NetLock: Fast, Centralized Lock Management Using Programmable Switches

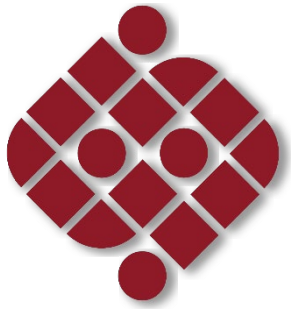
Zhuolong Yu, Yiwen Zhang, Vladimir Braverman, Mosharaf Chowdhury, Xin Jin
in *Proc. of ACM SIGCOMM*, 2020

NSLab Seminar

Gaeun Seo
Networking and Systems Lab.
20211133@sungshin.ac.kr
January 12, 2023

Session

- **SIGCOMM** (**S**pecial **I**nterest **G**roup on Data **Comm**unication)
  - SIGCOMM is ACM's professional forum for discussing communications and computer networks
  - SIG-wide Awards
    - SIGCOMM Award for Lifetime Contribution
    - SIGCOMM Doctoral Dissertation Award
    - SIGCOMM Rising Star Award
    - Test of Time Paper Award
    - IMC Test of Time Award
    - SIGCOMM Networking Systems Award
    - Best Paper Award winners
    - Student Paper Award winners
  - Recent SIGCOMM Best Paper Award Winners
    - 2022 Huangxun Chen
    - 2021 Petros Gigis
    - 2020 João Luís Sobrinho
    - 2019 Junsu Jang and Fadel Adib
    - 2018 Amogh Dhamdhere



Contents

- Introduction
- Background and Motivation
- NetLock Architecture
- NetLock Design
- Evaluation
- Conclusion

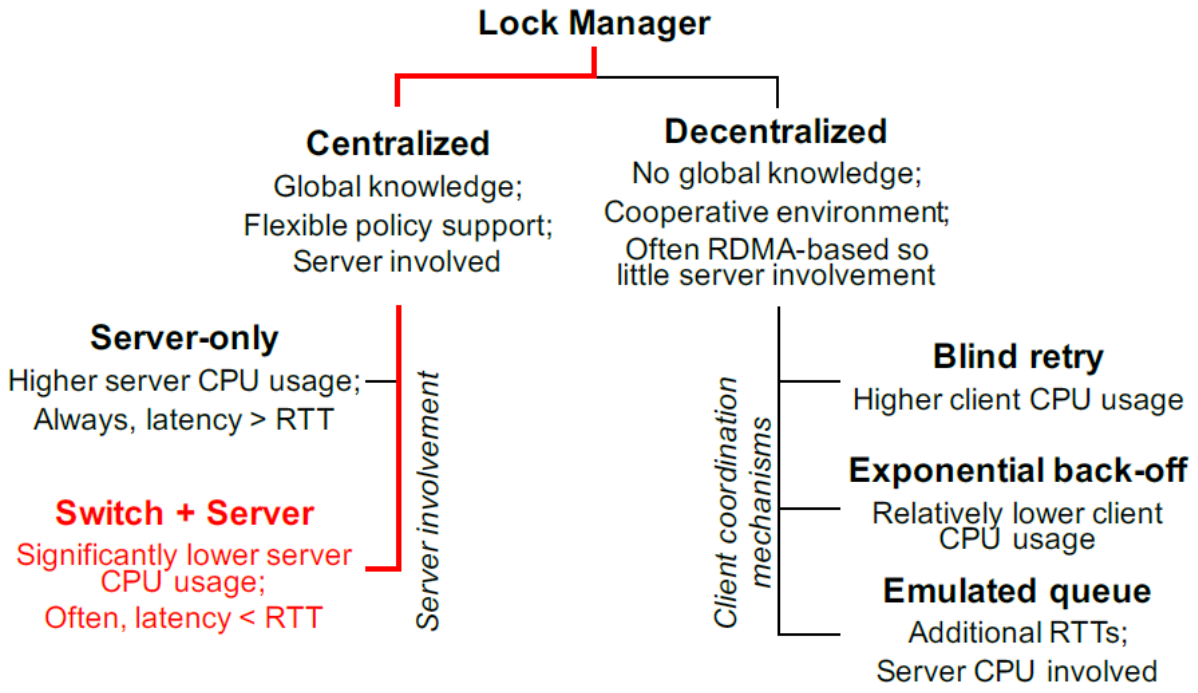
Introduction

- As more and more enterprises move their workloads to the cloud
  - Enterprises are increasingly relying on databases provided by public cloud providers

- Lock managers are a critical building block of cloud databases
  - the overhead of acquiring and releasing locks is now a major component in the end-to-end performance of cloud-based enterprise software
- P4DB hence provides significant benefits compared to traditional DBMS architectures
  - P4DB can achieve a speedup of up to 8×

Background and Motivation

- Existing lock manager
  - face a trade-off between performance and policy support
  - Centralized lock manager
    - Bottleneck at the server's capacity
    - Flexible
  - Decentralized lock manager
    - Usually high performance
    - Hard to support



Design space for lock management

Background and Motivation (Cont'd)

