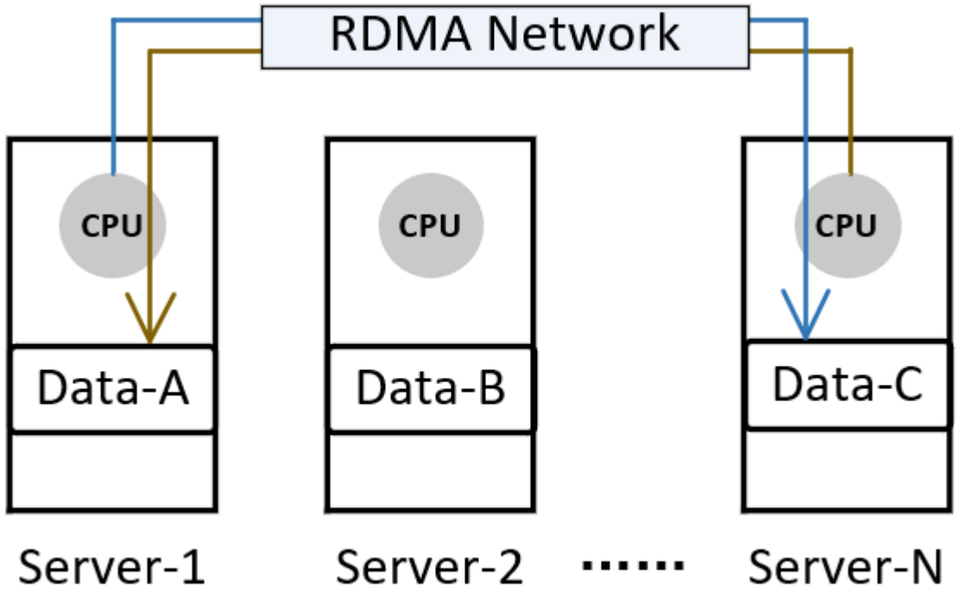
Fast Distributed Transactions for RDMA-based Disaggregated Memory

Haodi Lu, Haikun Liu, Yujian Zhang, Zhuohui Duan,
Xiaofei Liao, Hai Jin, Yu Zhang

https://github.com/CGCL-codes/HDTX

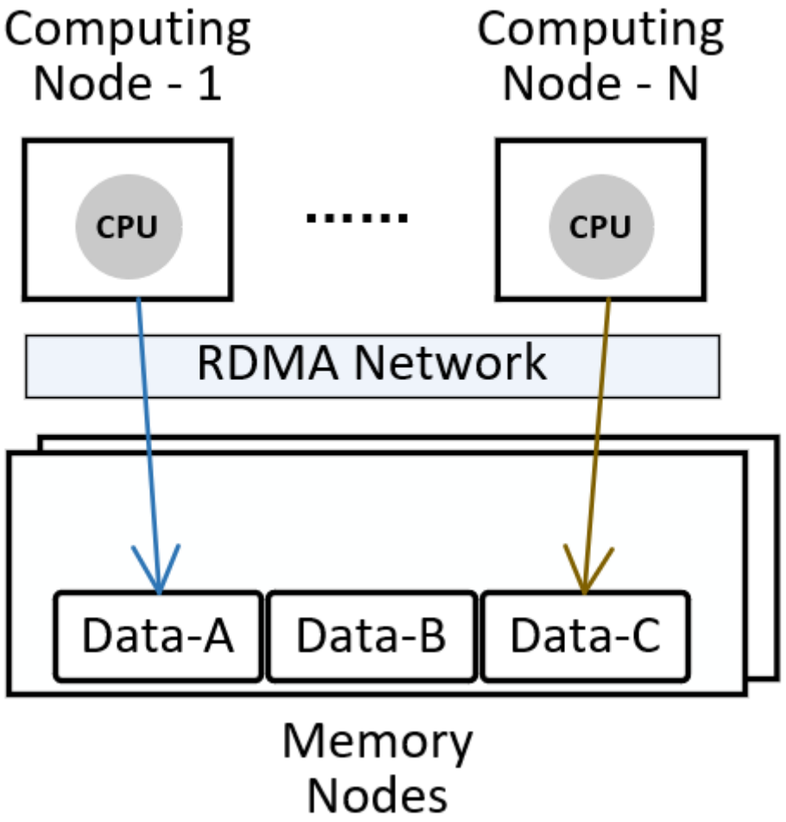
National Engineering Research Center for Big Data Technology and System
Services Computing Technology and System Lab, Cluster and Grid Computing Lab
School of Computer Science and Technology, Huazhong University of Science and Technology, China

Background



Monolithic Servers

- ✓ High Resource Utilization
- ✓ Resource Scalability
- ✓ Failure Isolation



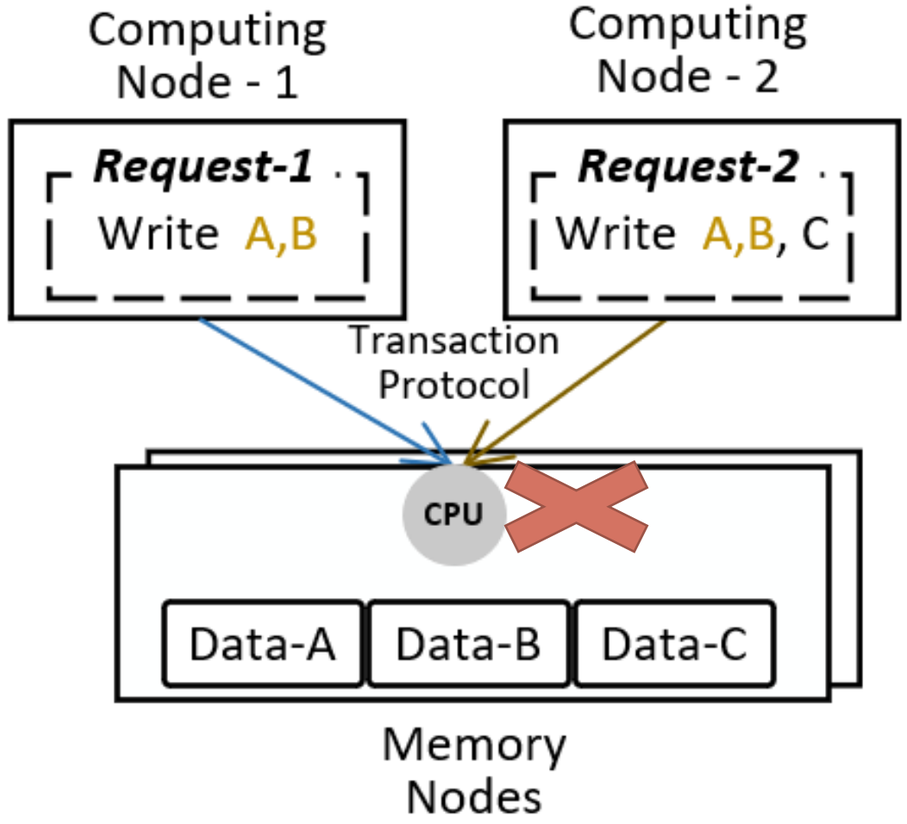
Disaggregation Memory (DM)

Memory disaggregation can improve resource utilization, and facilitate scaling and failure isolation.

Background

Computing nodes(CNs) exploit distributed transactions(Dtxn) to access shared objects with integrity and consistency

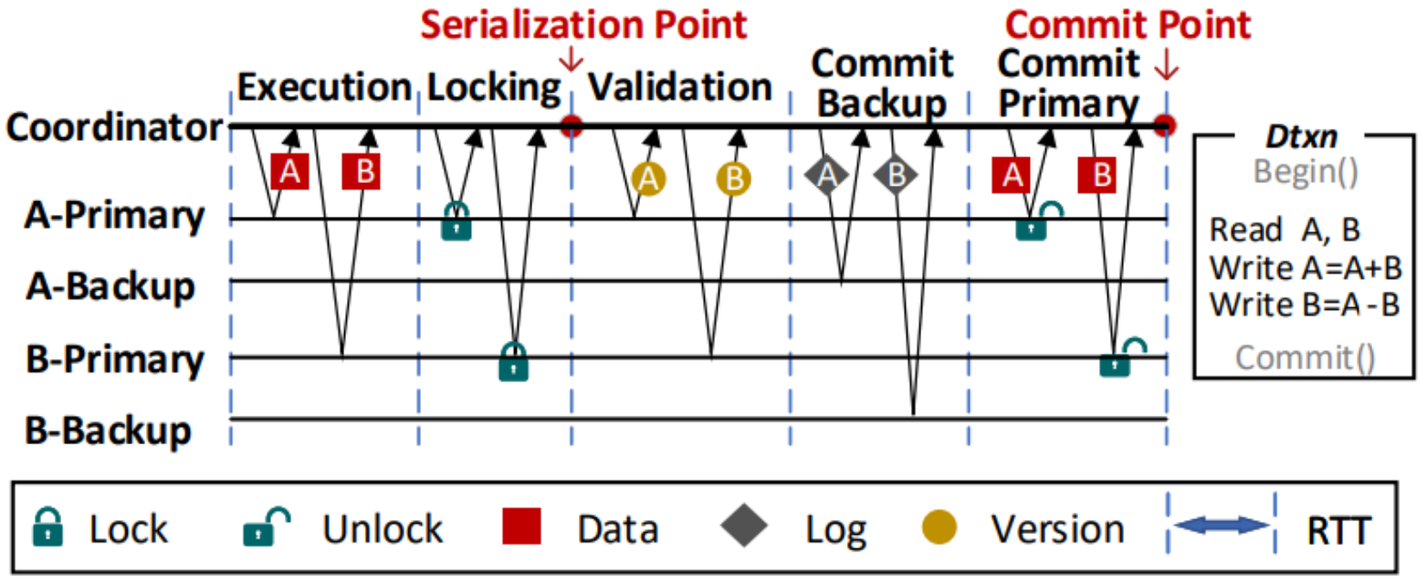
Unlike monolithic servers, the disaggregated memory nodes only has limited computing resource for initialization.



