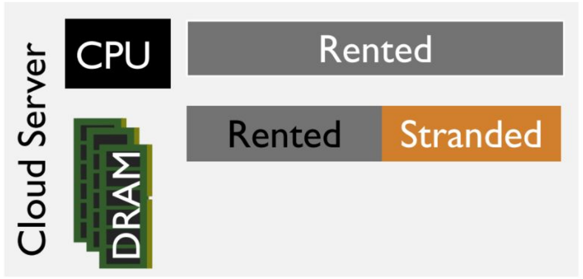
Pasha: An Efficient, Scalable Database Architecture for CXL Pods

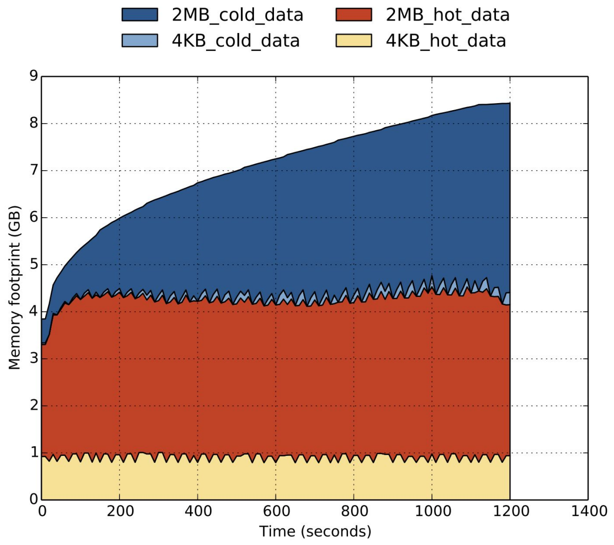
Yibo Huang, Newton Ni
Vijay Chidambaram, Emmett Witchel, Dixin Tang
The University of Texas at Austin

TEXAS
The University of Texas at Austin

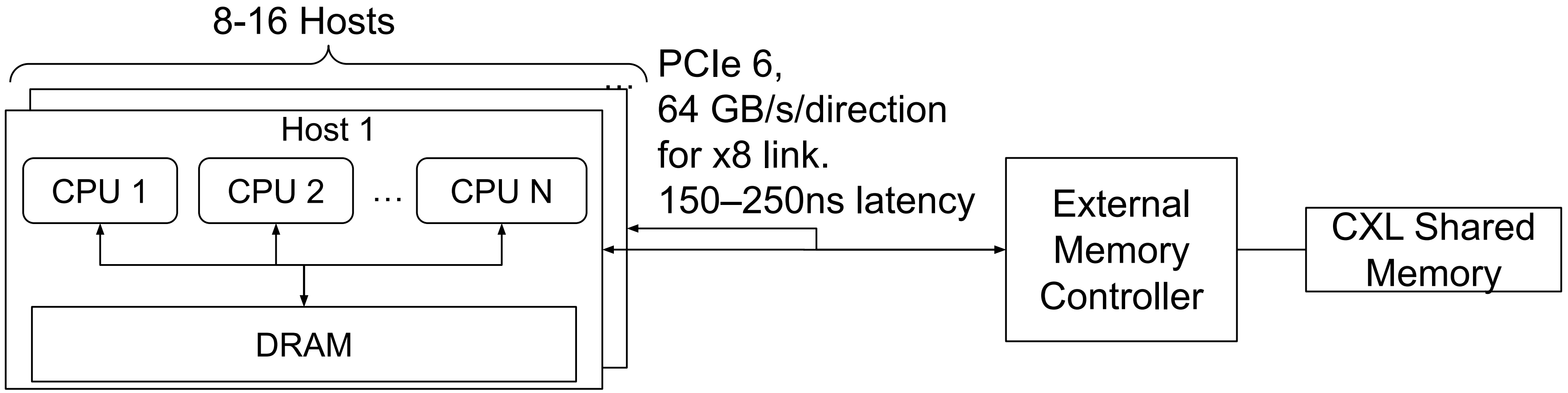
Memory problems in the datacenter

- Applications want more memory
- Azure VMs sell all processors and strand memory [Pond, ASPLOS 23]
    - Up to 25% stranded memory, memory is 40-50% of cost
- Store cold data in slow/cheap memory to save $$ [Thermostat ASPLOS 17]