- Existing lock manager
  - face a trade-off between performance and policy support
  - Centralized lock manager
    - Bottleneck at the server's capacity
    - Flexible
  - Decentralized lock manager
    - Usually high performance
    - Hard to support
  - recent decentralized solutions leverage fast RDMA networks to achieve high throughput and low latency
    - Clients acquire and release locks by updating the lock information on the lock server through RDMA, without involving the server's CPU
    - the locking decisions are made by the clients in a decentralized manner, it is hard to support and enforce rich policies

Background and Motivation (Cont'd)

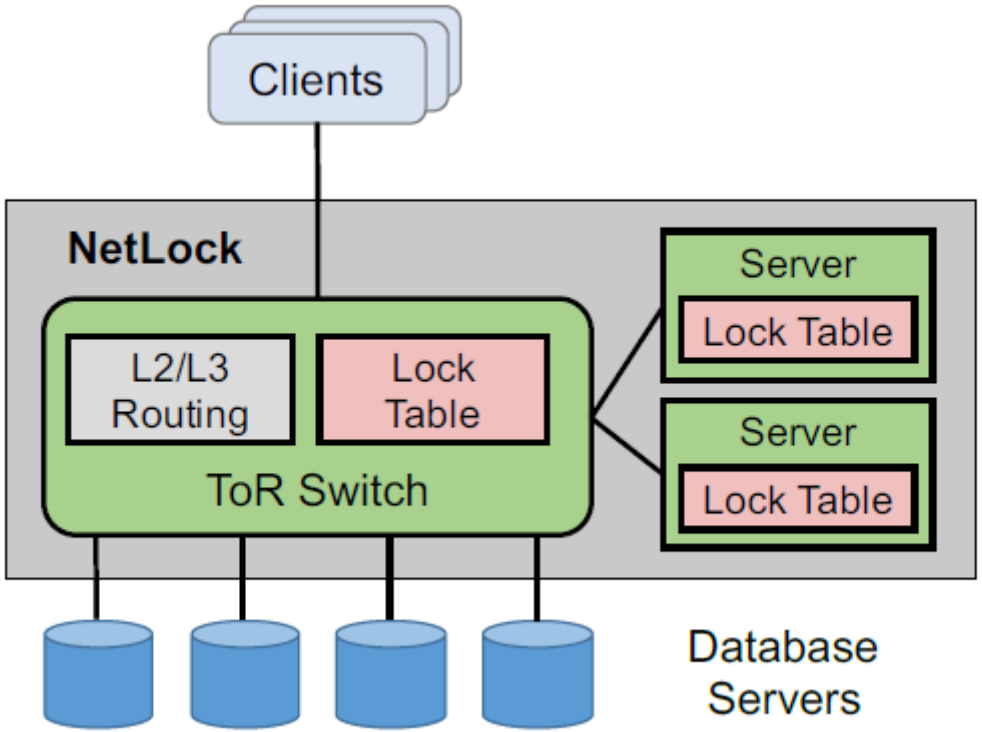
- NetLock
  - a new approach to design and build lock managers that sidesteps the trade-off and achieves both high performance and rich policy support
  - co-design switches and servers to build a fast, centralized lock manager
  - By using switches to process lock requests in the switch data plane, NetLock avoids the CPU bottleneck of server-based centralized approaches, and achieves high performance
    - Switches provide orders-of-magnitude higher throughput and lower latency than servers
  - Challenges
    - Limited memory to store the locks
    - Limited functionalities to process the locks and realize the policies

NetLock Architecture

- High throughput
  - each transaction can involve a few to tens of locks
  - the lock manager should be able to process up to a few billion lock requests per second (RPS)
- Low latency
  - the lock manager should provide low latency to process lock requests, in the range of a few to tens of microseconds
- Policy support
  - provide flexible policy support to accommodate tenantspecific requirements
  - Specifically, we consider common policies including starvation freedom, service differentiation,
    and performance isolation

NetLock Architecture (Cont'd)

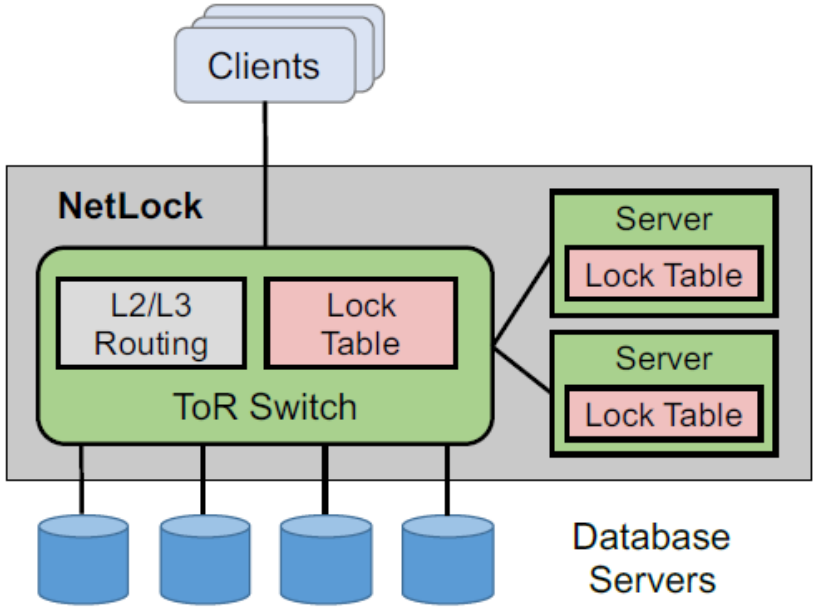
- A NetLock lock manager consists of one switch and multiple servers
- The switch only stores and processes the request on popular locks
- NetLock is incrementally deployable and compatible with existing datacenter networks