Poor computing resource on DM nodes challenges the efficiency of dtxns.

Motivation

- Challenge One
  Multiple network round trips (RTT) caused by multi-phase dtxn processing

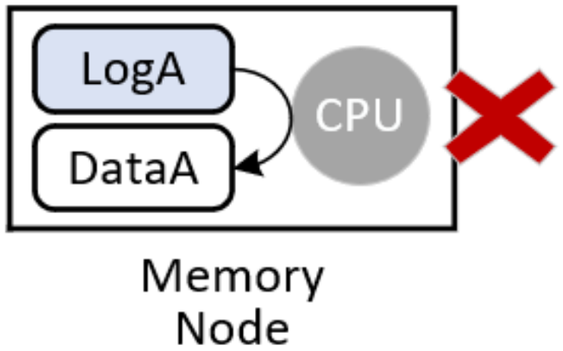


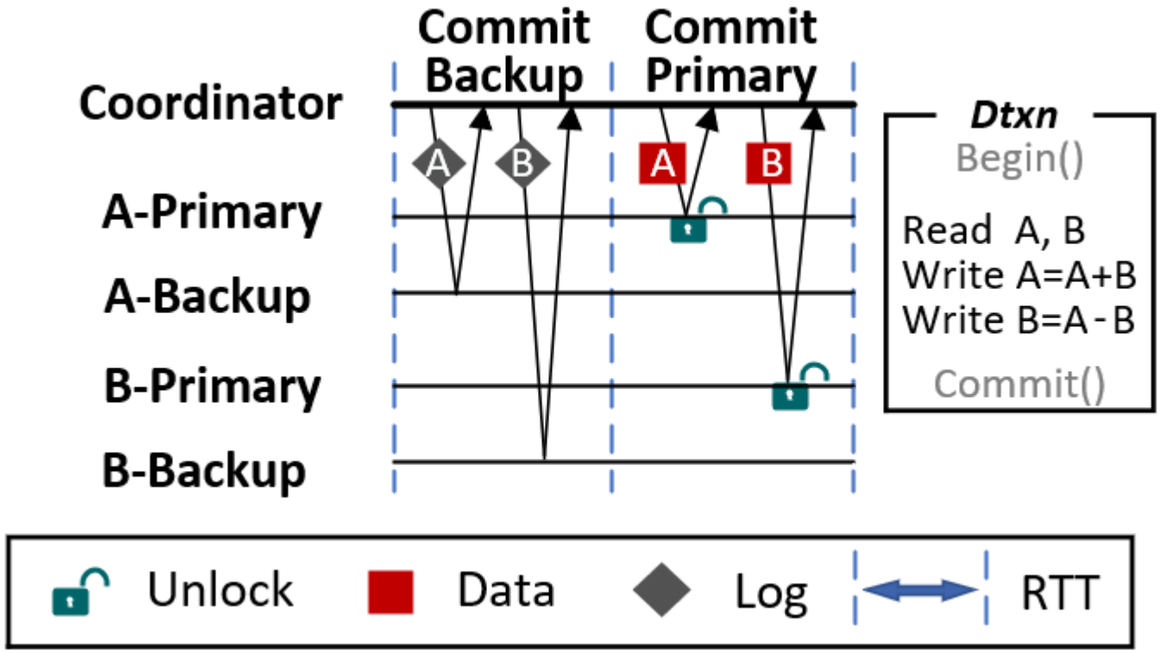
- Current dtxn systems using OCC and RBR require at least 3 RTTs to process a single dtxn.

Dtxn Systems	Phases of dtxn protocol	# RTTs to commit
FaRM [12, 39]	*Execution + Locking + Validation + Commit Backup + Commit Primary*	5 RTTs
FaSST [28]	*Execution&Locking + Validation + Log + Commit Backup + Commit Primary*	5 RTTs
DrTM+H [48]	*Execution + Locking&Validation + Commit Backup + Commit Primary*	4 RTTs
FORD [53]	*Execution&Locking (UndoLog) + Validation + Commit + Background Release*	3 RTTs

Motivation

- Challenge Two
  Inefficient data synchronization in dtxn commit phases





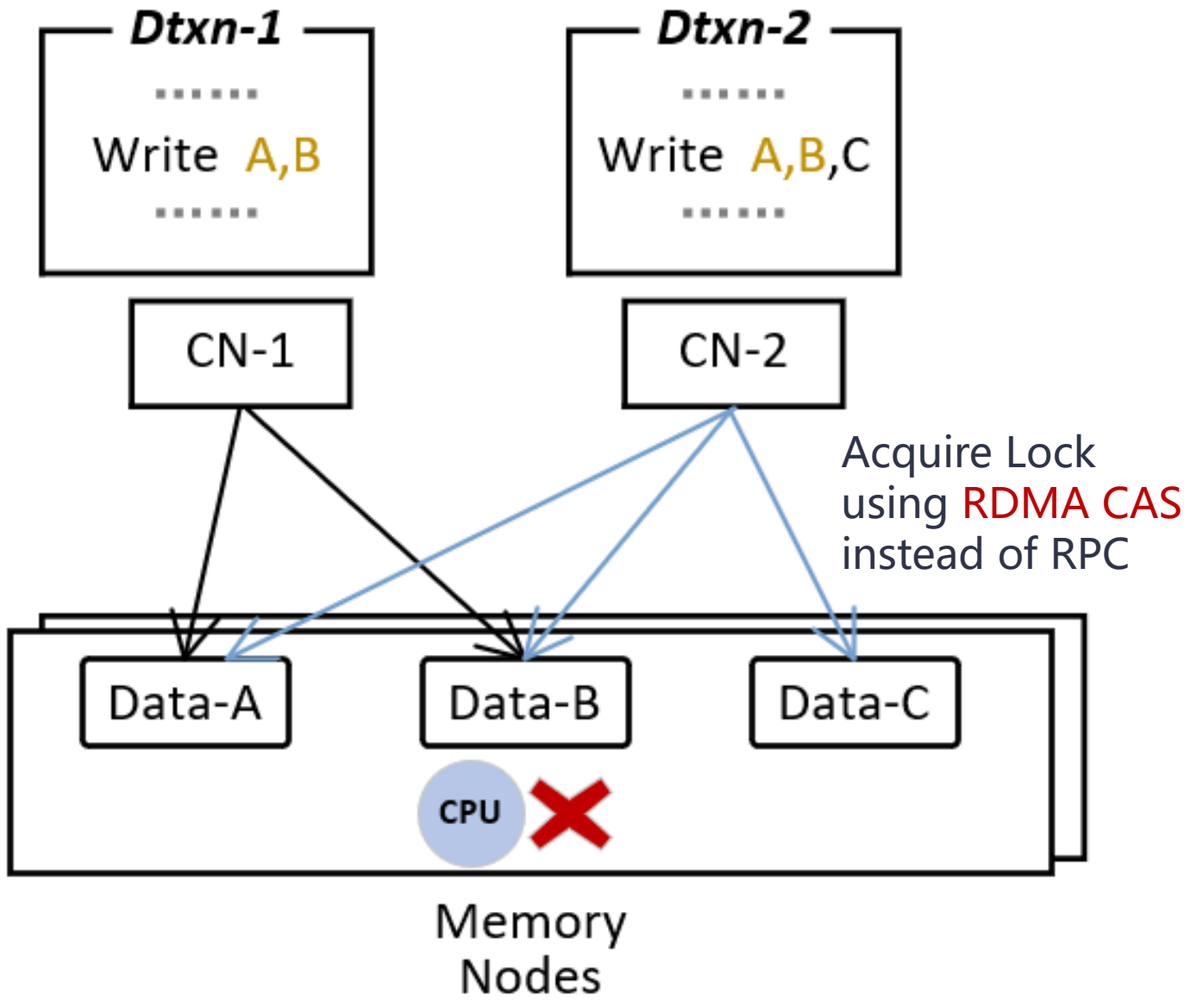
- Logging can guarantee atomicity, however, there is no CPU available for data synchronization in a DM scenario

- Two rounds of transfers are required to send the log and the data from computing nodes to all memory nodes

Motivation

- Challenge Three
  Limited CPU resource in DM nodes for scheduling mission-critical dtxns.