CXL memory is shared via PCIe



- 8-16 Hosts physically connected to a CXL memory module
  - Module has normal DRAM
  - Local DRAM parallel bus, PCIe is serial bus (↑ latency ↓ bandwidth)

Managing CXL as a tier of memory

- Use system software
  - Transparent to applications
  - Measure hot/cold data
  - Move data to proper tier
- Active area now
  - Pond [ASPLOS 23], TPP [ASPLOS 23], TMTS [ASPLOS 23], Nomad [OSDI 24], Colloid [SOSP24], Linux

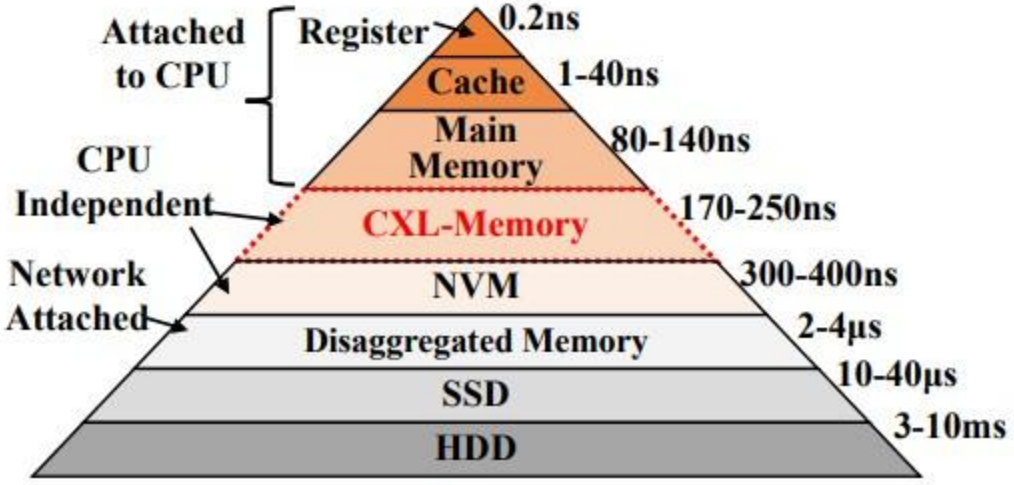
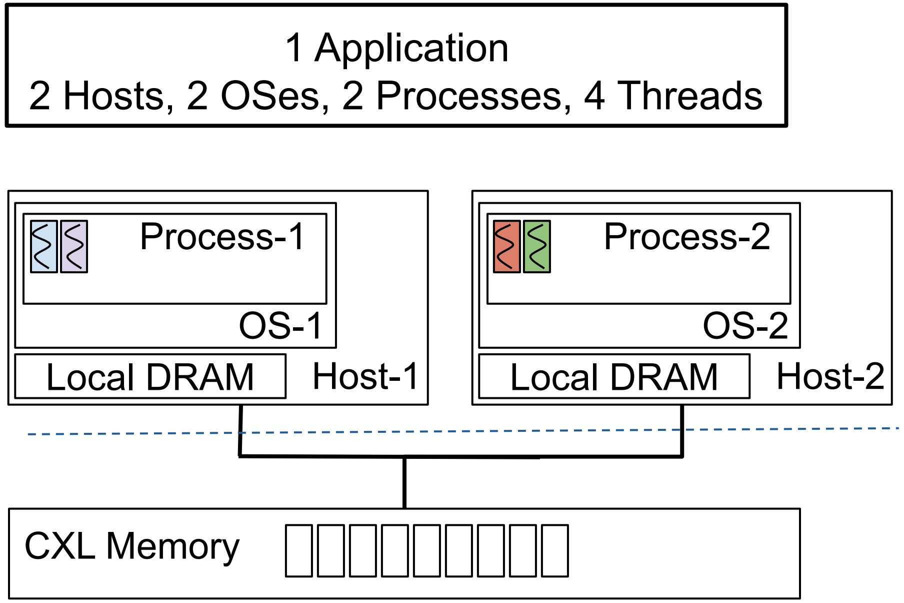