Design space for lock management

NetLock Architecture (Cont'd)

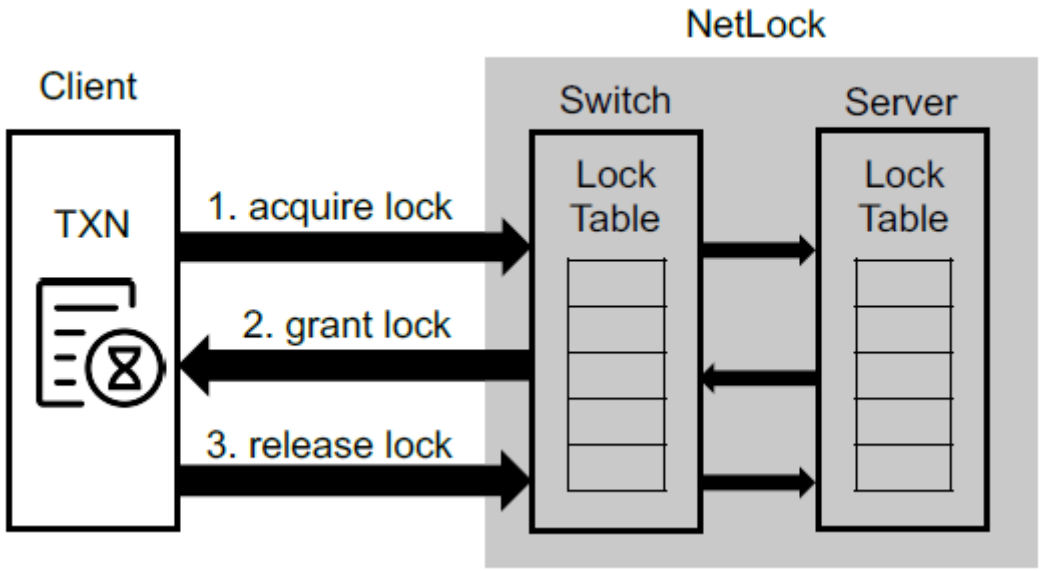
- Sysytem overview at a high level
  - clients send lock requests to NetLock without knowing whether the requests will be processed by a switch or a server
  - NetLock processes lock requests with a combination of switch and servers
  - When a lock request arrives at the switch, the switch checks whether it is responsible for the lock
    - If so, it invokes the data plane module to process the lock
    - otherwise, it forwards the lock requests to the server



Design space for lock management

NetLock Design

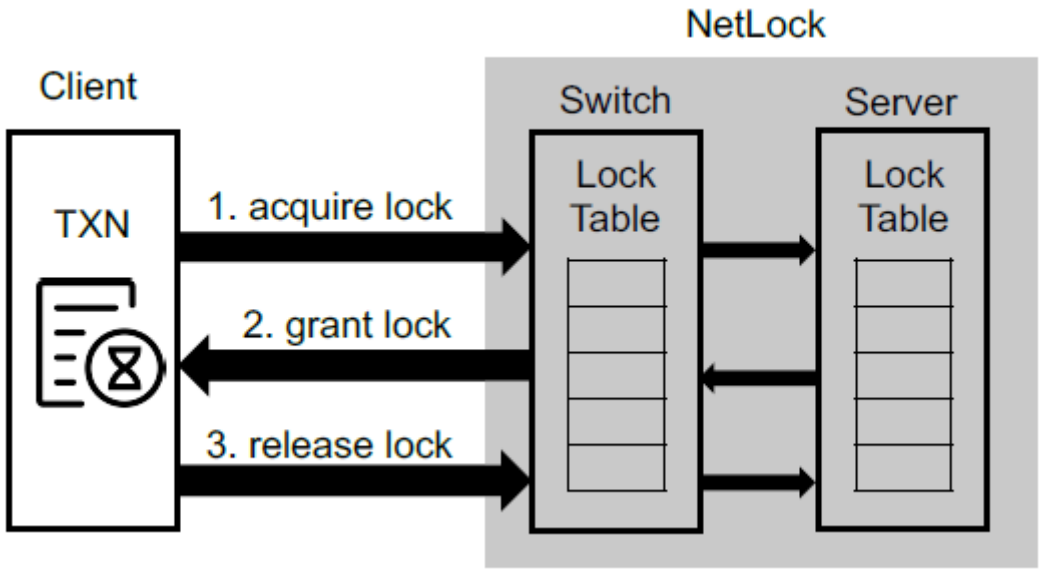
- Lock Request Handling
  - to acquire a lock for a transaction, the client first sends a lock request to NetLock and waits for NetLock to grant the lock
    - NetLock directly processes most lock requests with the lock switch and only leaves a small portion to the lock servers
  - After the lock is granted, the client executes its transaction
    - sends a release notification to NetLock if the lock is no longer needed