- RDMA CAS primitives with blind retries can lead to unpredictable performance and starvation issues.



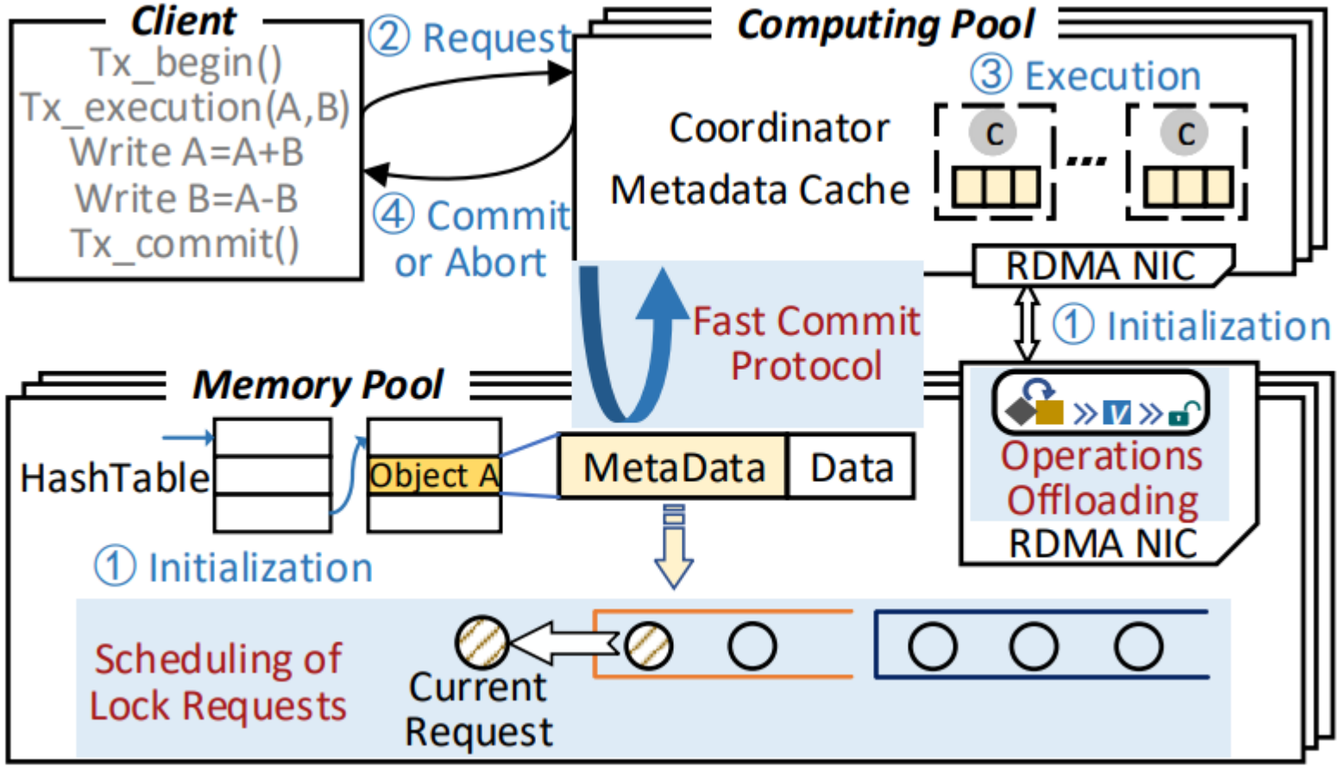
Design

- HDTX
  A high-performance distributed transaction system for RDMA-based disaggregated memory

✓ Fast commit protocol

✓ RDMA-enabled offloading

✓ Decentralized priority-based locking



Design

I. Fast commit protocol
   a) Combining Backup and Primary Commit Phases

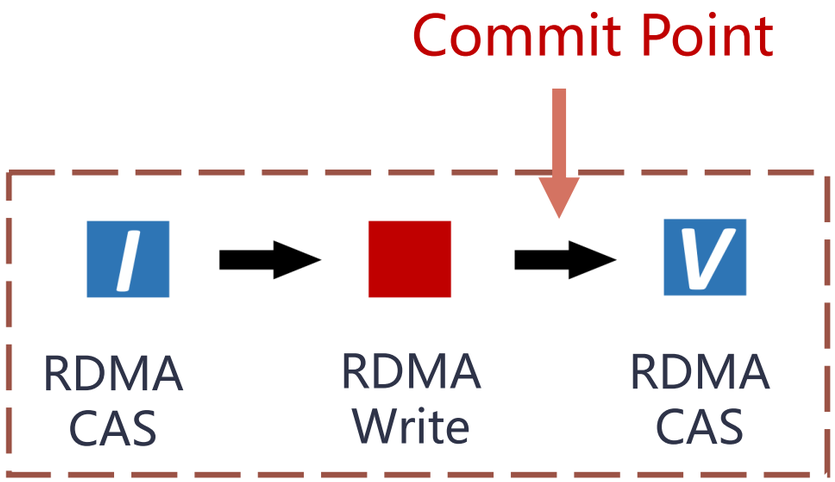
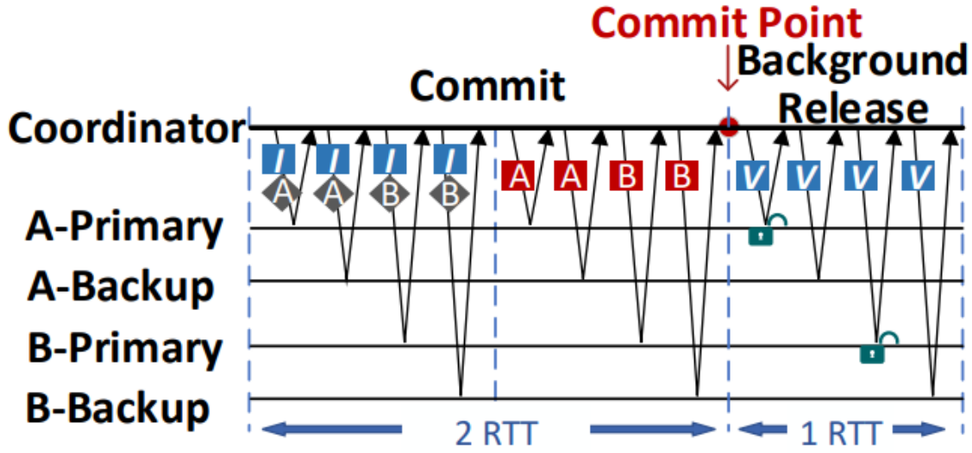


Table 1: Work Request Operation Ordering

First Operation \ Second Operation	Send	Write	Read	Atomic (FAA/CAS)
Send	#	#	#	#
Write	#	#	#	#
Read	F	F	#	F
Atomic (FAA/CAS)	F	F	#	F

*F: Order is maintained only if the second operation has set a Fence flag.
*#: Order is always maintained.



