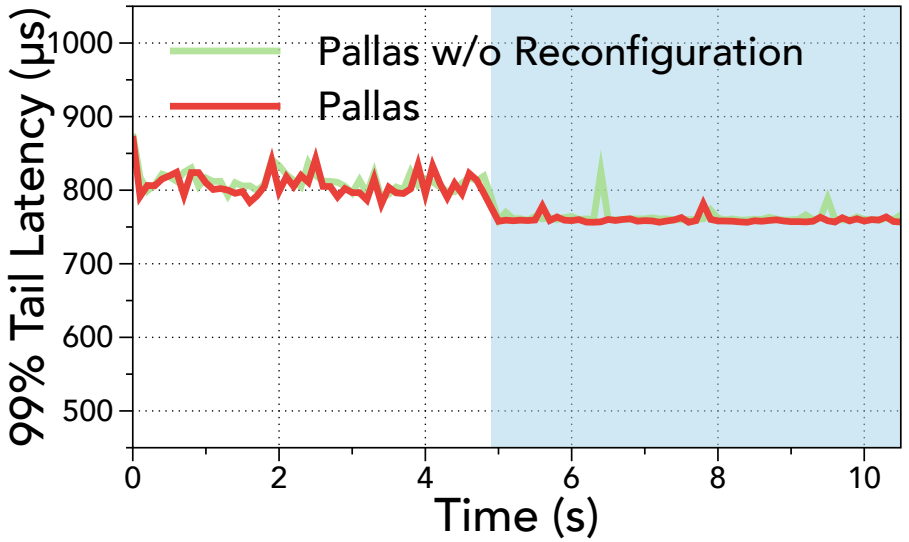


Port Bimodal
(50%-GET, 50%-SCAN)

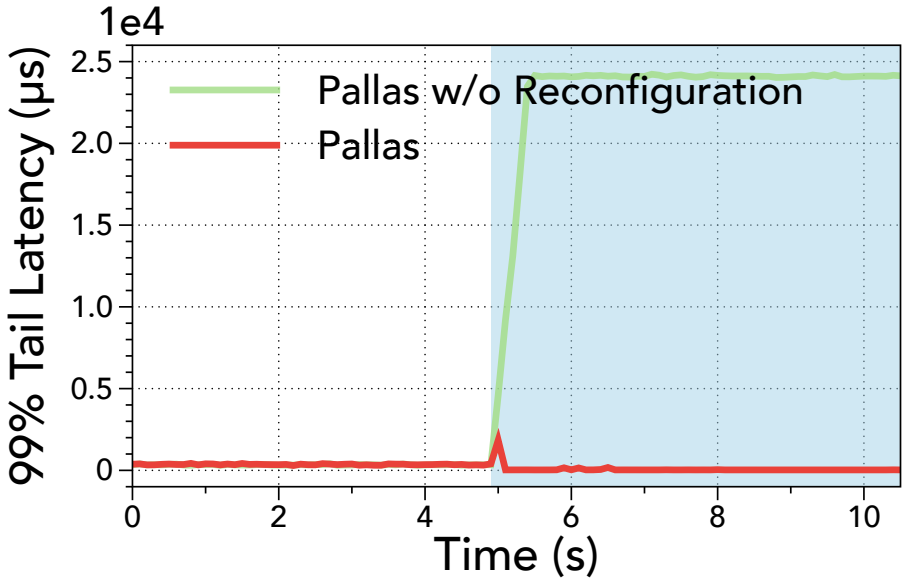
Pallas optimizes the tail latency by 5.5x.

Graceful workload change adaptation

Workload Change from Port Bimodal (50% SCAN) to Bimodal (10% SCAN)