Lock request handling in NetLock. The switch directly processes most lock requests

NetLock Design (Cont'd)

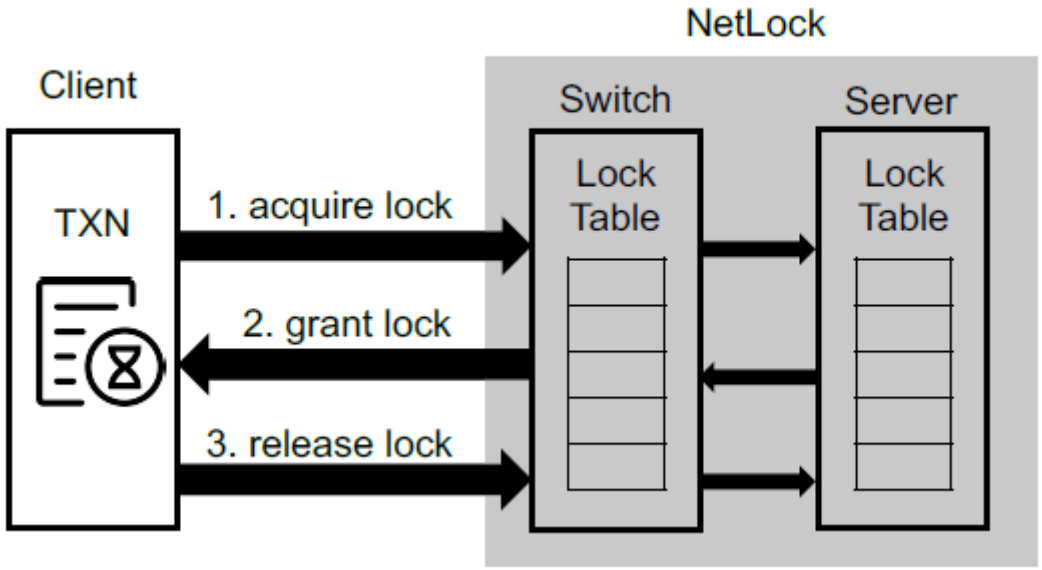
- the pseudocode of the switch
  - the switch is responsible for the corresponding lock object, it checks the lock availability and policy
  - the lock can be granted, the switch directly responds to the client
  - the lock cannot be granted immediately, the switch queues the request if it has enough memory
  - the switch is not responsible for the lock object or does not have sufficient memory, it forwards the request to the lock server based on the destination IP



Lock request handling in NetLock. The switch directly processes most lock requests

NetLock Design (Cont'd)

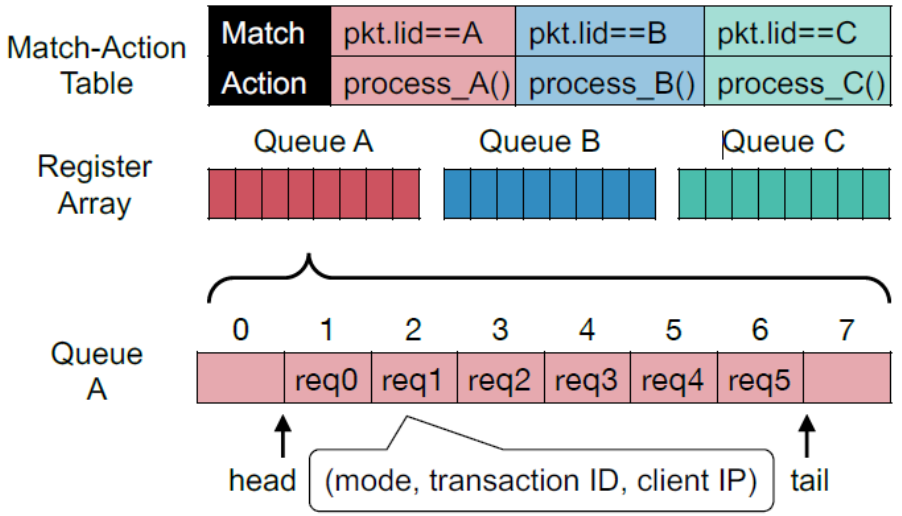
- The locks are partitioned between the lock servers
- The client obtains the partitioning information from an off-the-shelf directory service in datacenters, and sets the destination IP to that of the server responsible for the lock
- The performance benefit of NetLock comes from that most requests can be directly processed by the switch, without the need to visit a lock server



Lock request handling in NetLock. The switch directly processes most lock requests

NetLock Design (Cont'd)