Image credit: TPP, ASPLOS 23

CXL Pod

- Explicit management from
  - Applications, like databases
  - Memory allocator
- Cross-host shared CXL
  - Cache line sharing
  - Requires next HW standard
- 16 hosts X 288 cores
  - 4,608 cores Intel Sierra Forest
  - 7,200 hyperthreads from MapReduce [OSDI 04]

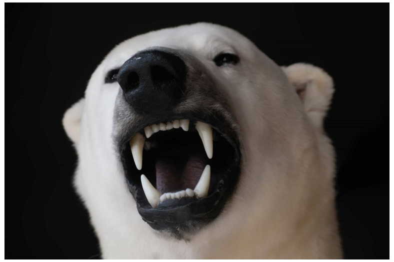


Find the right climate for your software

One Host
- Shared mutable state
- Centralized state
- Many efficient algorithms
- Limited concurrency
- Database

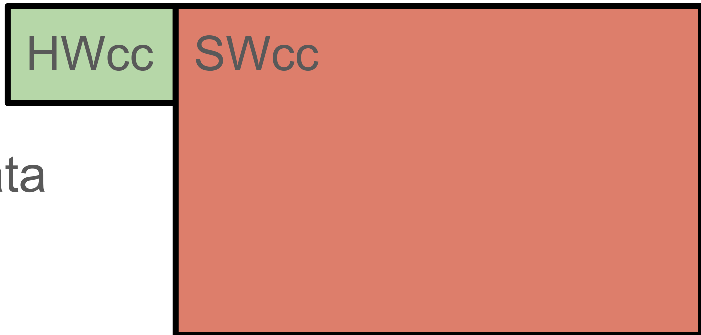
CXL Pod
- Easy port target
- Low tail latency
- The “SQLite” of distributed systems

Distributed (many hosts)
- Replicated state machines
- Scalable
- Fast failover
- Difficult to construct and maintain (performance)
- Key-value store

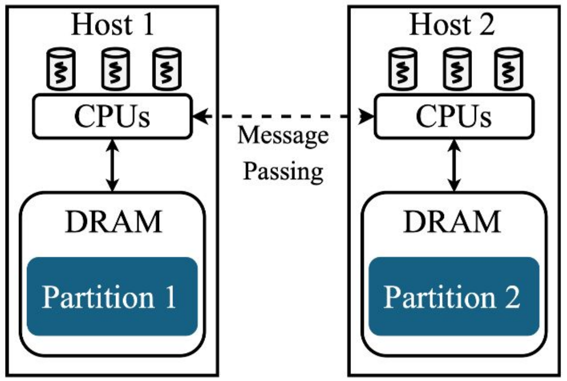


Challenges for CXL Pod

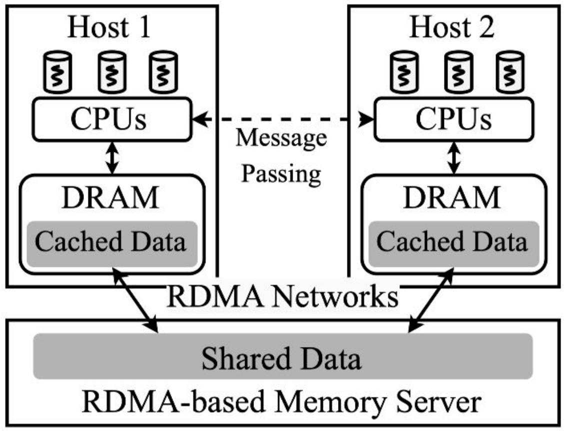
- CXL memory has higher latency than local memory
  - Bad for index structures, pointer chasing
  - ~250ns access time
- CXL memory has lower bandwidth than local memory
  - Bad for large, sequential reads/writes
  - 5-25 GB/s depending on access pattern
- CXL memory has limited (and expensive) support for hardware cache coherence
  - HWcc hundreds of MB, for TB capacity
  - HWcc only for write-shared sync-heavy data



