RackSched: A Microsecond-Scale Scheduler for Rack-Scale Computers

Hang Zhu

Kostis Kaffes, Zixu Chen, Zhenming Liu, Christos Kozyrakis, Ion Stoica, Xin Jin

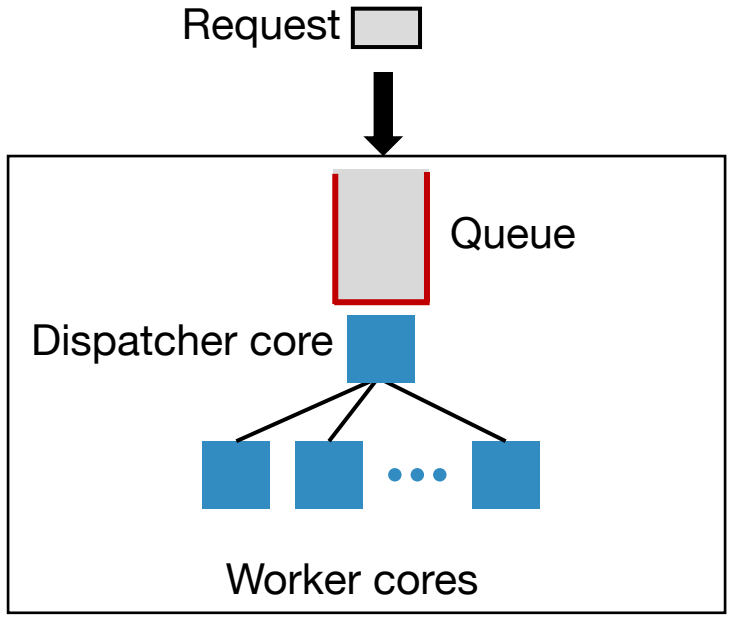
Online services require low tail latency

Low tail latency: 10s~100s of microseconds

How to serve microsecond-scale workloads?

Scale up

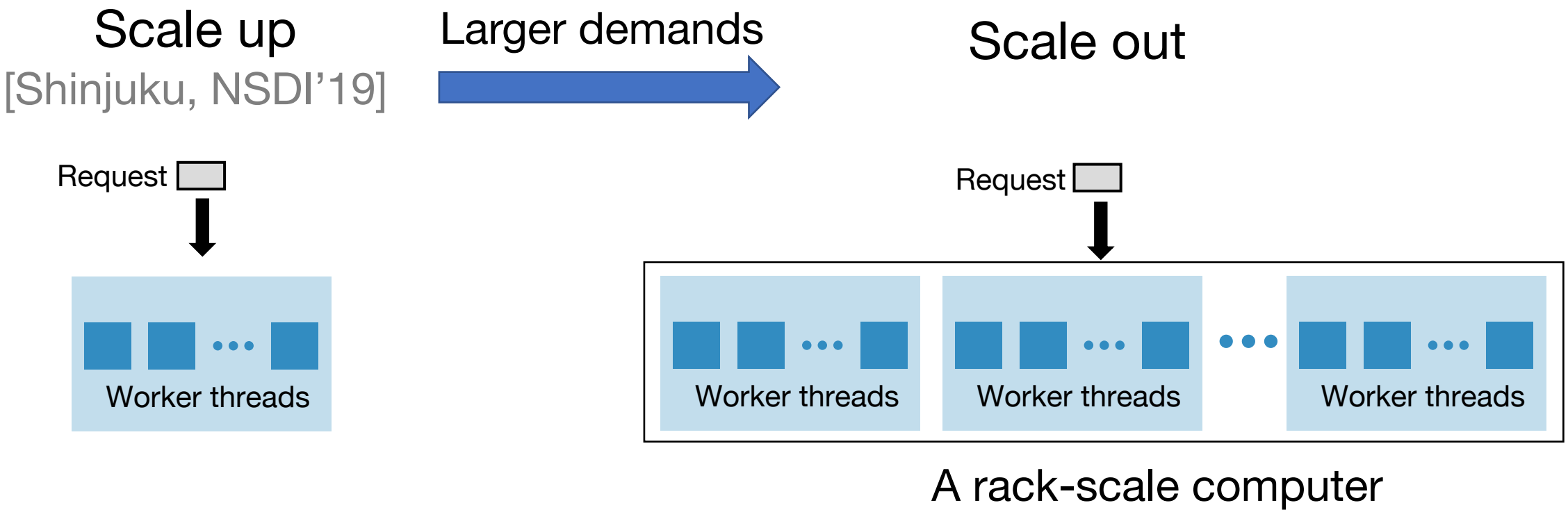
[Shinjuku, NSDI'19]



A multi-core server

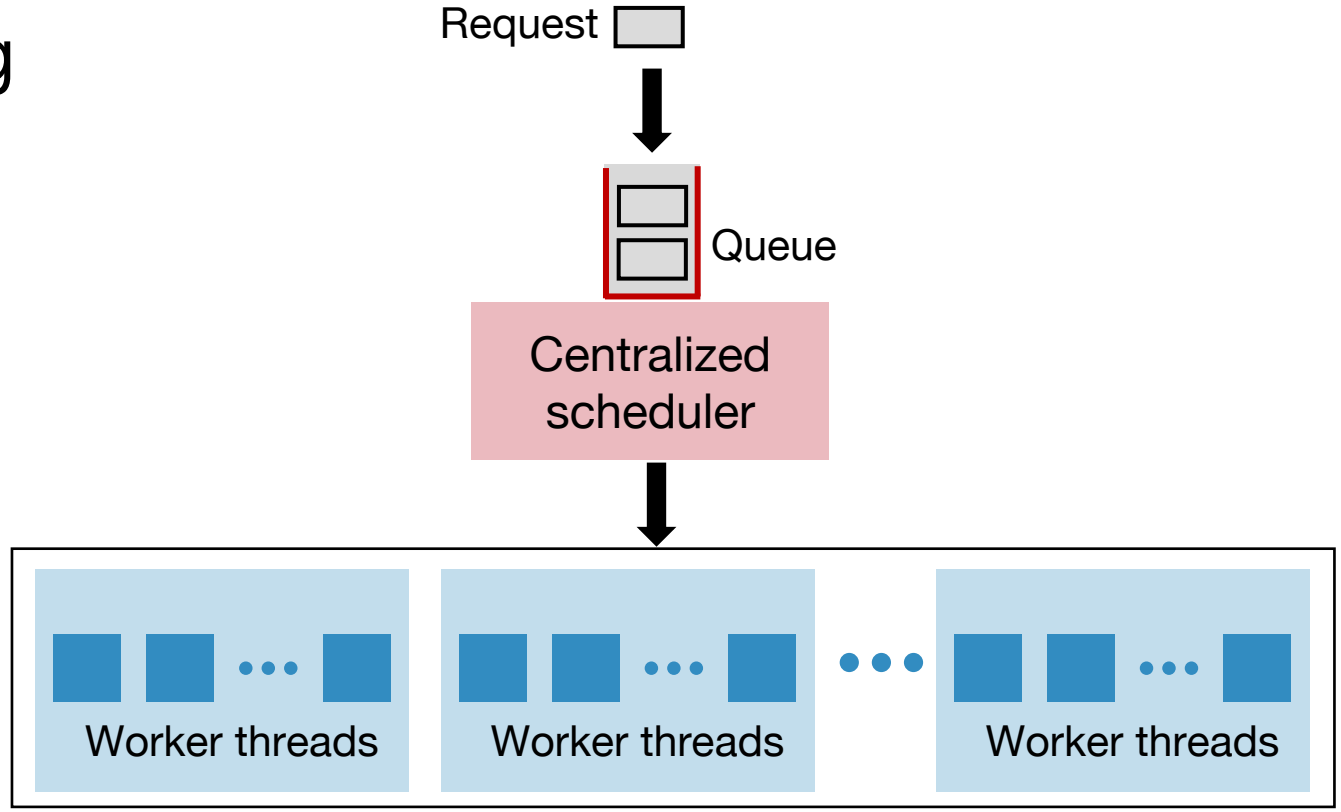
- Centralized scheduling policies
  - cFCFS (centralized first-come-first-serve)
  - PS (processor sharing)

How to serve microsecond-scale workloads?