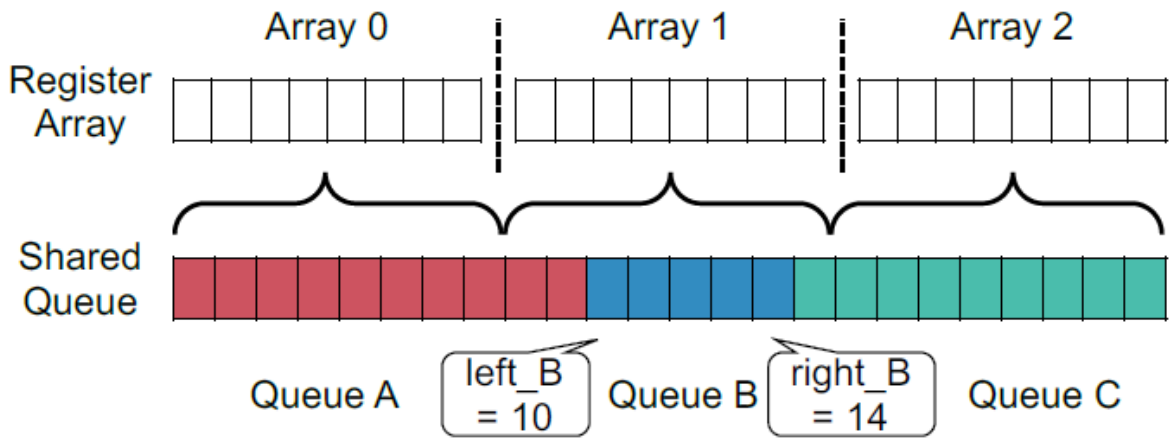
- Switch Data Plane
  - The design allocates one array for each lock to queue its requests
  - A lock request contains several fields: action type (acquire/release), lock ID, lock mode, transaction ID, and client IP
  - The match-action table maps a lock ID (i.e., lid) to its corresponding register array, and the action in the table performs operations on the register array to grant and release locks
  - register arrays can only be accessed based on a given index, they do not natively support queue operations such as enqueue and dequeue
    - implement circular queues based on register arrays to support necessary operations for NetLock



Basic data plane design for lock management

NetLock Design (Cont'd)

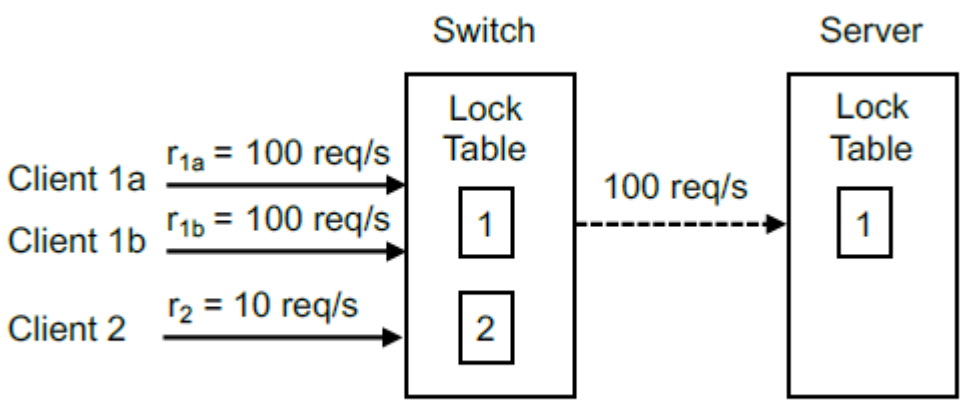
- Optimize switch memory layout
  - the basic design cannot flexibly change the queue size at runtime
    - Because the memory for each register array is pre-allocated and the size is fixed after the data plane program is compiled and loaded into the switch
  - Allocating a large queue to accommodate the maximum possible contentions for each lock is undesirable because it would cause memory fragmentation and result in low memory utilization, especially given that the switch on-chip memory is limited
  - To address this problem, we design a shared queue to pool multiple register arrays together and enable the queue size to be dynamically adjusted at runtime



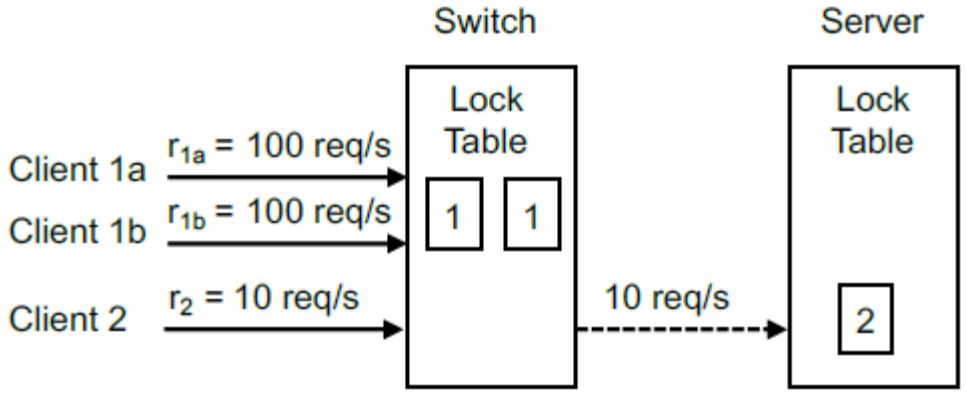
Combine multiple register arrays to a shared queue for locks with different queue sizes

NetLock Design (Cont'd)

- Memory Allocation
  - Process as many lock requestions in the switch as possible
  - Clients send lock requests to NetLock without knowing where the requests will be processed



(a) Naive memory allocation.