(a) Using undo log

Design

I. Fast commit protocol
   a) Combining Backup and Primary Commit Phases

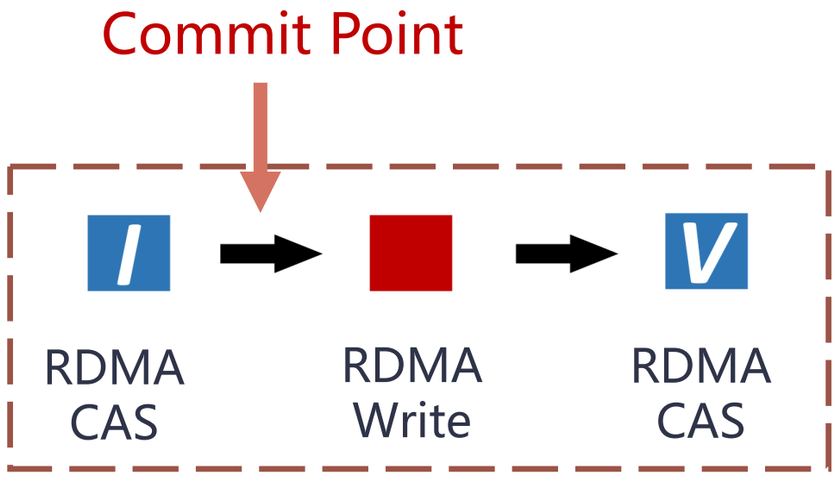
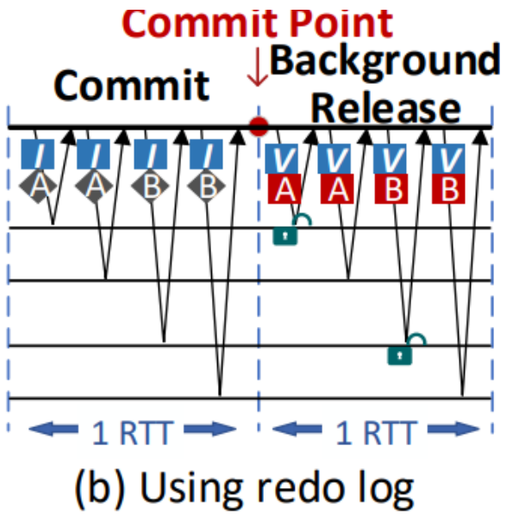


Table 1: Work Request Operation Ordering

First Operation \ Second Operation	Send	Write	Read	Atomic (FAA/CAS)
Send	#	#	#	#
Write	#	#	#	#
Read	F	F	#	F
Atomic (FAA/CAS)	F	F	#	F

*F: Order is maintained only if the second operation has set a Fence flag.
*#: Order is always maintained.



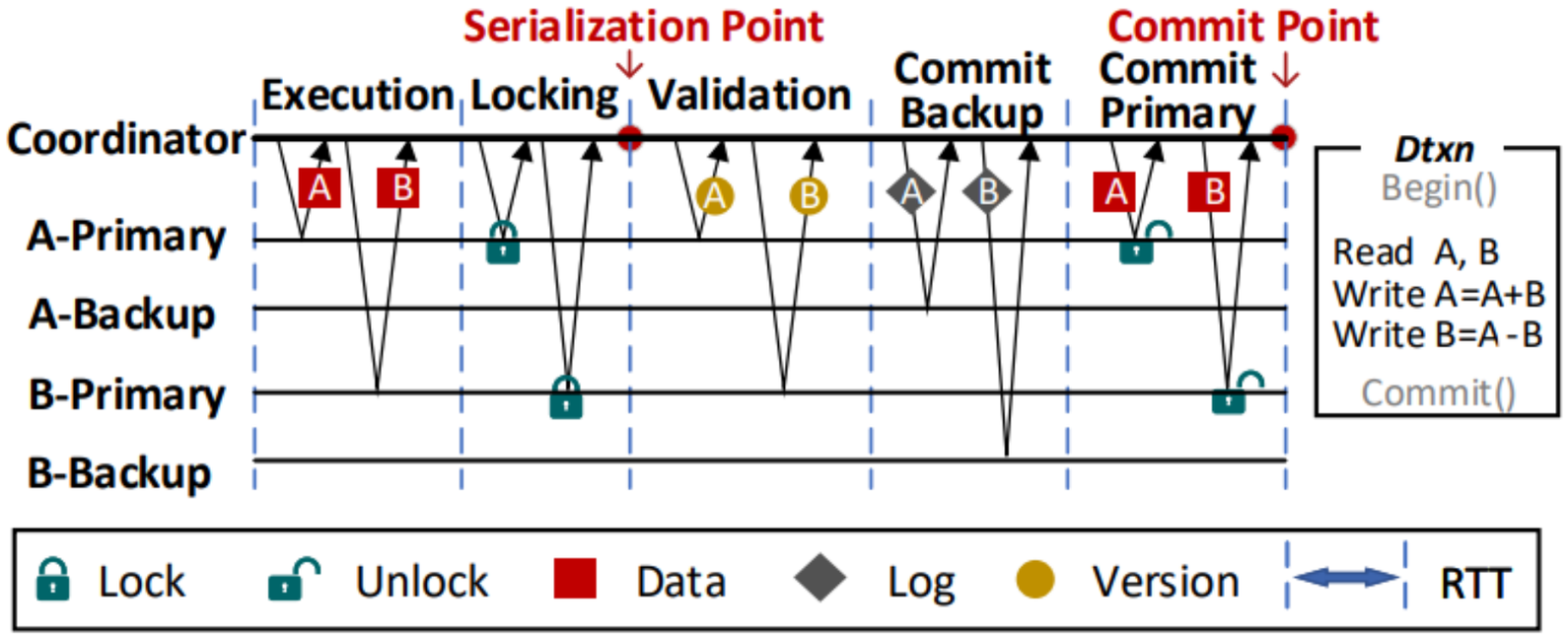
(b) Using redo log

Due to the limitation of RDMA operation ordering, we adopt the redo log and the visibility control mechanism for fast commits.

Design

I. Fast commit protocol

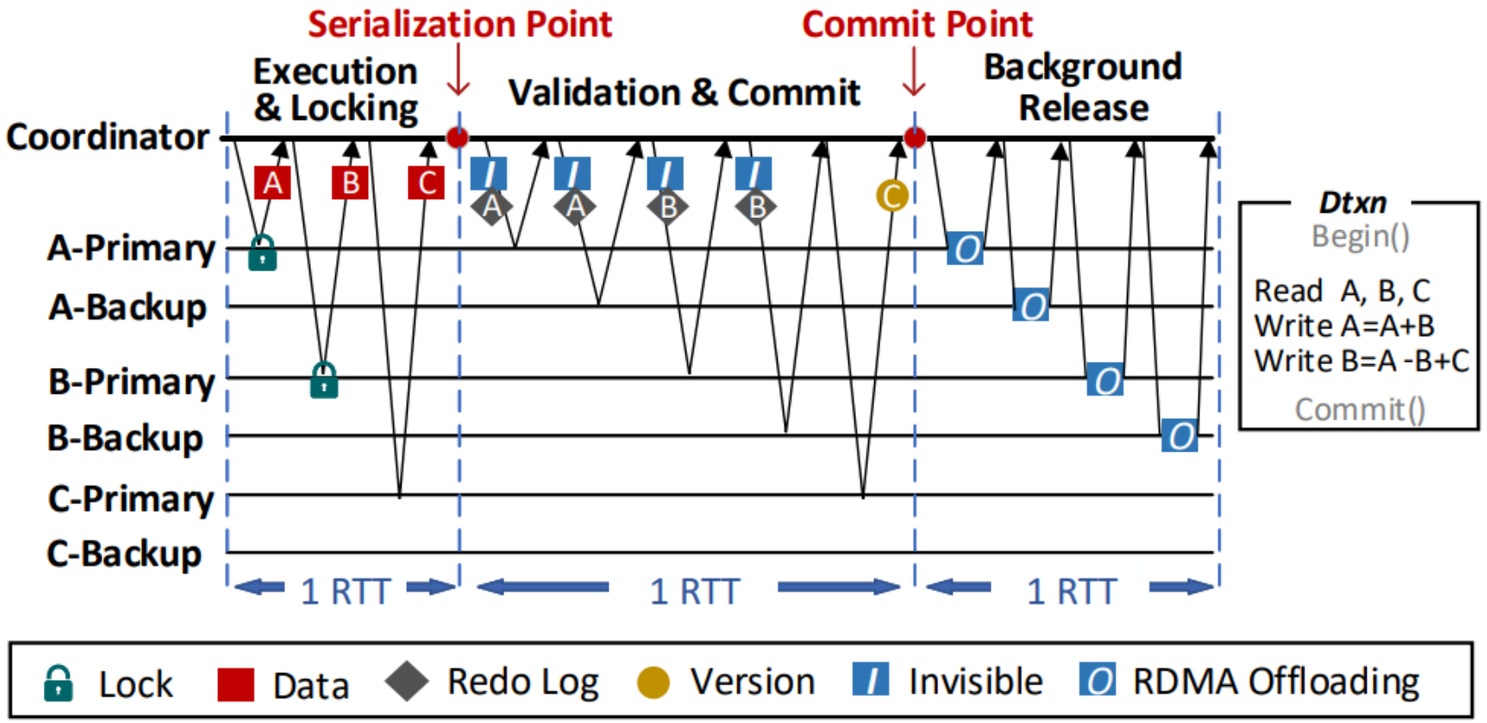
b) Combining Validation and Commit Phases



Since there is no data dependency between the Validation phase and the Commit phase, HDTX combines these two phases to reduce RTTs.