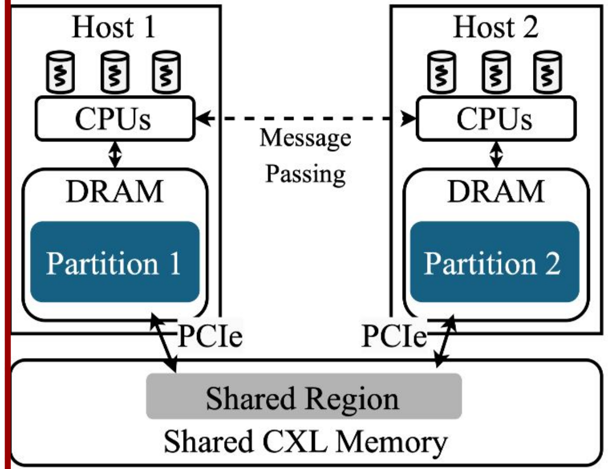
Database organizations



Partition-based
Shared nothing



RDMA-based
Shared memory



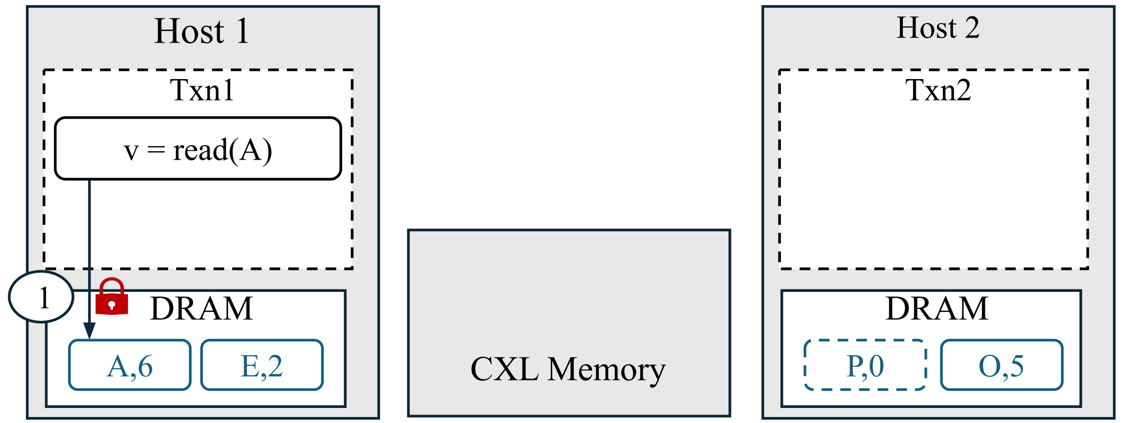
CXL-based
Partitioned & shared

- Real workloads have cross-partition transactions

Pasha in (on) a nutshell

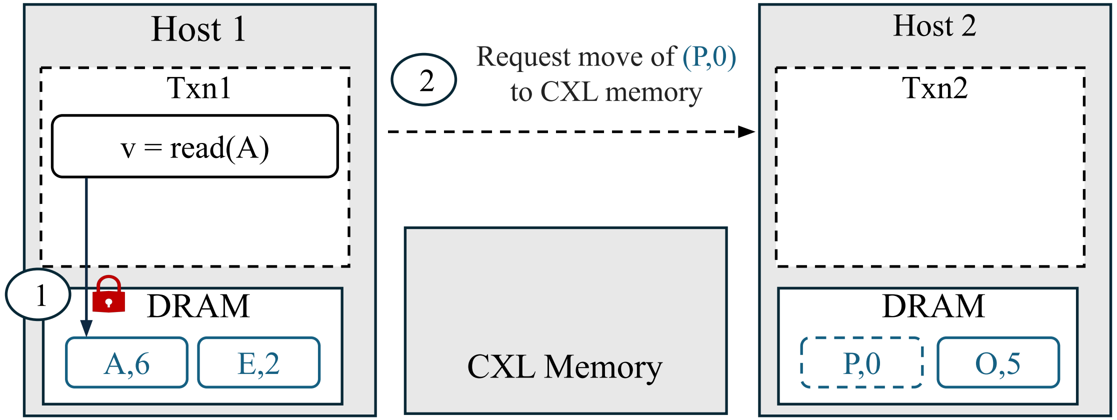
- Synchronize via atomics in CXL memory
  - CXL memory allows processor atomics, unlike RDMA
  - Do not use message passing and two-phase commit
- Keep data in local memory partitions (local DRAM is fast)
- Move shared, cross-partition, active tuples to CXL memory
  - Active data is small (need CXL-aware policies to limit bandwidth)