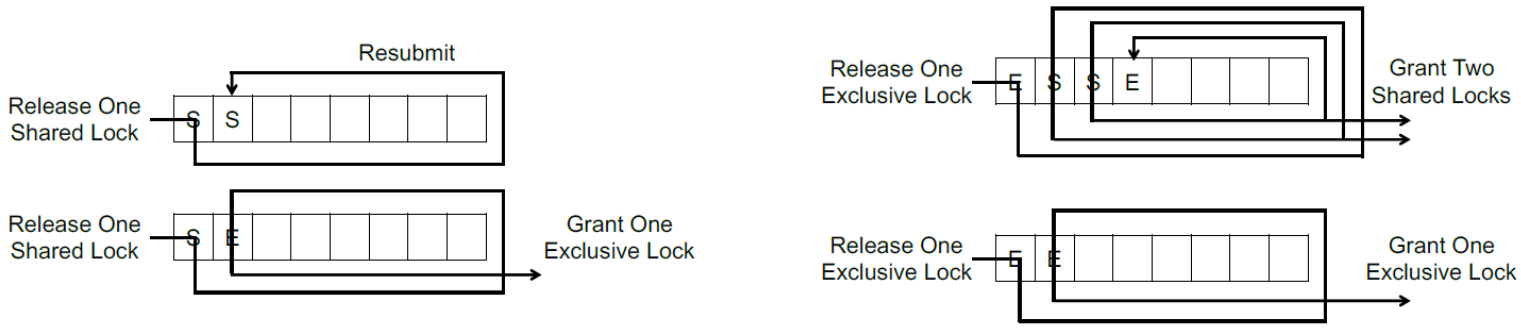


(b) Optimal memory allocation.

By allocating two slots in the switch to lock 1, the optimal allocation can process all lock requests to lock 1 in the switch, minimizing the server load

NetLock Design (Cont'd)

- how to handle the four cases for shared and exclusive locks
  - Shared → shared
    - When a shared lock is released, the switch dequeues the head, and uses resubmit to check the new head. If the new head is a shared lock request, the processing stop
  - shared → exclusive
    - This case differs from the first case on that the new head is an exclusive lock request, which has not been granted yet
  - exclusive → shared
    - the packet is resubmitted to grant the next lock request in the queue
  - exclusive → exclusive
    - the lock is exclusive and cannot be shared, the switch does not need to resubmit it again



Handle shared and exclusive locks

NetLock Design (Cont'd)

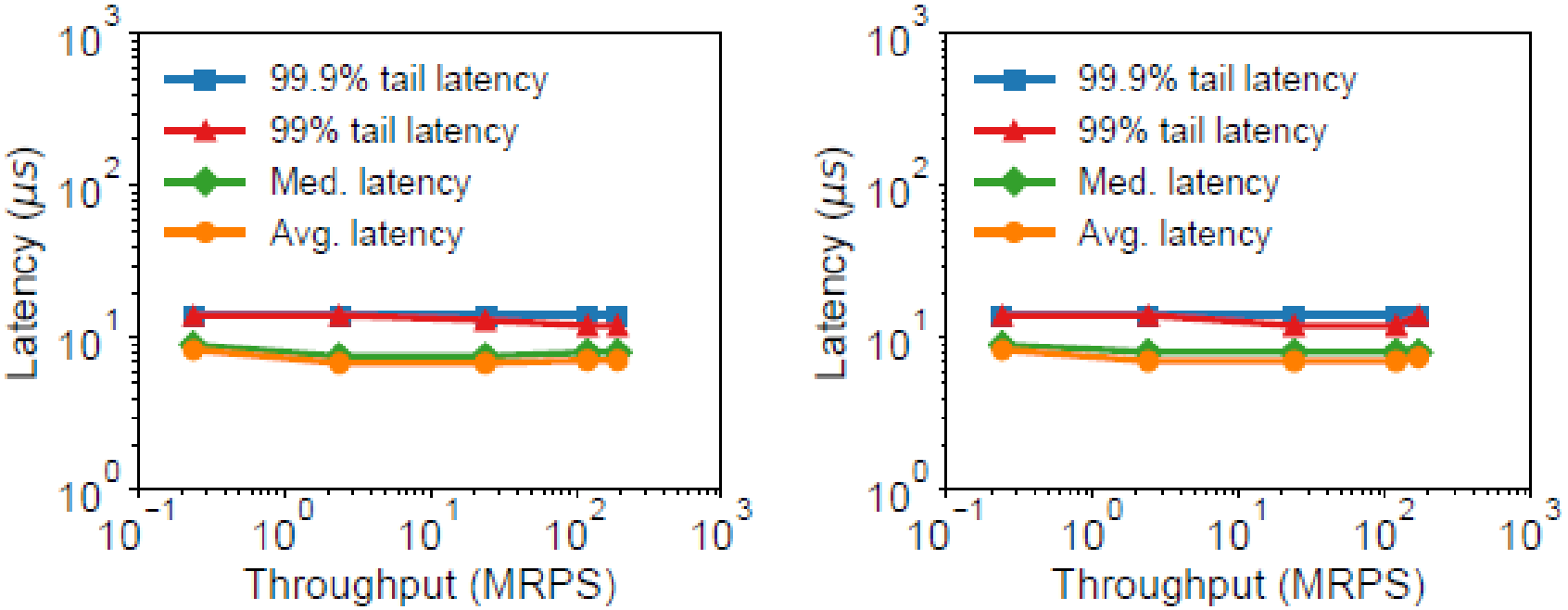
- Policy Support
  - Starvation-freedom
    - The requests are in FCFS queue
  - Service differentiation with priority
    - One priority per stage
    - Low-priority can be offloaded to the lock servers
  - Performance Isolation
    - A rate limiter is implemented to enforce a qutoa for each tenant