Can we do centralized scheduling?

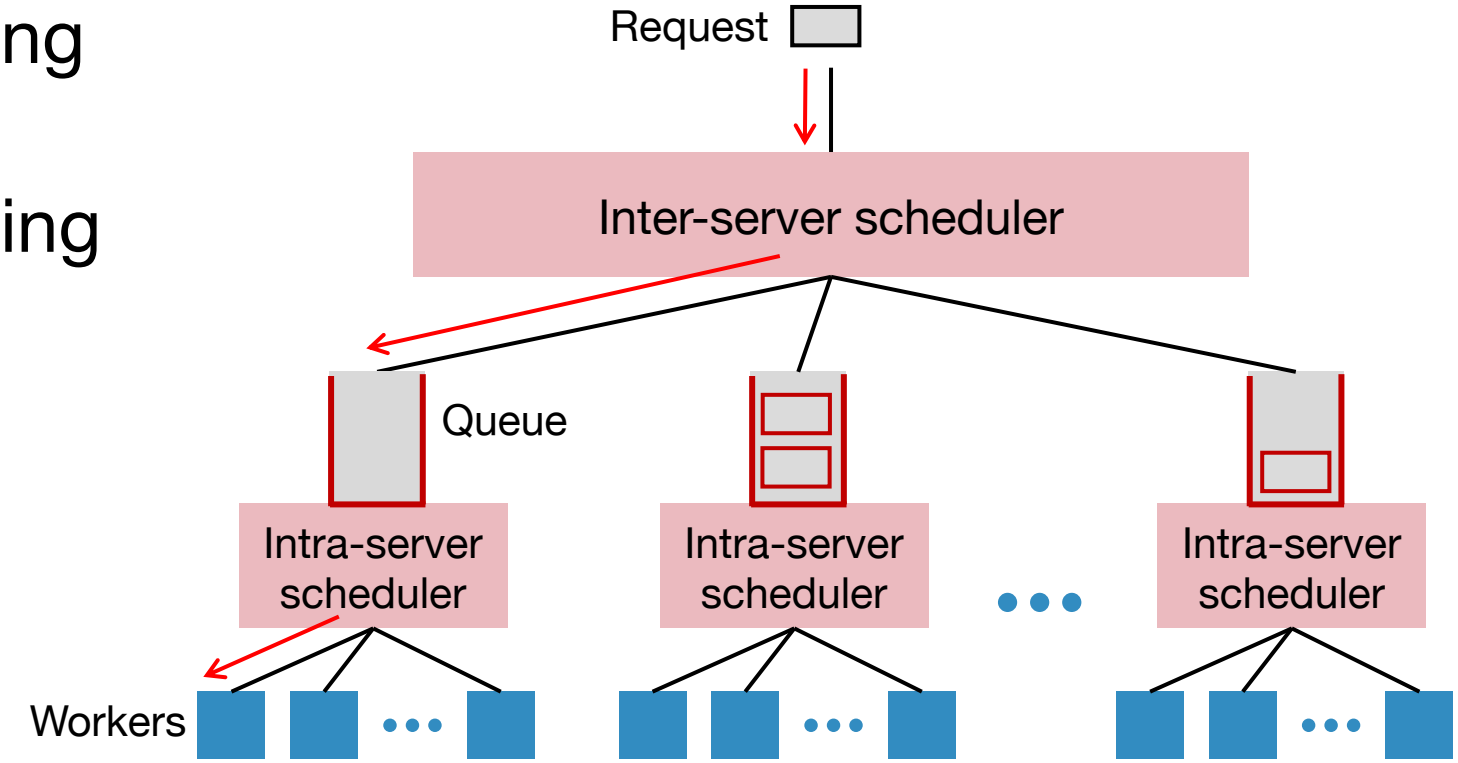
- Centralized scheduling cannot scale



A rack-scale computer

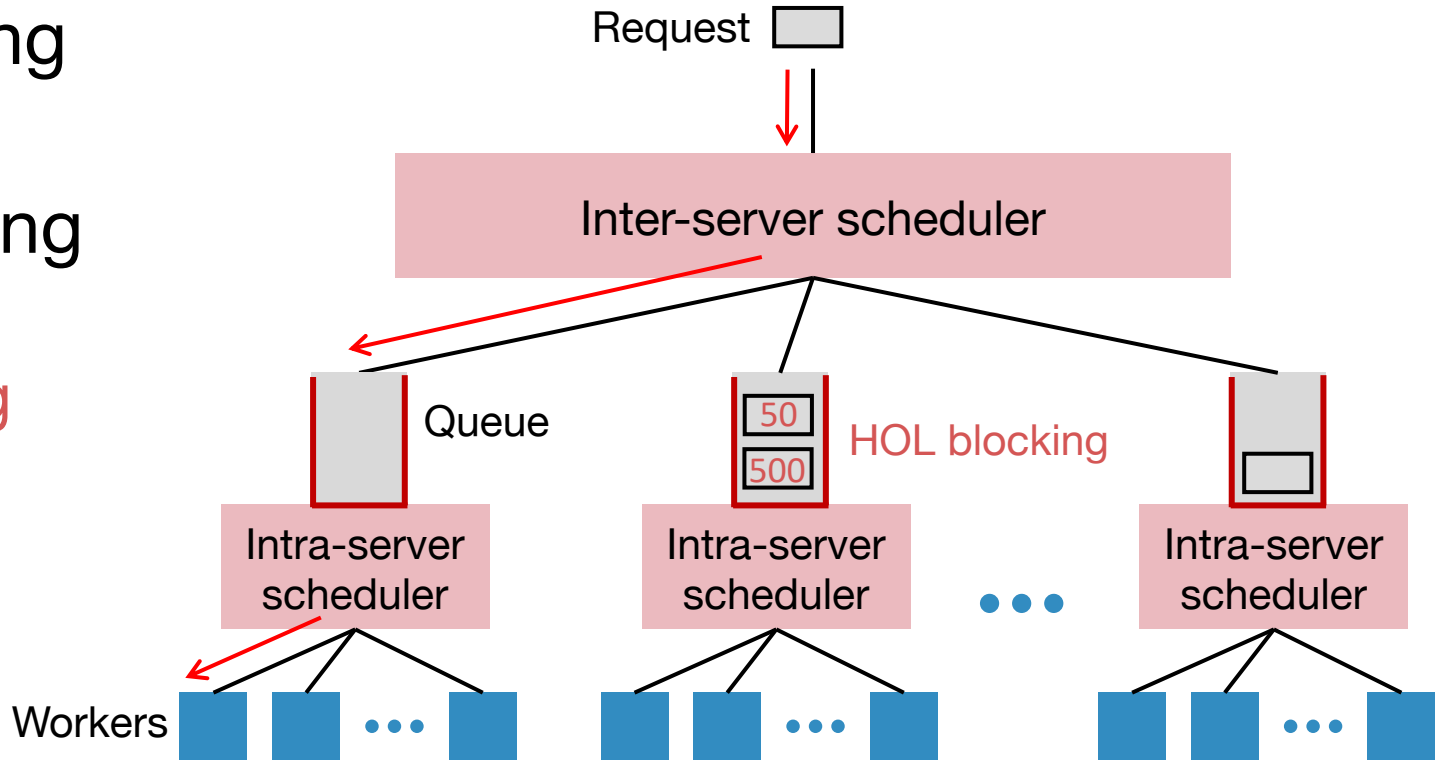
Our proposal: hierarchical scheduling

- Centralized scheduling cannot scale
- Hierarchical scheduling
  - Load imbalance