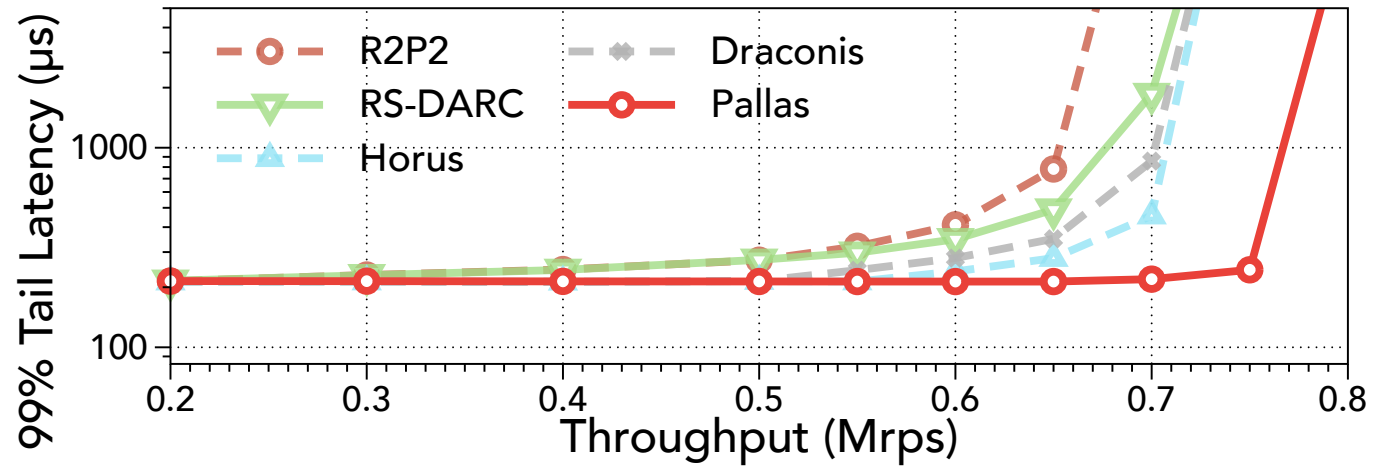
GET Requests



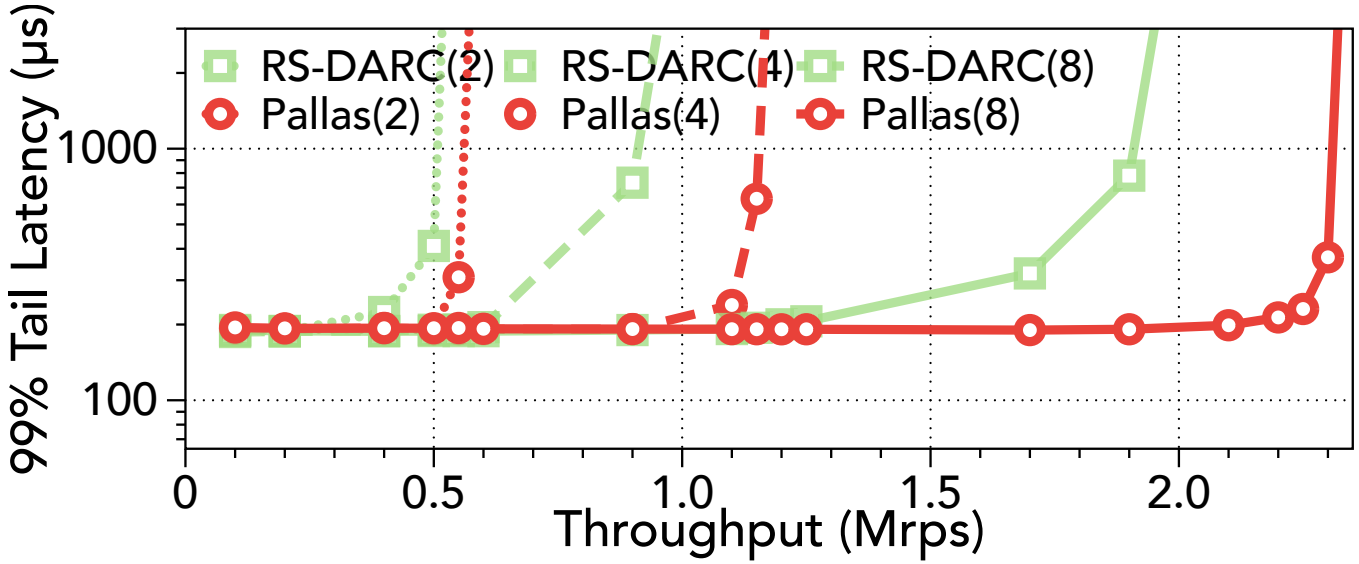
SCAN Requests

Pallas gracefully reacts to workload changes.

Comparison with other methods & Scalability



Pallas is better than other scheduling proposals.



Pallas exhibits almost linear scalability.

See more results in the paper!

Pallas recap

- Pallas is an *application-aware* rack-scale CPU scheduler for microsecond-level workloads.
- The core of Pallas is to proactively transform mixed workloads into uniform ones via in-network workload shaping and then perform ***simple yet near-optimal*** load balancing and intra-server scheduling.
- Pallas is available at https://github.com/HKUST-SING/pallas .

Thank You!

Q&A

Contact email: xudong.liao.cs@gmail.com