Evaluation

- Testbed
  - 6.5 Tbps Barefoot Tofino switch
  - 12 servers with an 8-core CPU and a 40G NIC
- Comparison
  - DSLR, DrTM (RDMA based)
  - NetChain(Programmable Switch)
- Workload
  - Microbenchmark(simple lock requests)
  - TPC-C with 1/10 warehouse per node

Evaluation (Cont'd)

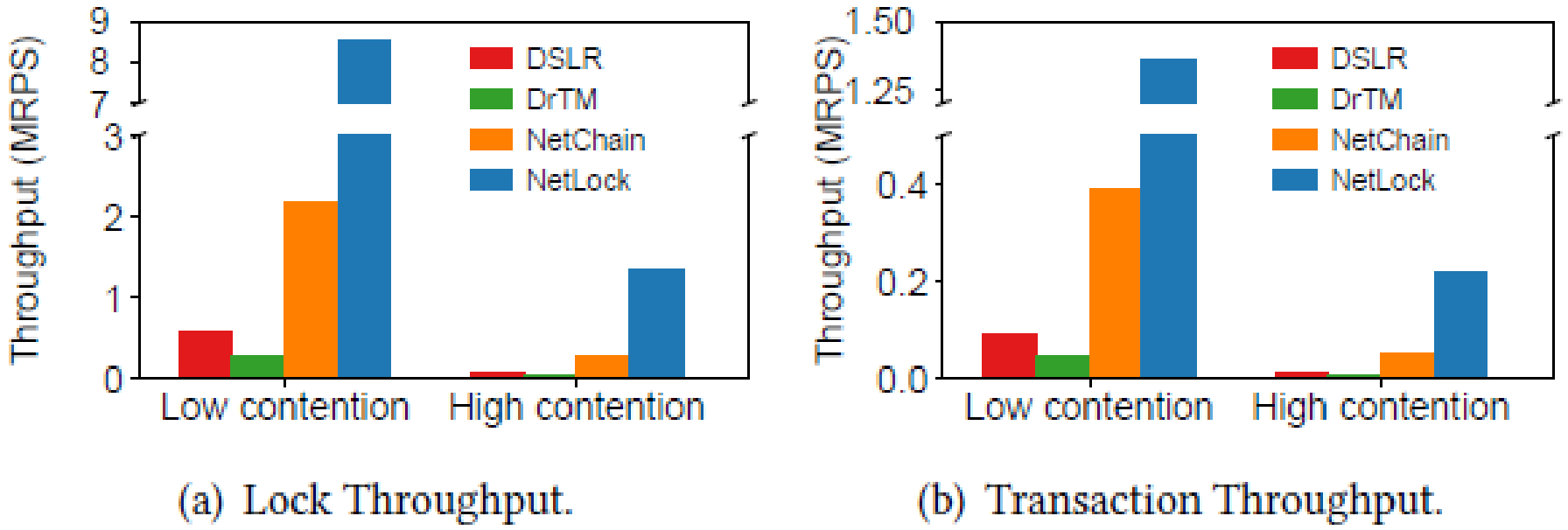
- Microbenchmark
  - 12 servers cannot saturate the switch which is able to process billions of packets per second



Microbenchmark results of switch performance on handling lock requests

Evaluation (Cont'd)

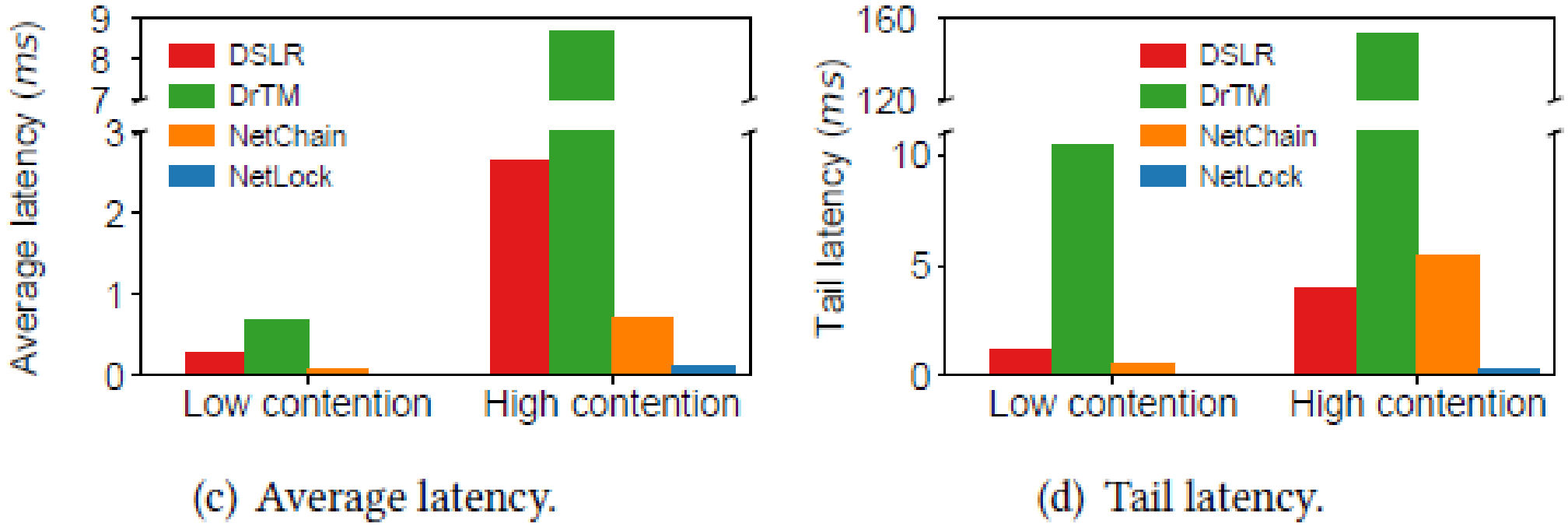
- Transaction performance
  - NetLcok improves the throughtput by 14.0-18.4x compared with DSLR