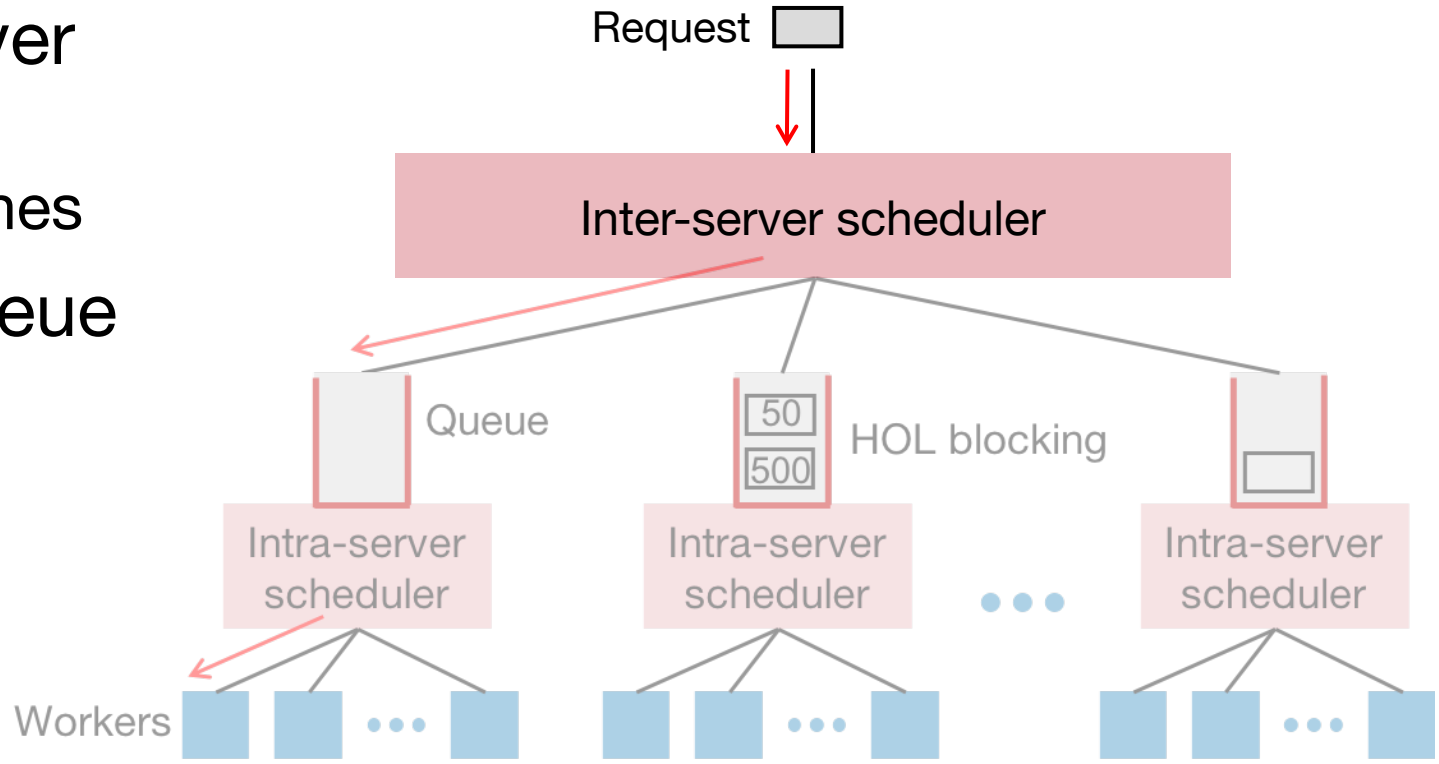
Our proposal: hierarchical scheduling

- Centralized scheduling cannot scale
- Hierarchical scheduling
  - Load imbalance
  - Head-of-line blocking



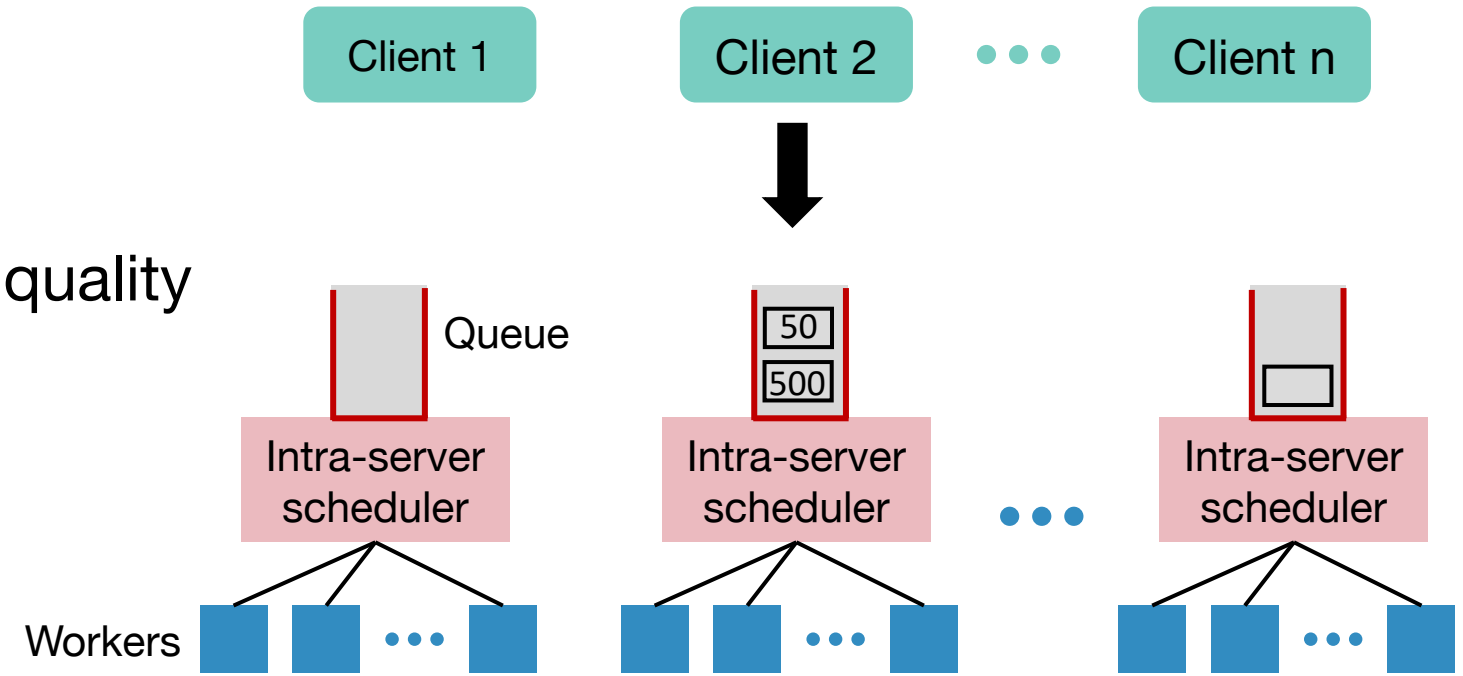
Inter-server scheduler

- Scaling the inter-server scheduler
  - Programmable switches
- Join-the-shortest-queue (JSQ) is bufferless
- Approximating JSQ
  - Power-of-k-choices



A distributed, client-based inter-server scheduler?