- Sync-heavy metadata in HWcc, everything else in SWcc
- More challenges/opportunities in paper
  - Partial failures, MVCC, parallel logging, data partitioning, high concurrency

Pasha basics



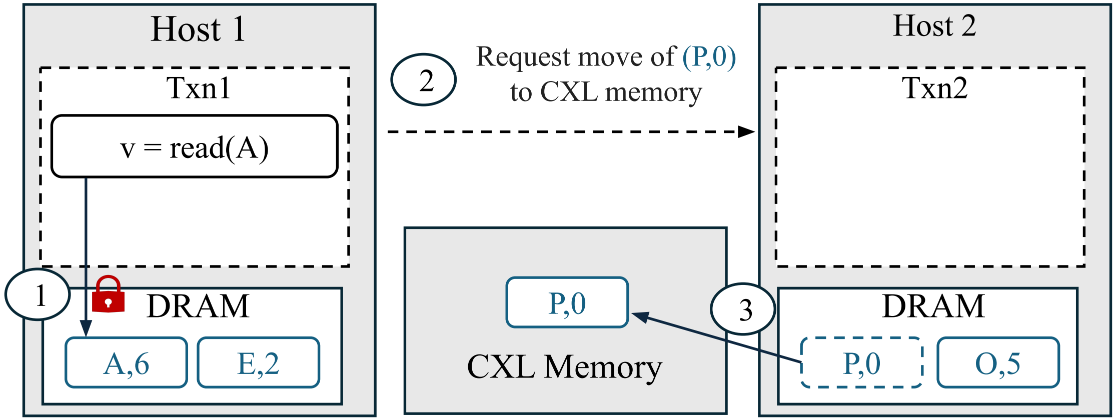
- ① Txn1 locks A and reads it (A=6)
- ② Txn1 message to H2 about P
- ③ H2 moves P to CXL
- ④ Txn1 locks P and writes it (P=9)
- ⑤ Txn2 read of P is denied

Pasha basics



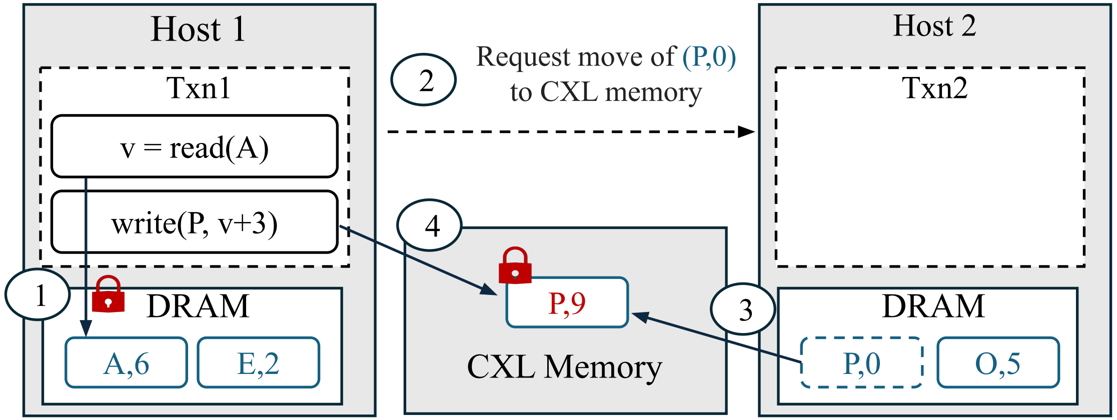
- ① Txn1 locks A and reads it (A=6)
- ② Txn1 message to H2 about P
- ③ H2 moves P to CXL
- ④ Txn1 locks P and writes it (P=9)
- ⑤ Txn2 read of P is denied