System comparison under TPC-C with ten clients and two lock servers

Evaluation (Cont'd)

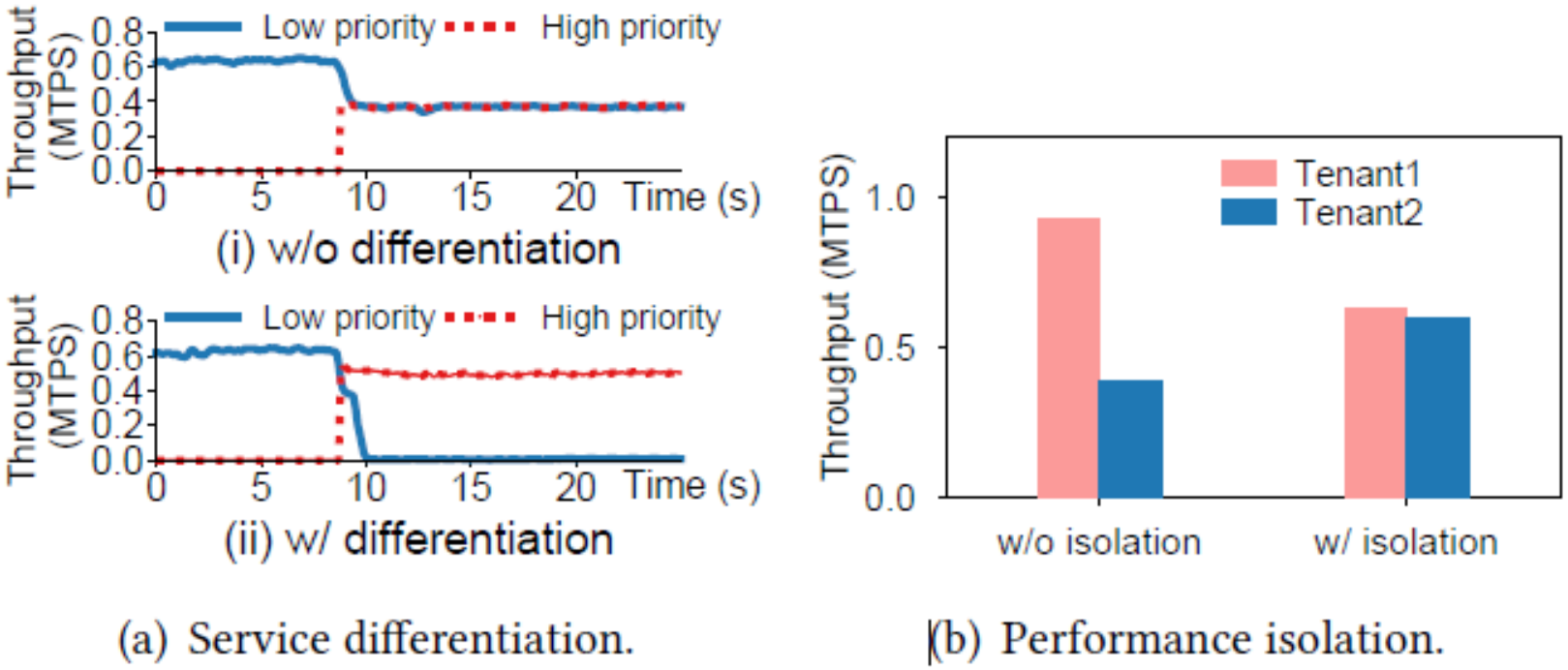
- Transaction performance
  - NetLcok also reduces the latency by 4.7-20.3x compared with DSLR



System comparison under TPC-C with ten clients and two lock servers

Evaluation (Cont'd)

- Service differentiation
  - Two clients get similar throughput regardless of priority
  - High priority client gets much higher throughput
- Performance isolation
  - After enforcing isolation, two tenants get similar throughput



(a) Service differentiation. (b) Performance isolation.

Policy support of NetLock

Conclusion

- NetLock is a centralized lock manager
- Co-design programmable switches and servers
- Provide high performance and flexible policy support