- Client complexity
- Overhead for reconfiguration
- Worse scheduling quality



How to realize the two-layer scheduling framework?

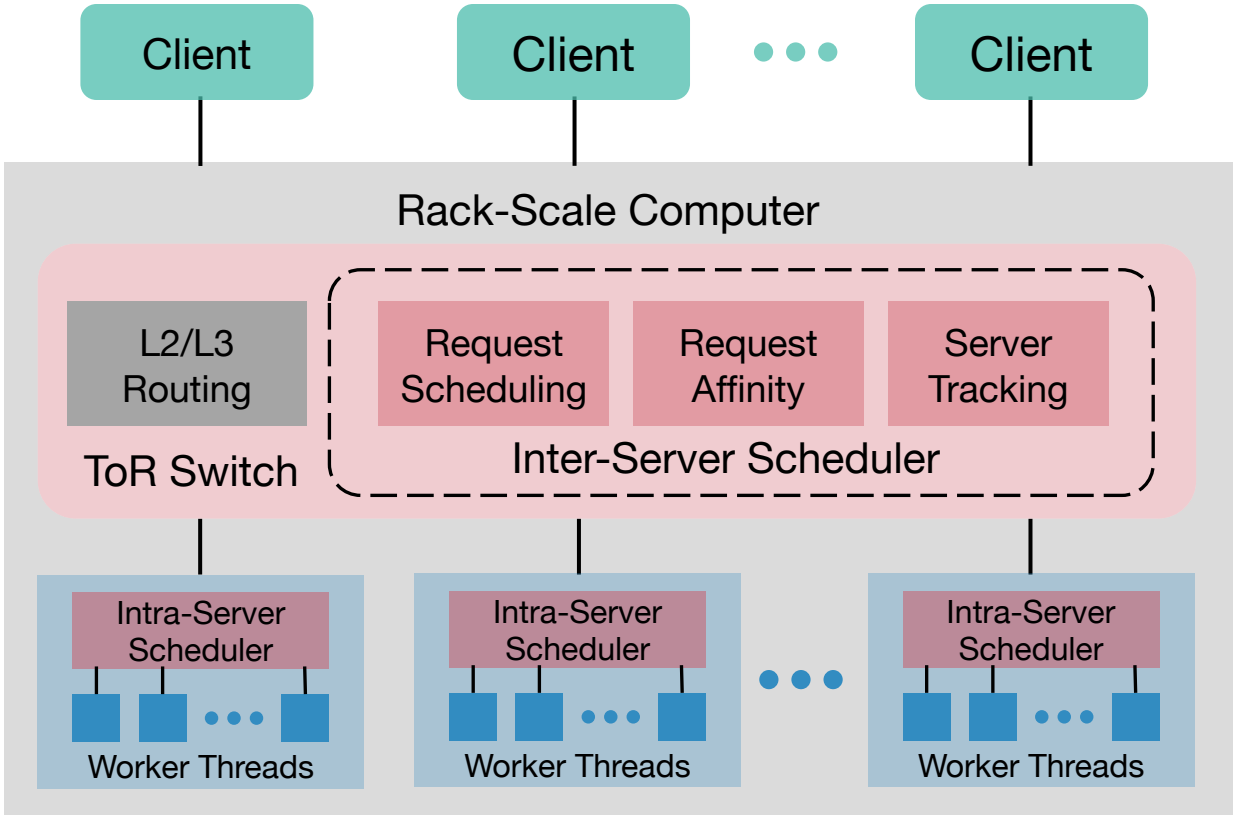
- What is the system architecture?
- How to process/schedule requests based on the server loads?
- How to ensure request affinity?
- How to handle practical scheduling requirements?

How to realize the two-layer scheduling framework?

- What is the system architecture?
- How to process/schedule requests based on the server loads?
- How to ensure request affinity?
- How to handle practical scheduling requirements?

RackSched architecture

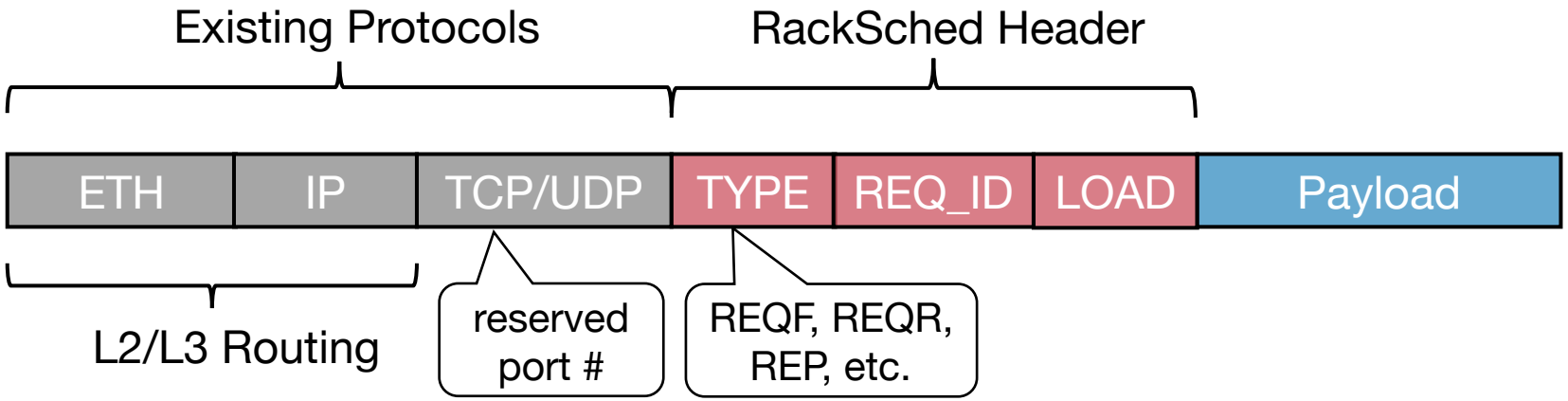
- Inter-server scheduling
  - Handle temporal load imbalance
- Intra-server scheduling
  - Handle head-of-line blocking



How to realize the two-layer scheduling framework?

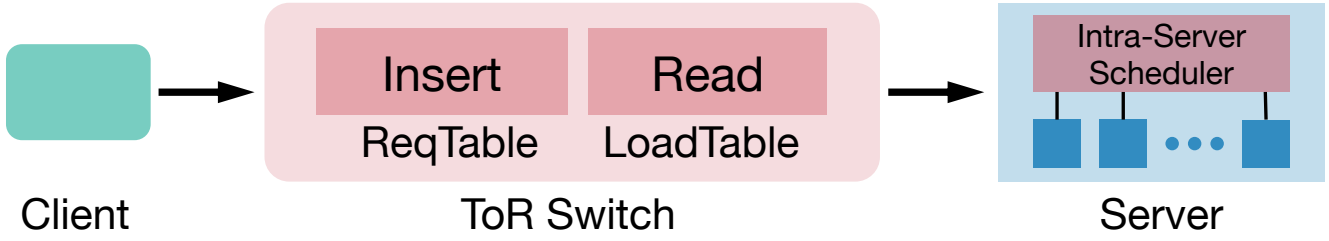
- What is the system architecture?
- How to process/schedule requests based on the server loads?
- How to ensure request affinity?
- How to handle practical scheduling requirements?