Pasha basics



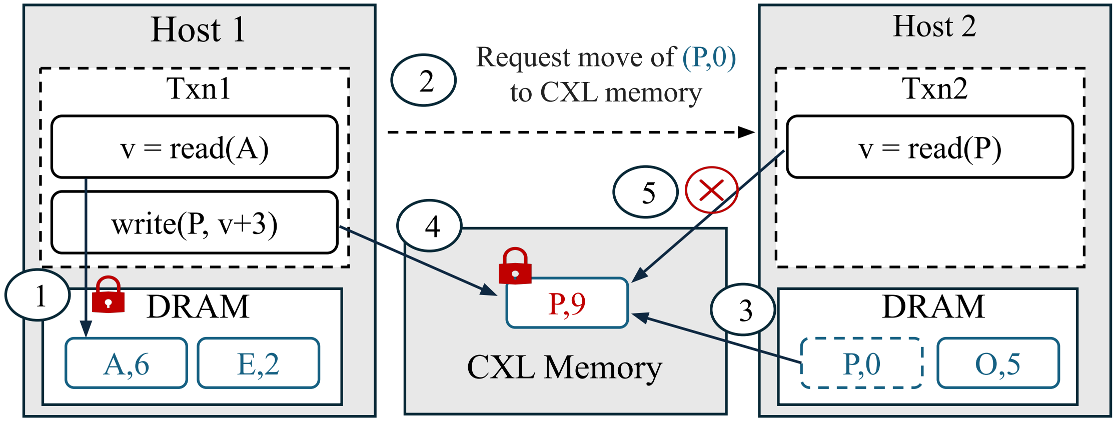
- ① Txn1 locks A and reads it (A=6)
- ② Txn1 message to H2 about P
- ③ H2 moves P to CXL
- ④ Txn1 locks P and writes it (P=9)
- ⑤ Txn2 read of P is denied

Pasha basics



- ① Txn1 locks A and reads it (A=6)
- ② Txn1 message to H2 about P
- ③ H2 moves P to CXL
- ④ Txn1 locks P and writes it (P=9)
- ⑤ Txn2 read of P is denied

Pasha basics



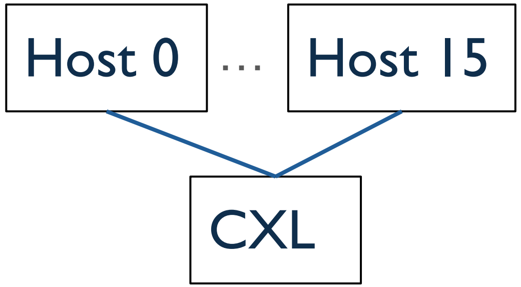
- ① Txn1 locks A and reads it (A=6)
- ② Txn1 message to H2 about P
- ③ H2 moves P to CXL
- ④ Txn1 locks P and writes it (P=9)
- ⑤ Txn2 read of P is denied

Pasha converts a multi-host transaction into a more efficient single-host transaction

CXL hardware assumptions

- Inter-host memory coherence will be expensive
  - 4-6x local coherence cost
  - Difficult to model with multiple VMs with single CXL device
- Global persistent flush (GPF)
  - On a power failure, processor has energy to write back dirty cache lines to CXL memory
  - After store fence, data is “committed”
  - Crucial for performance
- Build the software to guide the hardware