Design

I. Fast commit protocol
b) Combining Validation and Commit Phases



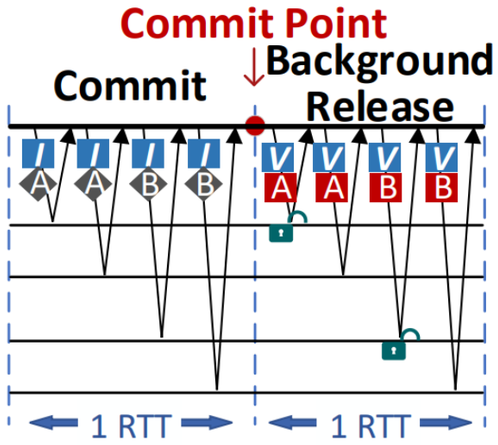
(1) Execution & Locking phase
(2) Validation & Commit phase
(3) Release phase.

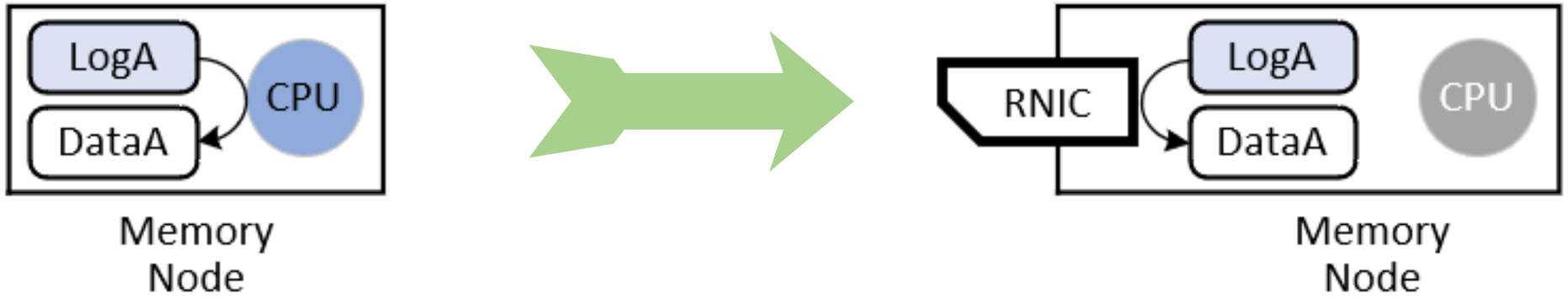
Since there is no data dependency between the Validation phase and the Commit phase, HDTX combines these two phases to reduce RTTs.

Design

II. RDMA-enabled Release Phase Offloading

- It is costly to send the latest data and the log in two rounds of data transfers.





Monolithic servers rely on CPUs to perform data synchronization

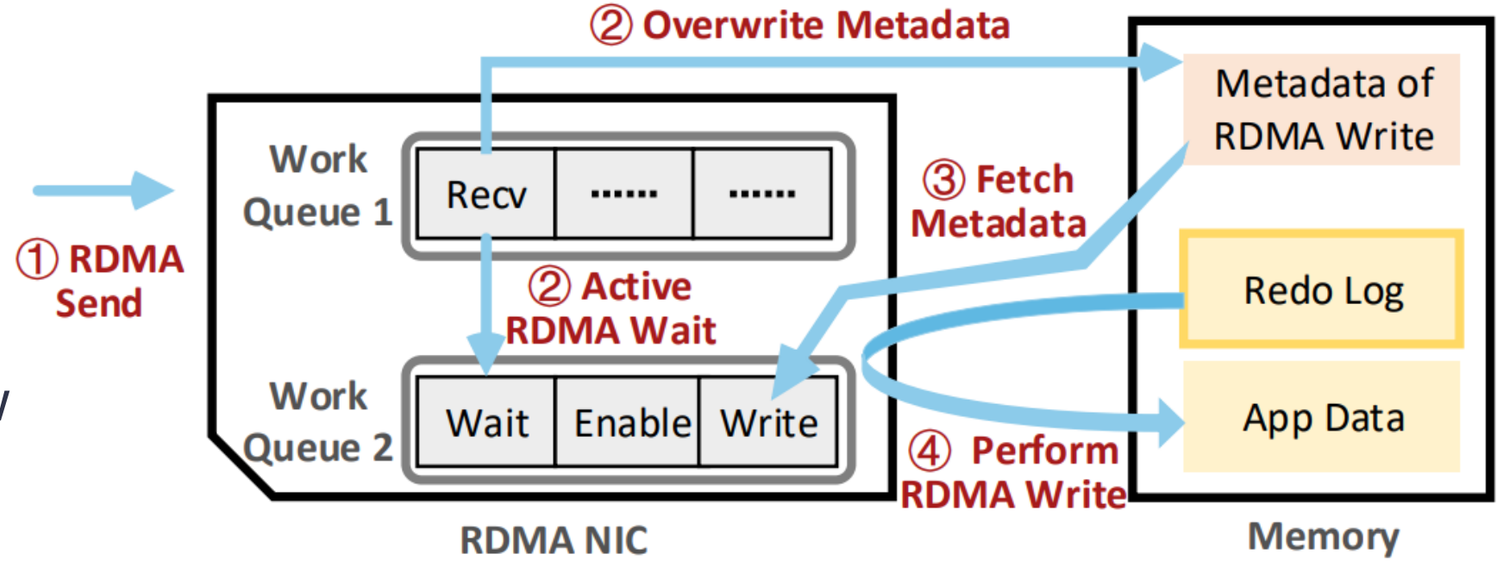
Offloading the data synchronization to RDMA NIC

II. RDMA-enabled Release Phase Offloading

The memory node's RNIC can "copy" the redo log to update the data locally via RDMA Wait and Enable primitives

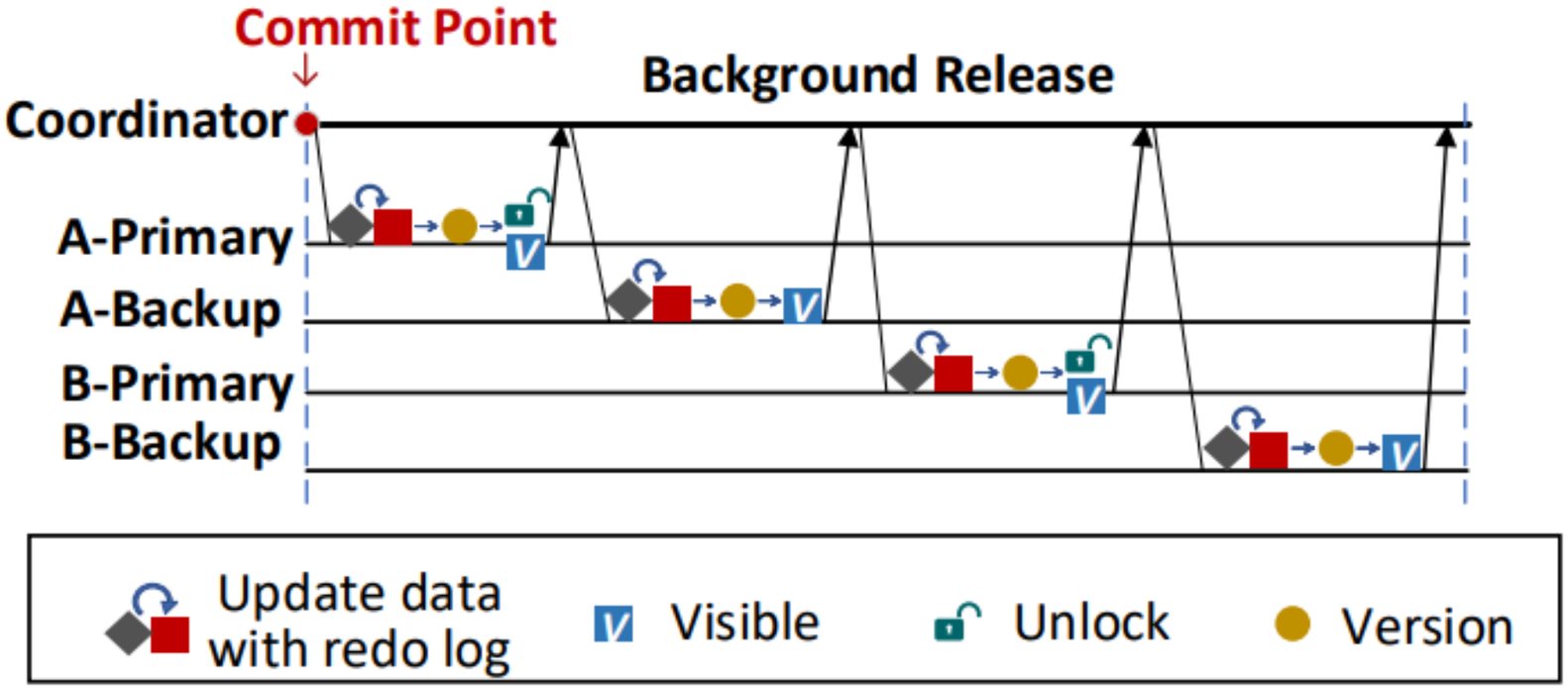
Blocking RDMA Write with RDMA Wait. Then:

① Send the addresses of log and data
② Overwrite the metadata of RDMA Write
③ Active the RDMA Write with new metadata
④ Perform "copy" operation



II. RDMA-enabled Release Phase Offloading

Two RDMA Write and one RDMA atomic primitives are used to update data with redo log, increase the version and unlock the data



Design

III. Decentralized Priority Locking

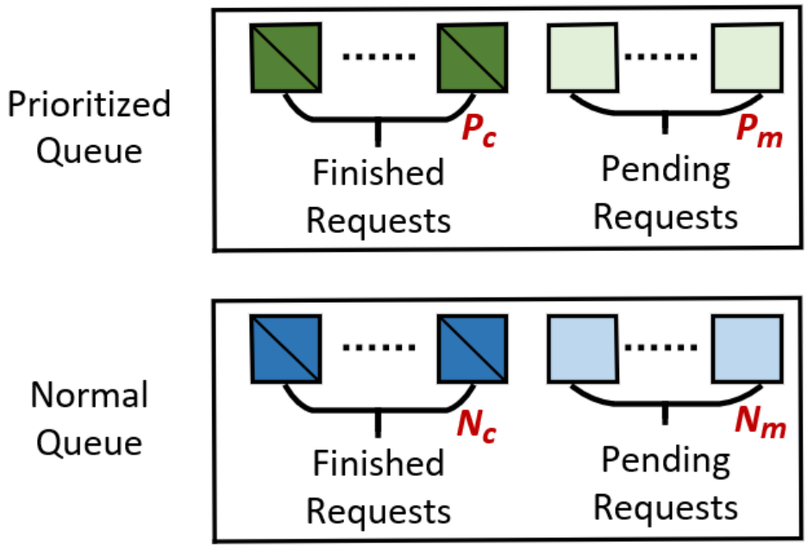
- A 64-bit write lock object
- Lamport’s Bakery algorithm

Visibility Bit (1-bit)	Reserved (1-bit)	Prioritized Counter (15-bit)	Prioritized Max (15-bit)	Normal Counter (16-bit)	Normal Max (16-bit)
		P_c	P_m	N_c	N_m

(a) A 64-bit write lock object

We construct two queues assigned different priorities for each lock in memory nodes.

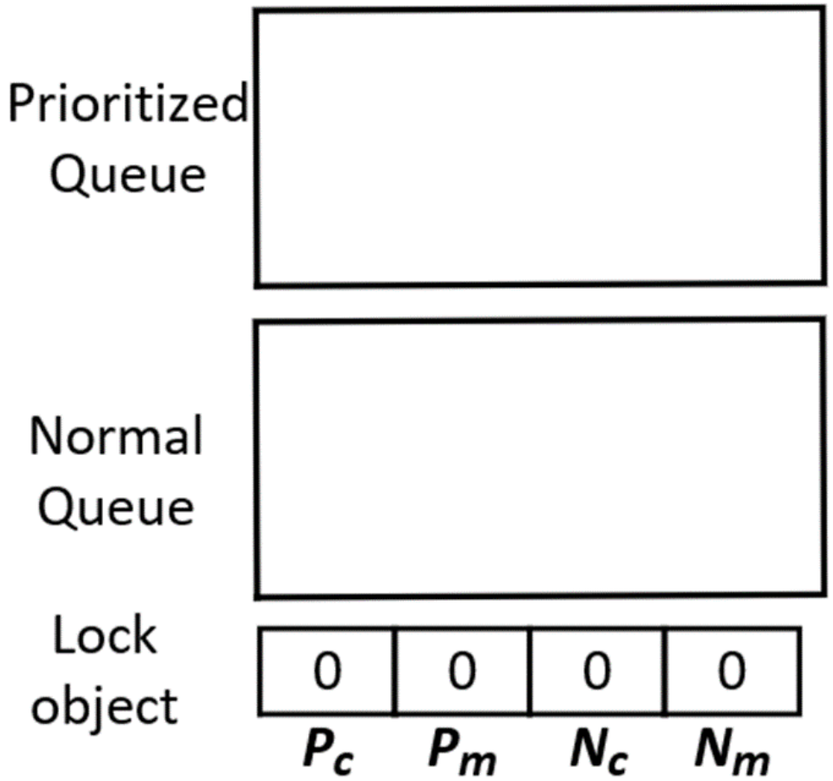
- Requests in the same queue are scheduled in a FIFO manner using Lamport’s Bakery algorithm.
- Coordinators determine the order of requests in different queues



Design

III. Decentralized Priority Locking

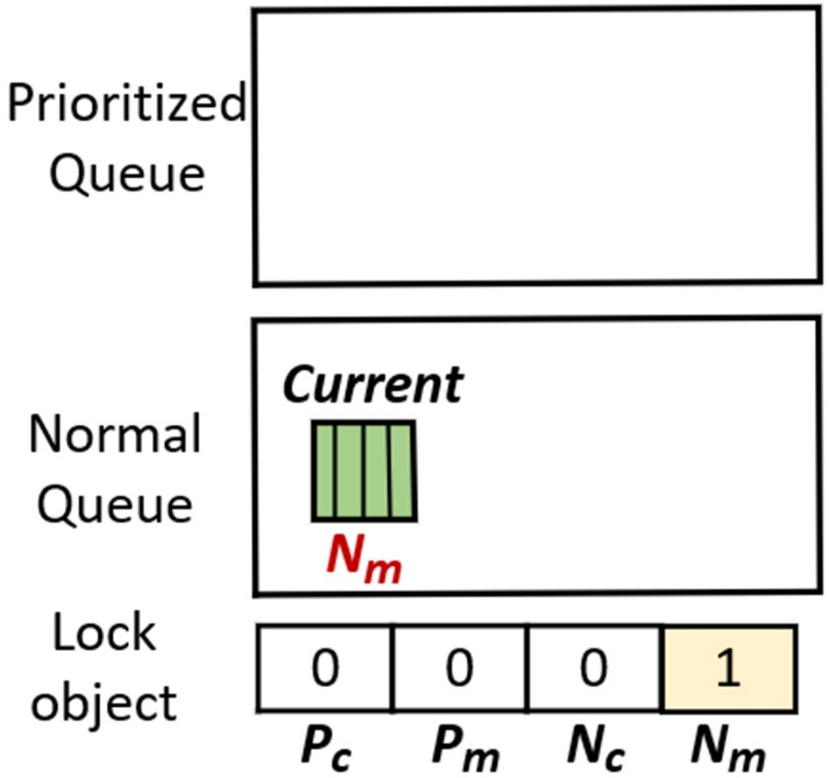
- An example of lock acquisition process



0. After initialization, all segments in a lock object are set to “0”

III. Decentralized Priority Locking

- An example of lock acquisition process

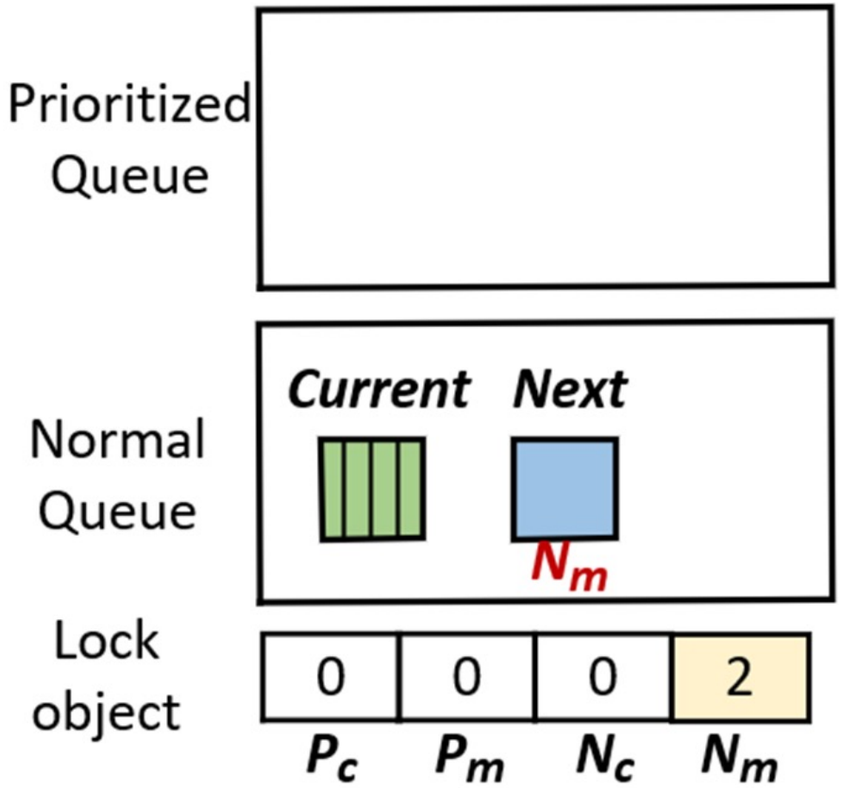


1. When a normal "green" lock request arrives, "Nm" increments by "1".

The "green" request can obtain the permission.