Packet format

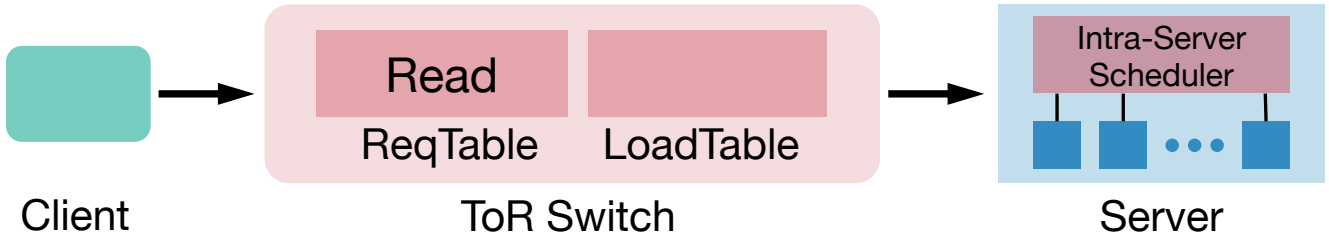


Request processing

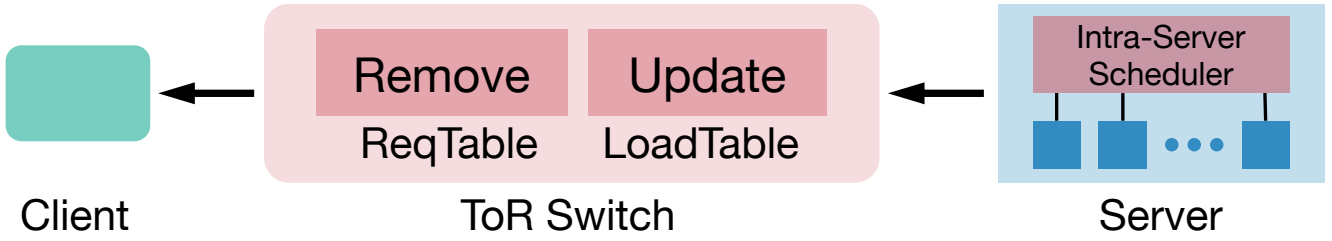
- The 1st packet of a request



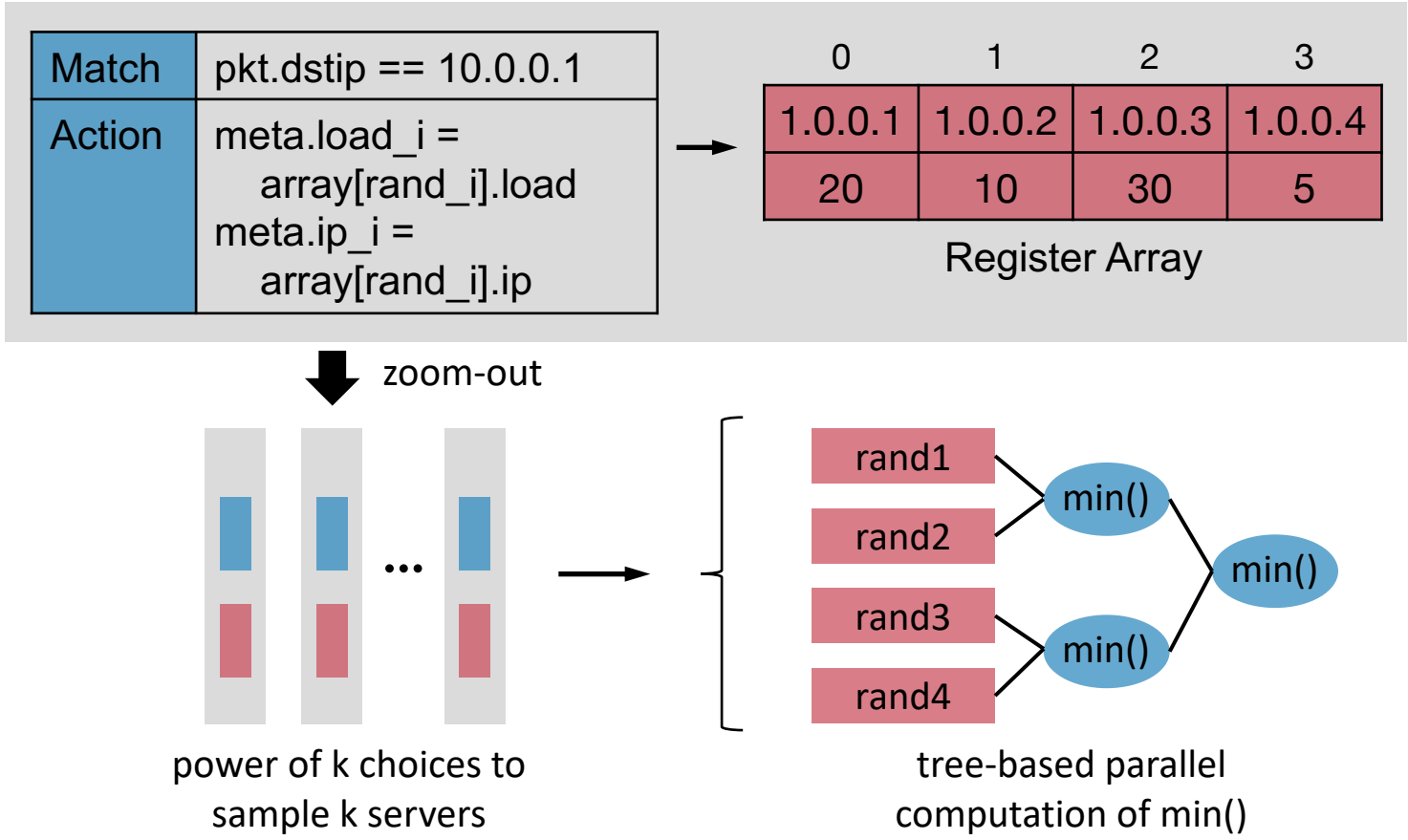
- The following packets of the request



- The reply packets



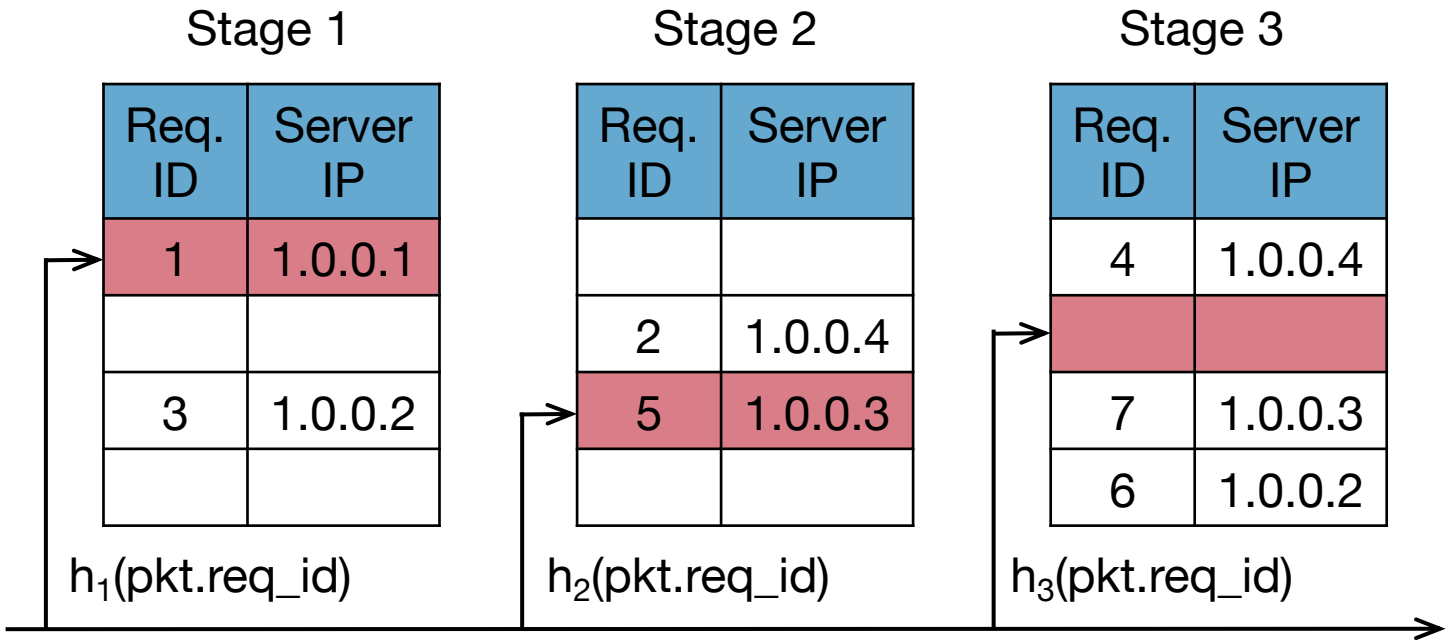
Request scheduling



How to realize the two-layer scheduling framework?

- What is the system architecture?
- How to process/schedule requests based on the server loads?
- How to ensure request affinity?
- How to handle practical scheduling requirements?

Request affinity



- **Insert: iterate over stages to find an empty slot**
- **Read: find a matched slot to read the server IP**
- **Remove: remove a completed request**

How to realize the two-layer scheduling framework?

- What is the system architecture?
- How to process/schedule requests based on the server loads?
- How to ensure request affinity?
- How to handle practical scheduling requirements?

Handling scheduling requirements

- Multi-queue support
  - Separate queue for each request type on each server
- Locality and placement constraints
  - Data locality
  - Request dependency
- Resource allocation policies
  - Strict priority
  - Weighted fair sharing