Challenges of the CXL pod - partial failure



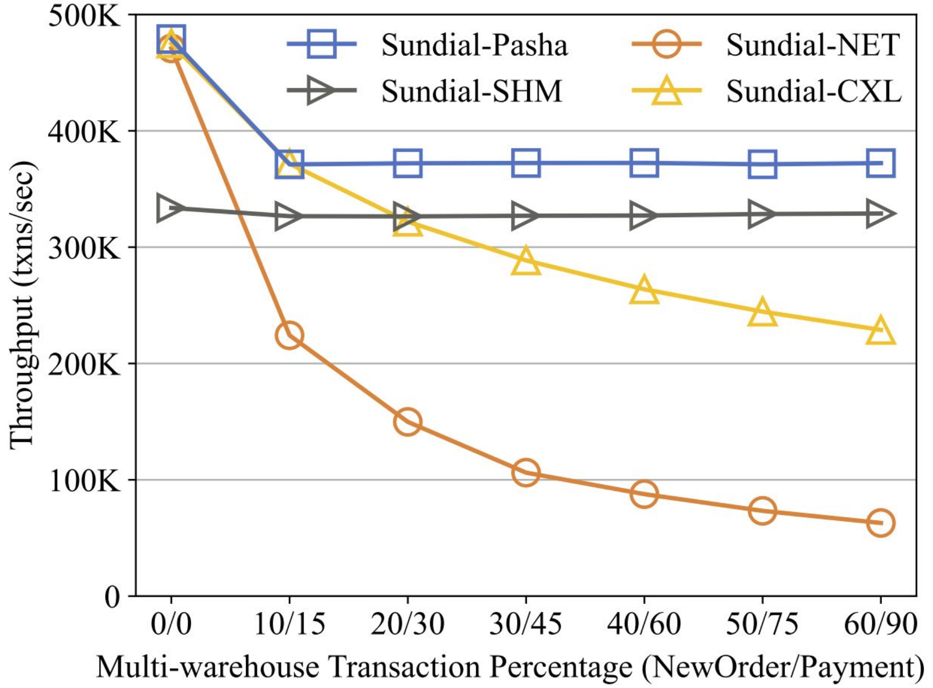
- Let’s say one process dies
  - Do I have to restart all processes?
  - Full restart is bad for availability
    - TPC-C does 590 allocations/ms/core
    - 1-15ms for restart (and ~10ms for recovery)
    - 32-core machine would delay 18,290-274,350 allocations
- Tolerating partial failure means
  - Process recovers and rejoins
  - Application remains available during partial recovery
  - Requires non-blocking data structures or lock ownership + logs [Lupin DIMES 24]

Evaluation

- 8 VMs with 4 vCPUs and 8 GB local DRAM
  - CPU (Intel SPR): 2× Intel® Xeon 8460H CPUs @2.2 GHz
  - RAM: 8× DDR5-4800 channels on each socket (16 in total)
  - 1× DDR5-4800 CXL memory with PCIe 5.0 ×8, CXL 1.1
  - NVMe SSD
- No cross-host CXL as it does not yet exist
  - Single machine coherence stand in inter-machine
- Sundial [VLDB 18]
  - Partition-based distributed database
  - Optimistic reads
  - Pessimistic (two-phase locking) writes

NewOrder+Payment from TPC-C; ↑ % cross-warehouse

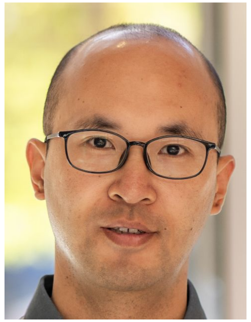
- NET - network message
- CXL - CXL message queue
- SHM - all tables in CXL
- Speedup at 60/90
  - 5.9× Sundial-NET
  - 1.6× Sundial-CXL
  - 1.1× Sundial-SHM
  - (1.4x at 0/0)



Many thanks

Yibo Huang

Newton Ni


Dixin Tang

Vijay Chidambaram