III. Decentralized Priority Locking

- An example of lock acquisition process

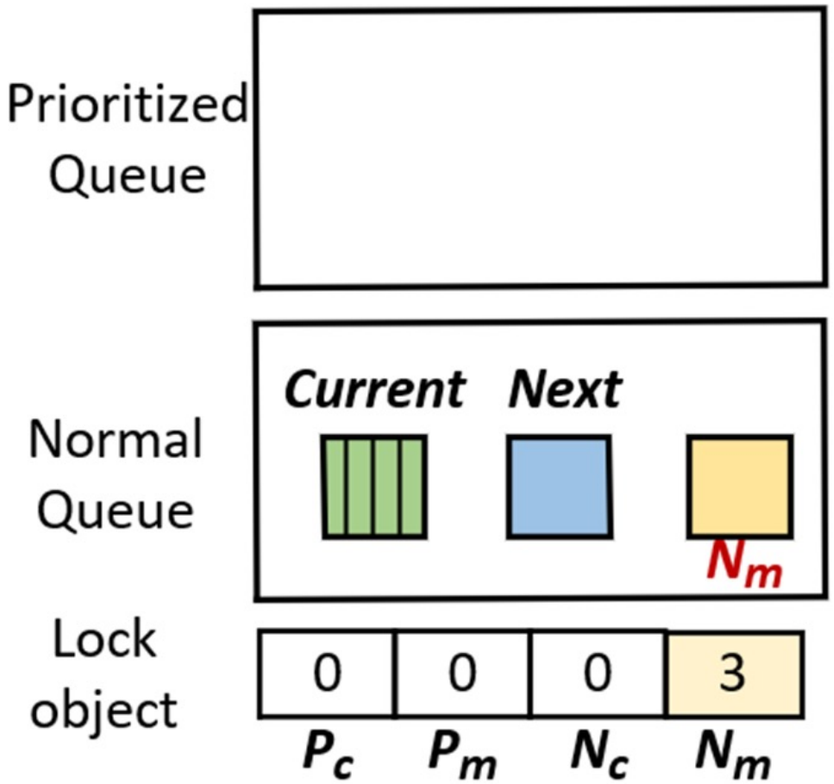


2. When a normal "blue" lock request arrives, "Nm" is incremented to "2".

The "blue" request is the next one to acquire the lock.

III. Decentralized Priority Locking

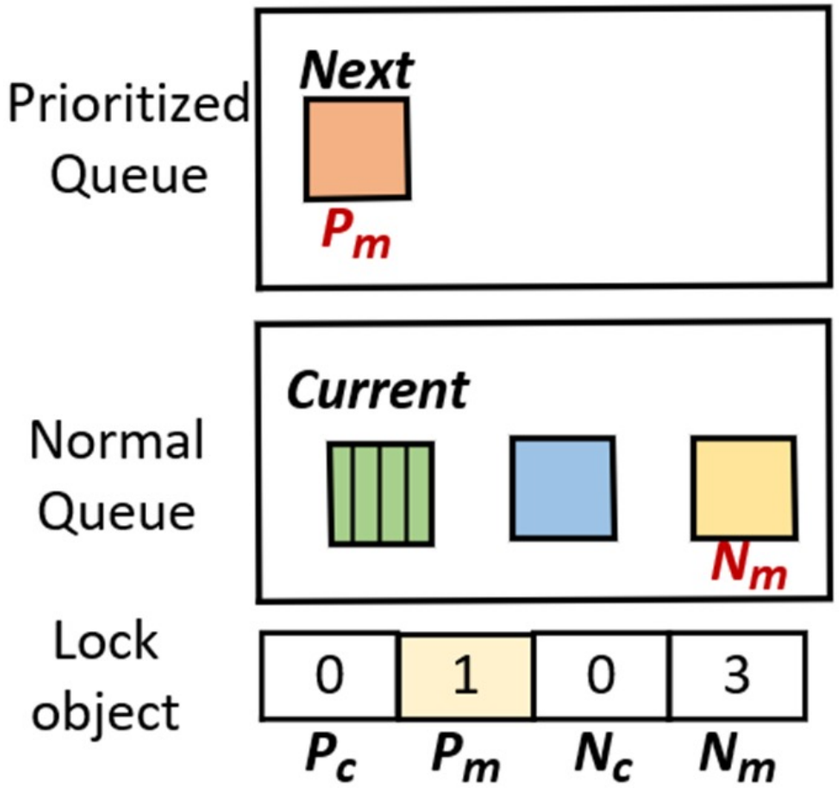
- An example of lock acquisition process



3. When a normal “yellow” lock request arrives, “Nm” is incremented to “3”.

III. Decentralized Priority Locking

- An example of lock acquisition process

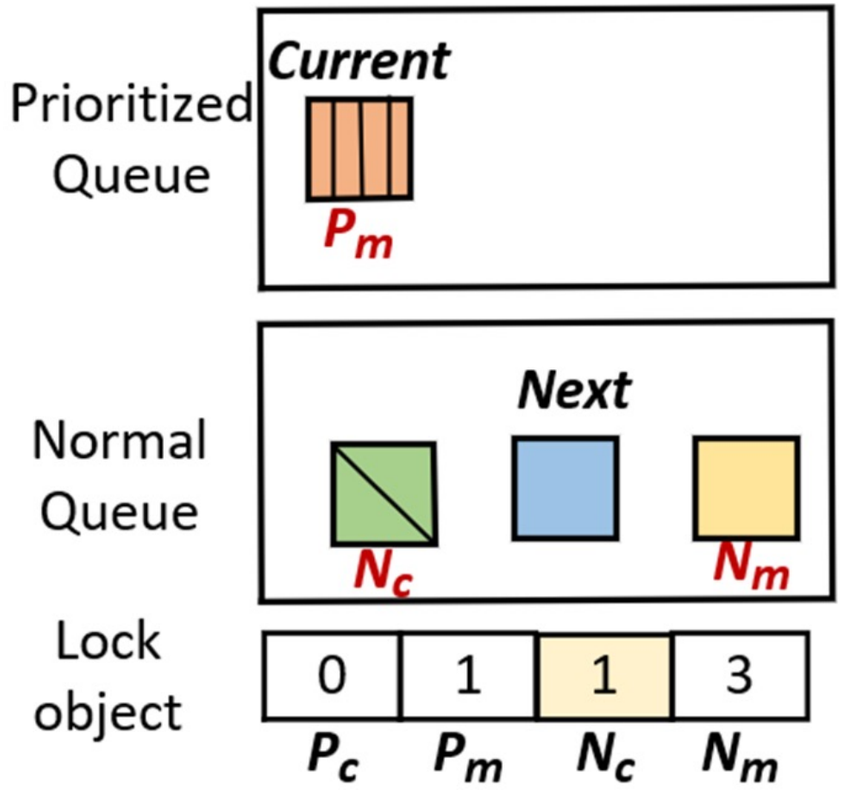


4. When an "orange" lock request with a high priority arrives, "Pm" is incremented to "1".

The "orange" request instead of the "blue" is the next one to acquire the lock.

III. Decentralized Priority Locking

- An example of lock acquisition process



5. After the "green" request completes, "Nc" is incremented to "1".

The "orange" request obtains the permission.

The "blue" request is the next one to acquire the lock.

Evaluation

Setup

Component	Description
CPU	Two-socket Intel Xeon Gold 6230 2.10 GHz 20-core processors
Memory	128 GB DRAM and 1 TB Intel Optane DCPMM
Network	Mellanox ConnectX-3 40/56 GbE network controller
DDIO	OFF
Optane DCPMM	Intel ndctl driver App Direct Mode （chardev）

Compare system：

- FORD: a state-of-the-art transaction system is designed particularly for DM architecture
- DM-compatible FaRM: a derivation of FaRM in which the coordinators use one-sided RDMA to access remote data

We deploy the DM system on an RDMA-capable cluster of 5 servers, configured with two memory nodes for 2-way replication. Each node is limited to a single CPU core for initialization.