Check our paper for more details

Implementation

- Switch
  - 6.5Tbps Barefoot Tofino switch
  - Written in P4
- Server
  - 8-core CPU and 40G NIC
  - Shinjuku
- Client
  - Intel DPDK 16.11.1

https://github.com/netx-repo/RackSched

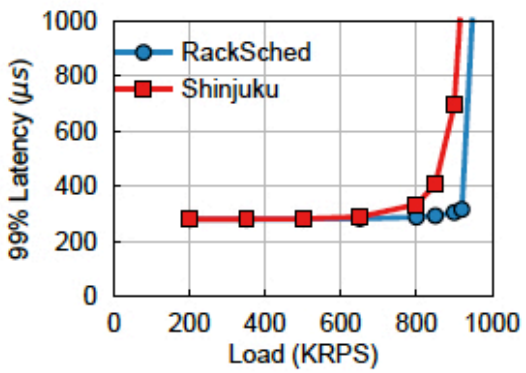
Evaluation

- Does RackSched improve the performance?
- Does RackSched scale?
- Does RackSched benefit applications?
- What is the impact of the design decisions?
- Does RackSched ensure request affinity?

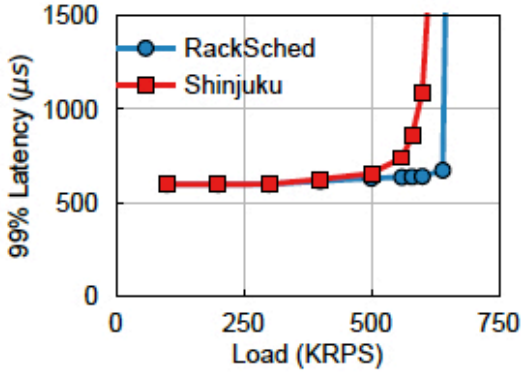
Evaluation

- Does RackSched improve the performance?
- Does RackSched scale?
- Does RackSched benefit applications?
- What is the impact of the design decisions?
- Does RackSched ensure request affinity?

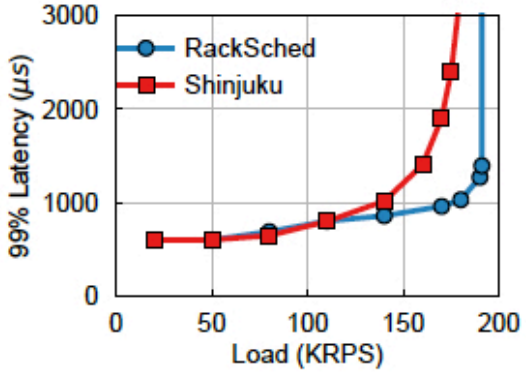
Does RackSched improve the performance?



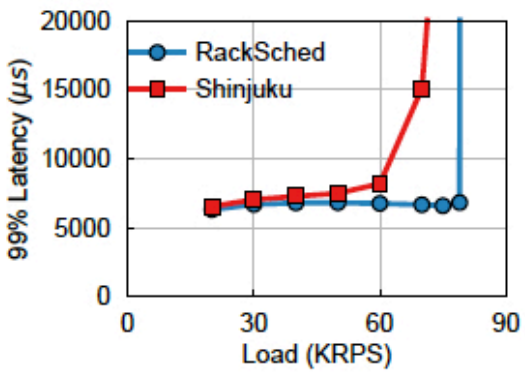
(a) Exp(50).



(b) Bimodal(90%-50, 10%-500).



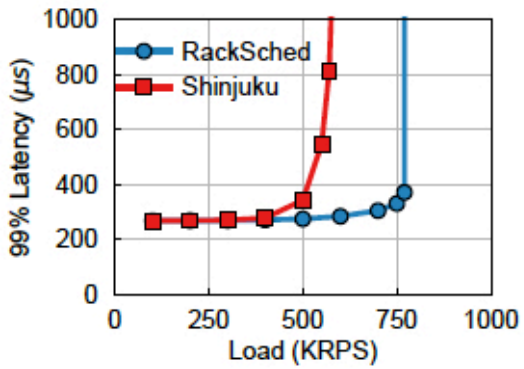
(c) Bimodal(50%-50, 50%-500).



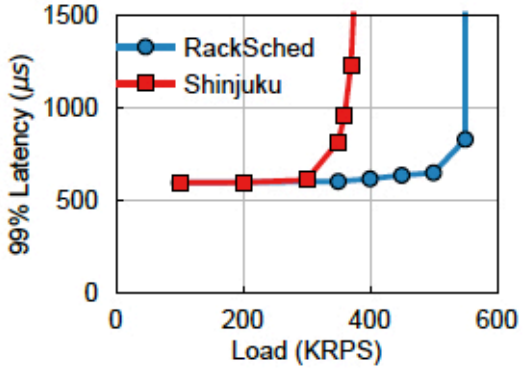
(d) Trimodal(33.3%-50, 33.3%-500, 33.3%-5000).

RackSched supports larger throughput with lower tail latency

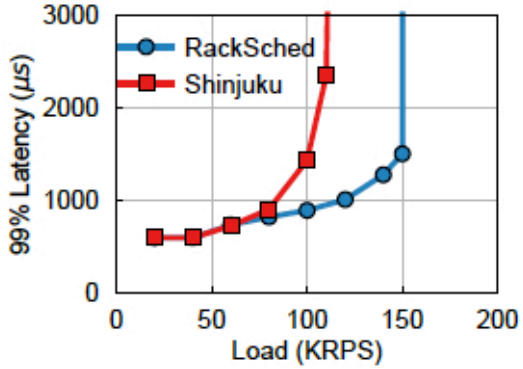
Does RackSched improve the performance?