Evaluation

Throughput and Latency

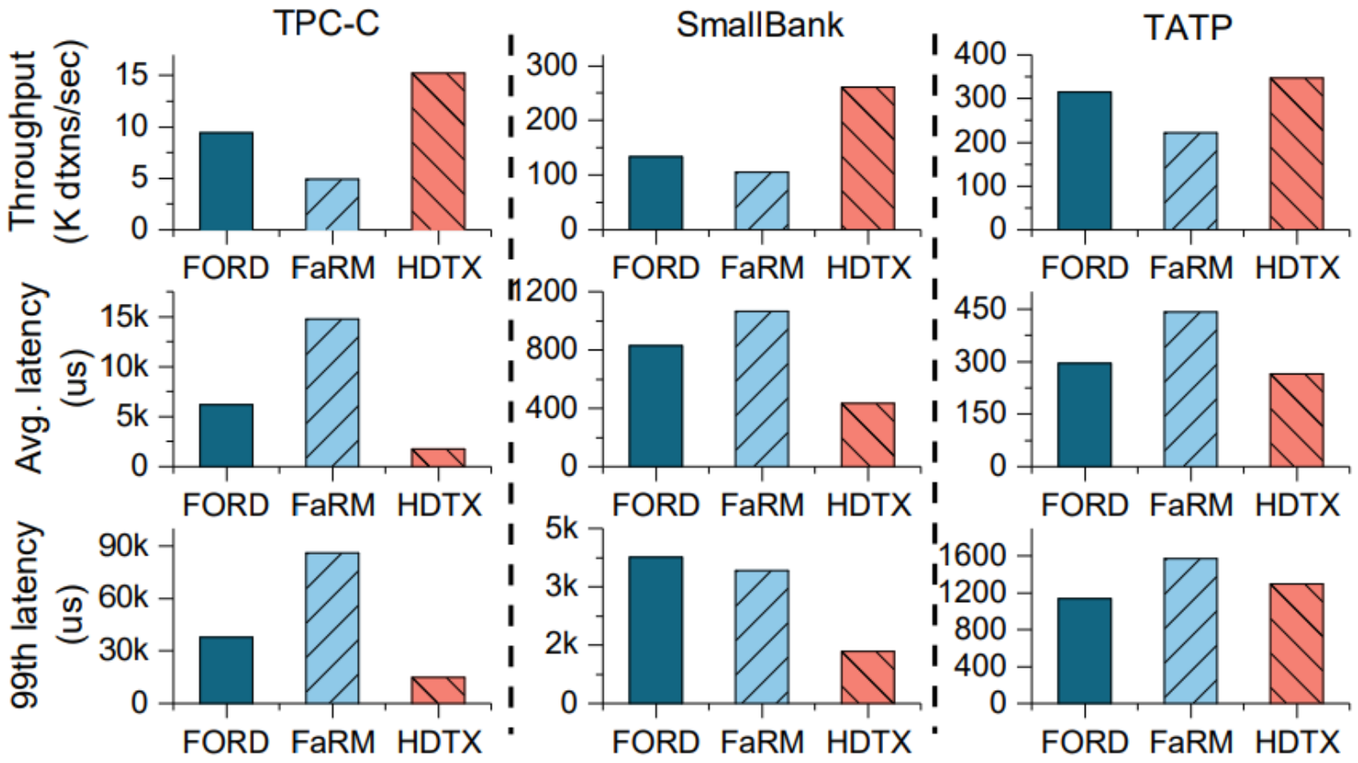


Figure. The end-to-end performance of different workloads

HDTX shows better performance.

For example, in TPC-C:

1. Average latency reduction：
   72.1% and 88.3%
2. Throughput improvement:
   84.7% and 2.08×
3. 99th percentile latency reduction:
   60.9% and 82.7%

Evaluation

The effectiveness of three designs

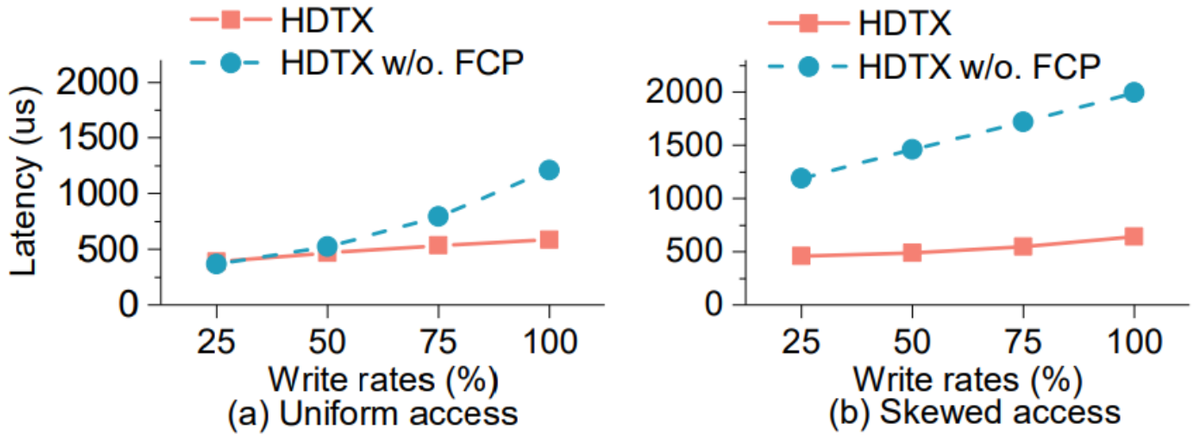


Figure. The effectiveness of fast commit protocol (FCP)

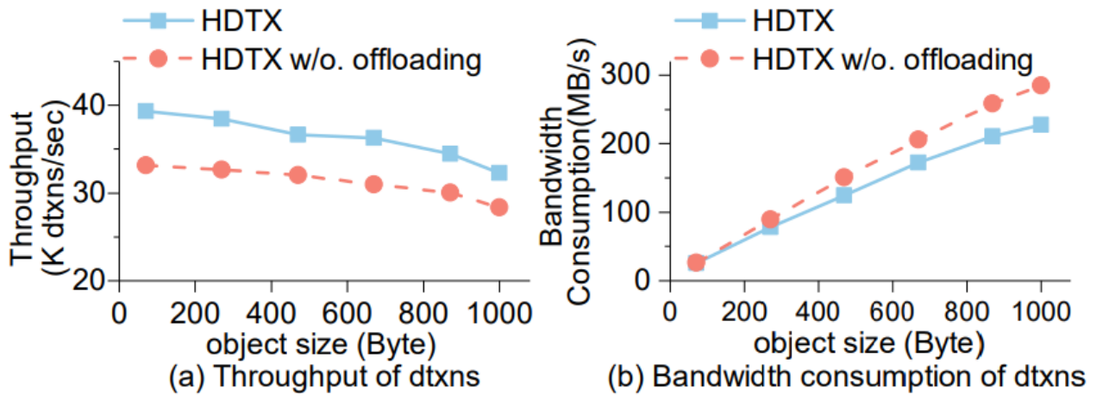


Figure. The effectiveness of release phase offloading

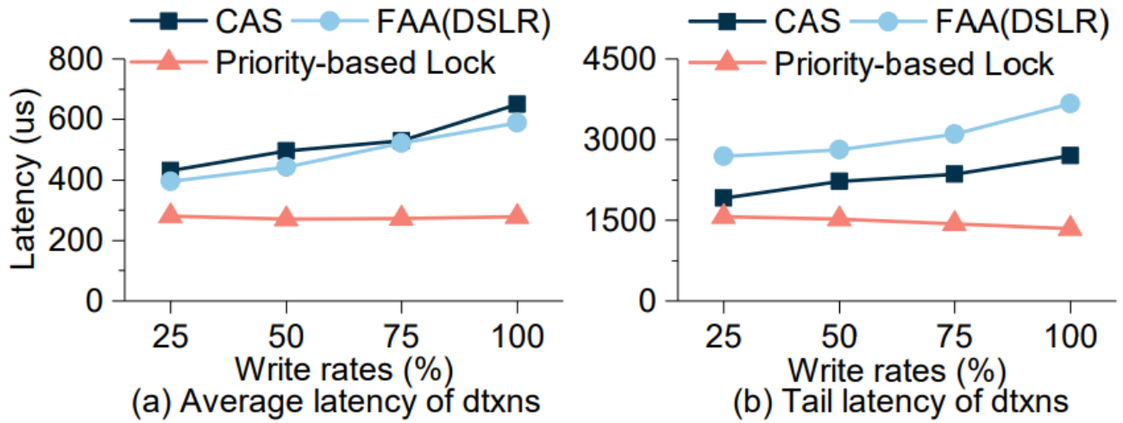


Figure. The effectiveness of priority-based locking

Conclusion

This paper presents HDTX, a high-performance dtxn system for DM. HDTX delivers fast dtxn services and supports flexible dtxn scheduling. Experimental results demonstrate that HDTX provides higher throughput and lower latency than state-of-the-art systems.

*See Section 4.5 of the paper for more details on fault tolerance.