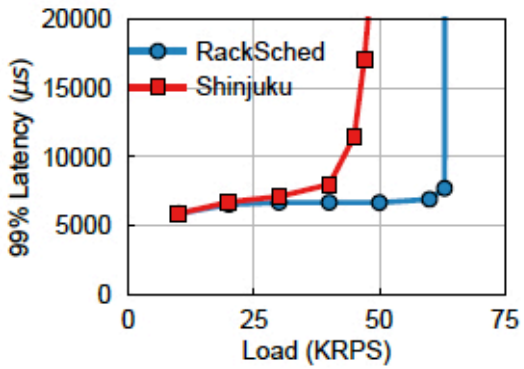
(a) Exp(50).



(b) Bimodal(90%-50, 10%-500).



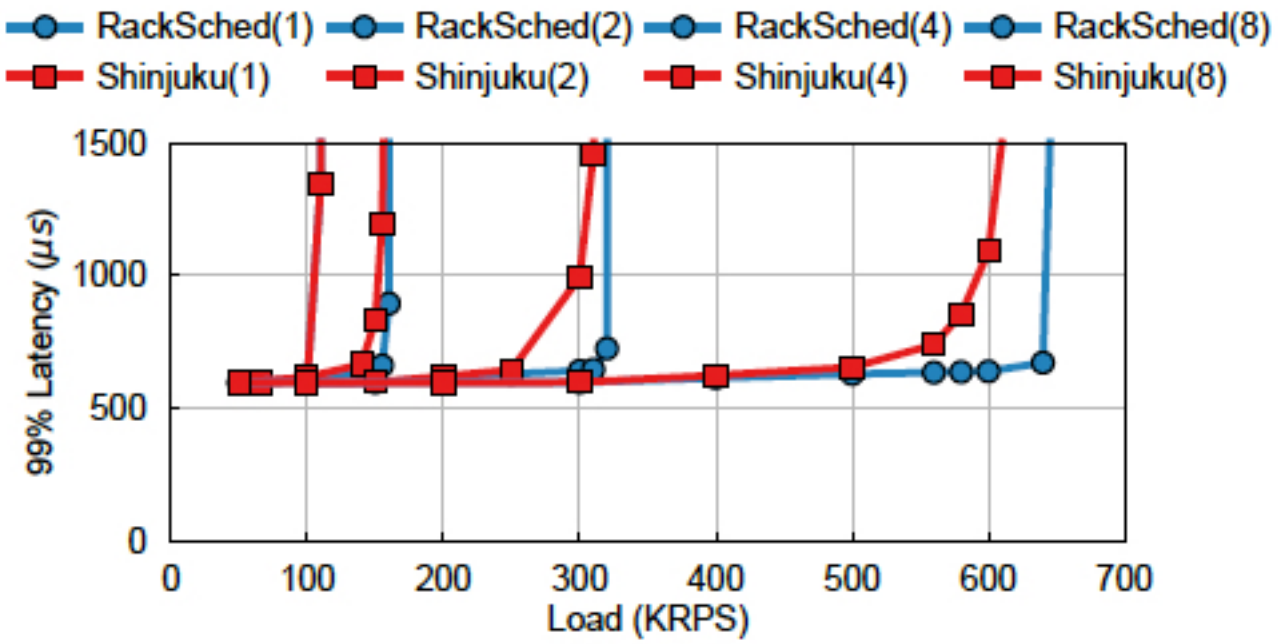
(c) Bimodal(50%-50, 50%-500).



(d) Trimodal(33.3%-50, 33.3%-500, 33.3%-5000).

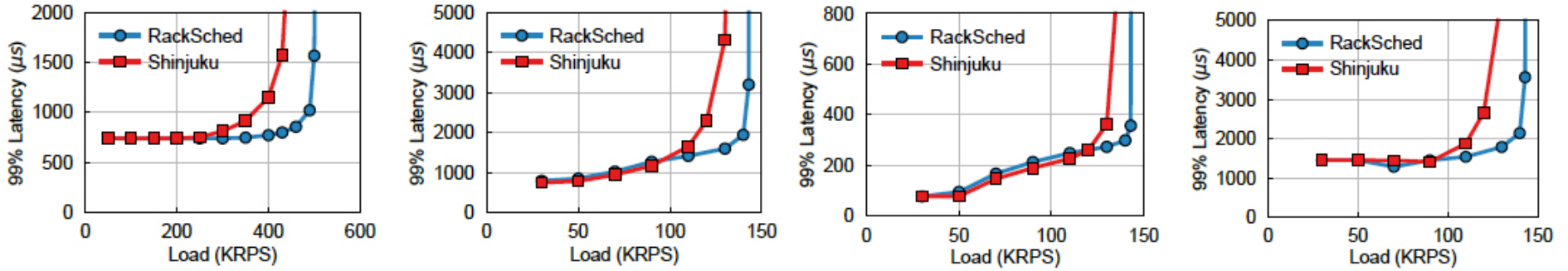
RackSched improves the throughput further by up to 1.44X

Does RackSched scale?



- **RackSched scales out the throughput near linearly**
- **The throughput is increased without increasing the tail latency**

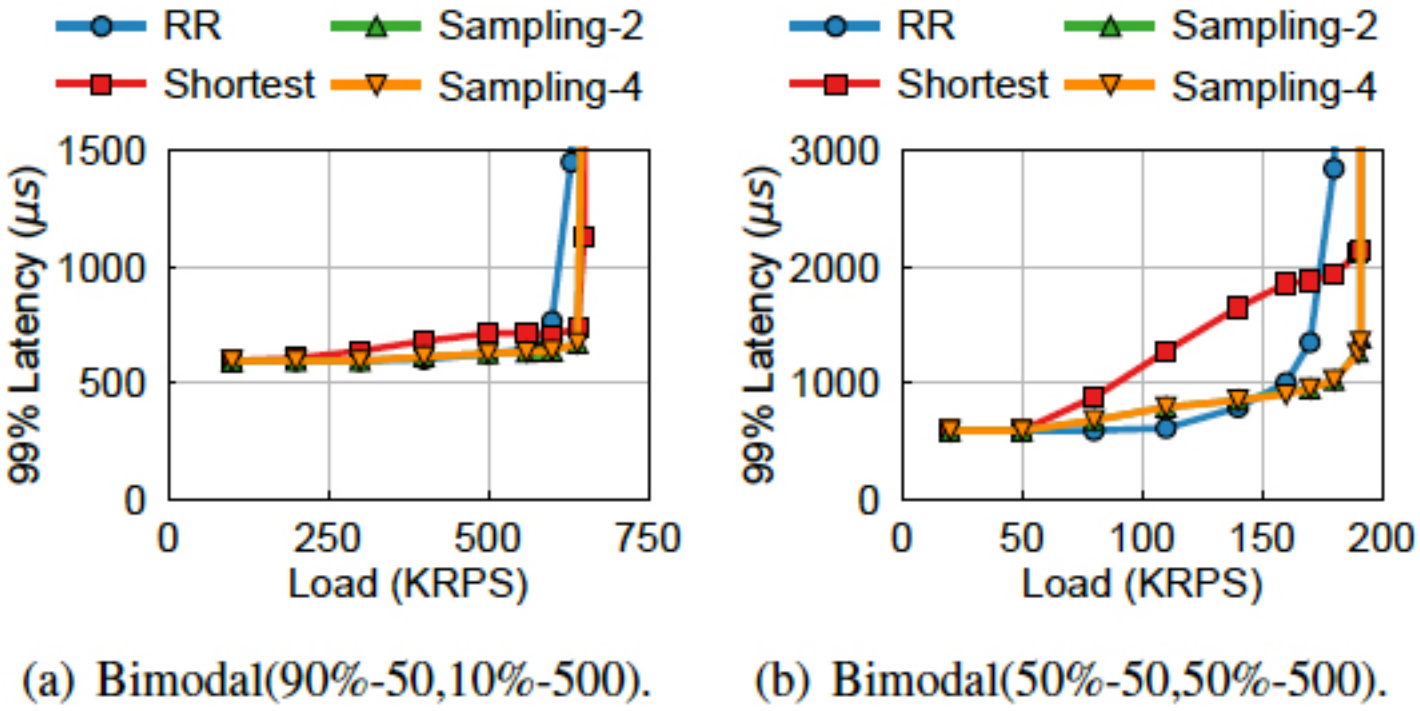
Does RackSched benefit applications?



(a) 90%-GET, 10%-SCAN. (b) 50%-GET, 50%-SCAN. (c) GET in 50%-GET, 50%-SCAN. (d) SCAN in 50%-GET, 50%-SCAN.

RackSched does not sacrifice any individual request type

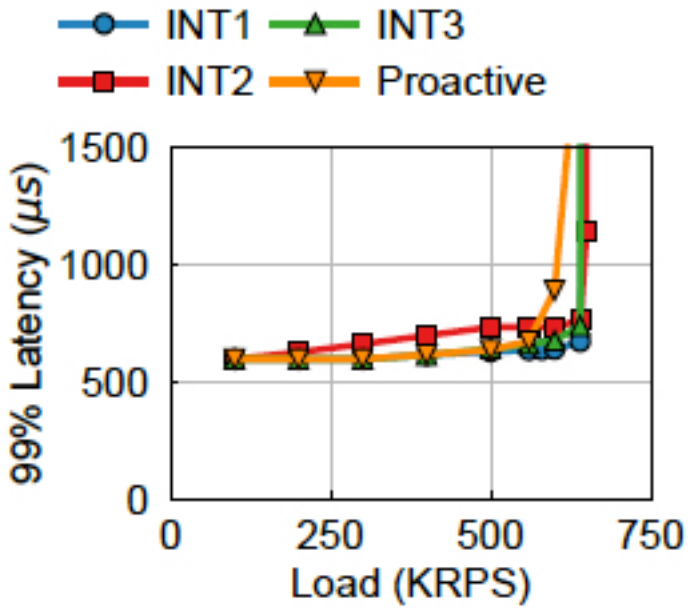
Design decisions: switch scheduling policies



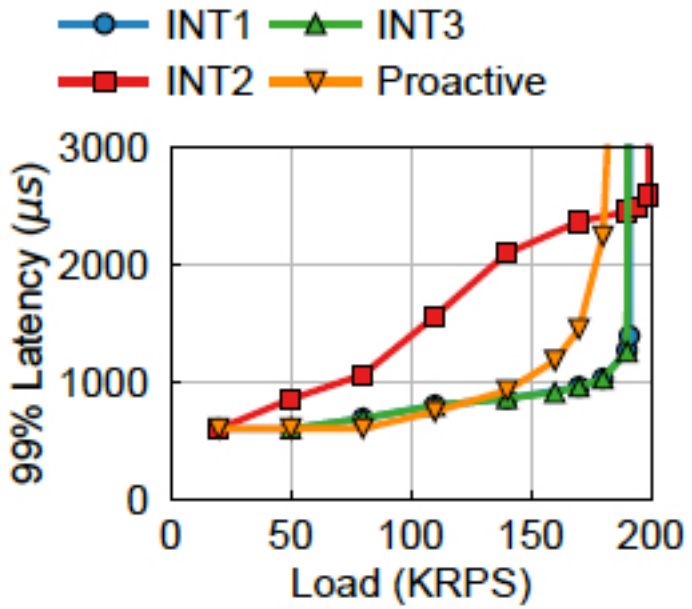
(a) Bimodal(90%-50,10%-500). (b) Bimodal(50%-50,50%-500).

- ***RR*: without considering the variability of service times**
- ***Shortest*: the herding behavior**

Design decisions: load tracking mechanisms



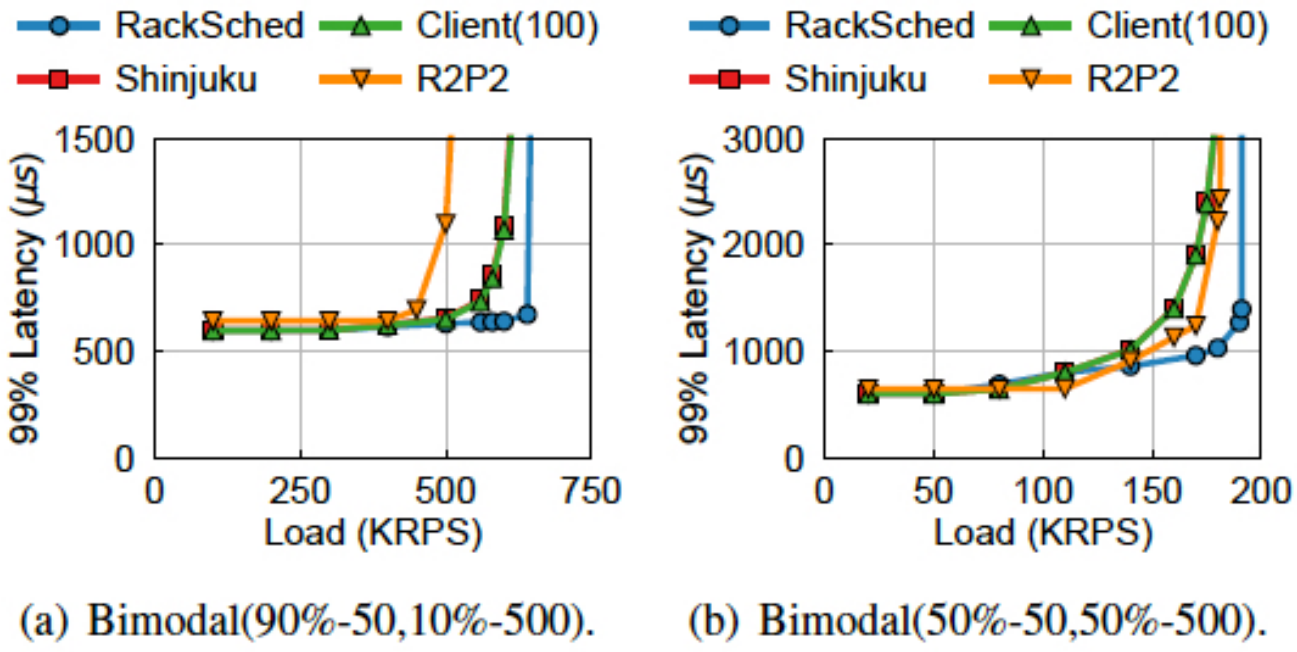
(a) Bimodal(90%-50,10%-500).



(b) Bimodal(50%-50,50%-500).

- **INT2: only track the minimal number of outstanding requests**
- **INT3: track the total service time of outstanding requests**
- **Proactive: increment and decrement the counters by the switch**

Design decisions: comparing with other solutions



(a) Bimodal(90%-50,10%-500). (b) Bimodal(50%-50,50%-500).

- ***Client(100)*** **has nearly the same performance as** ***Shinjuku***
- ***R2P2*** **has head-of-line blocking**

Conclusion

- Emerging workloads require microsecond-scale tail latency

- RackSched is a rack-level microsecond-scale scheduler that achieves scalability and low tail latency
  - Use a two-layer scheduling framework
  - Ensure request affinity
  - Support practical scheduling policies

Thanks!

E-mail address: hzhu@jhu.edu

https://github.com/netx-repo/RackSched