An Evaluation of Distributed Concurrency Control

Harding, Aken, Pavlo and Stonebraker

Presented by: Thamir Qadah

For CS590-BDS

Outline

- Motivation
- System Architecture
- Implemented Distributed CC protocols
  - 2PL
  - TO
  - OCC
  - Deterministic
- Commitment Protocol
  - 2PC
  - Why CALVIN does not need 2PC
    - What is the tradeoff
- Evaluation environment
  - Workload Specs
  - Hardware Specs
- Discussion
  - Bottlenecks
  - Potentiual soluutions

Motivation

- Concerned with:
    - When does distributing concurrency control benefit performance?
    - When is distribution strictly worse for a given workload?
- Costs of distributed transaction processing are well known [Bernstein et. al ‘87, Ozsu and Valduriez ‘11]
    - But, in cloud environments providing high scalability and elasticity, trade-offs are less understood.
- With new proposals of distributed concurrency control protocols, there is no comprehensive performance evaluation.

Publication	Experimental Comparisons Performed Lock	TS	MV	OCC	Det	None
Tango [7]	✓					
Spanner [20]						✗
Granola [21]	✓					
Centiman [25]						✗
FaRM [26]						✗
Warp [27]						✗
MaaT [39]	✓					
Rococo [41]	✓			✓		
Ren et al. [45]	✓			✓		
F1 [47]						✗
Calvin [54]						✗
Wei et al. [58]					✓	
TaPiR [61]	✓			✓		
Lynx [62]						✗
Deneva (this study)	✓×2	✓	✓	✓	✓	

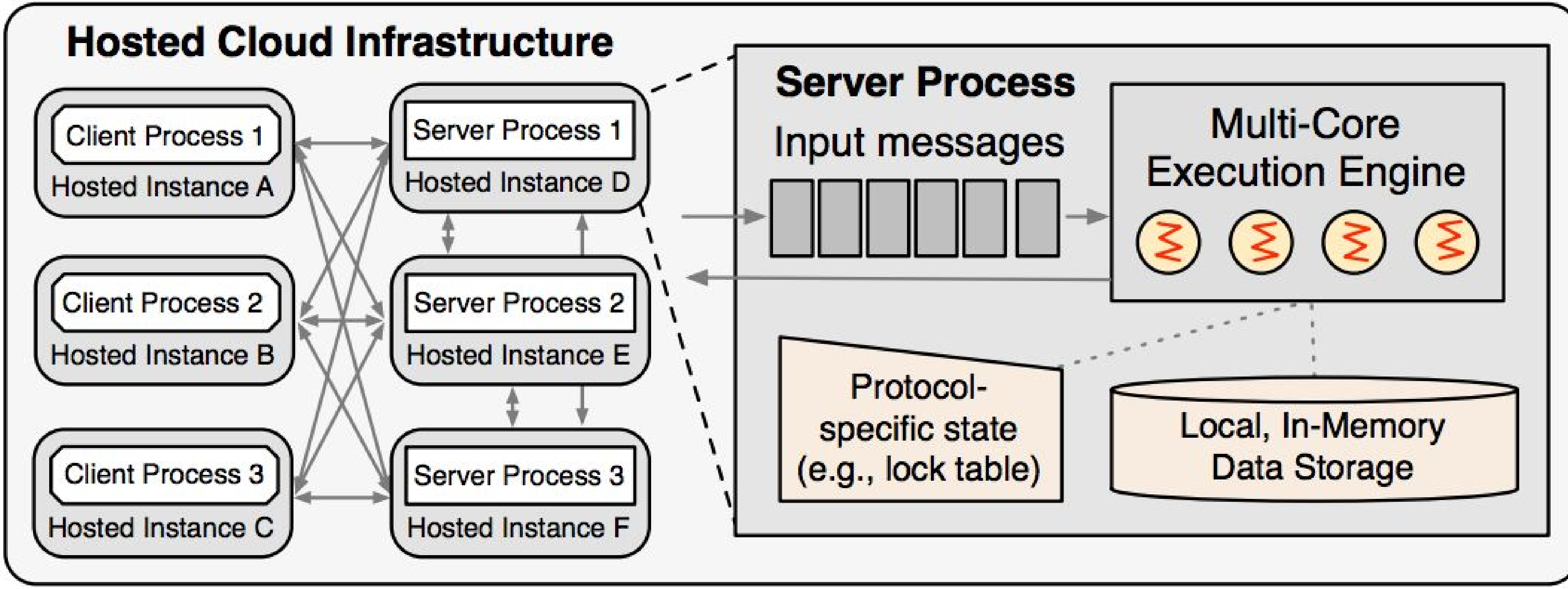
Publication	Experimental Comparisons Performed Lock	TS	MV	OCC	Det	None
Tango [7]	✓					
Spanner [20]						✗
Granola [21]	✓					
Centiman [25]						✗
FaRM [26]						✗
Warp [27]						✗
MaaT [39]	✓					
Rococo [41]	✓					
Ren et al. [45]	✓			✓		
F1 [47]						✗
Calvin [54]						✗
Wei et al. [58]					✓	
TaPiR [61]	✓			✓		
Lynx [62]						✗
Deneva (this study)	✓×2	✓	✓	✓	✓	

Note: Lock-based implementations may be different (e.g. deadlock detection/avoidance)

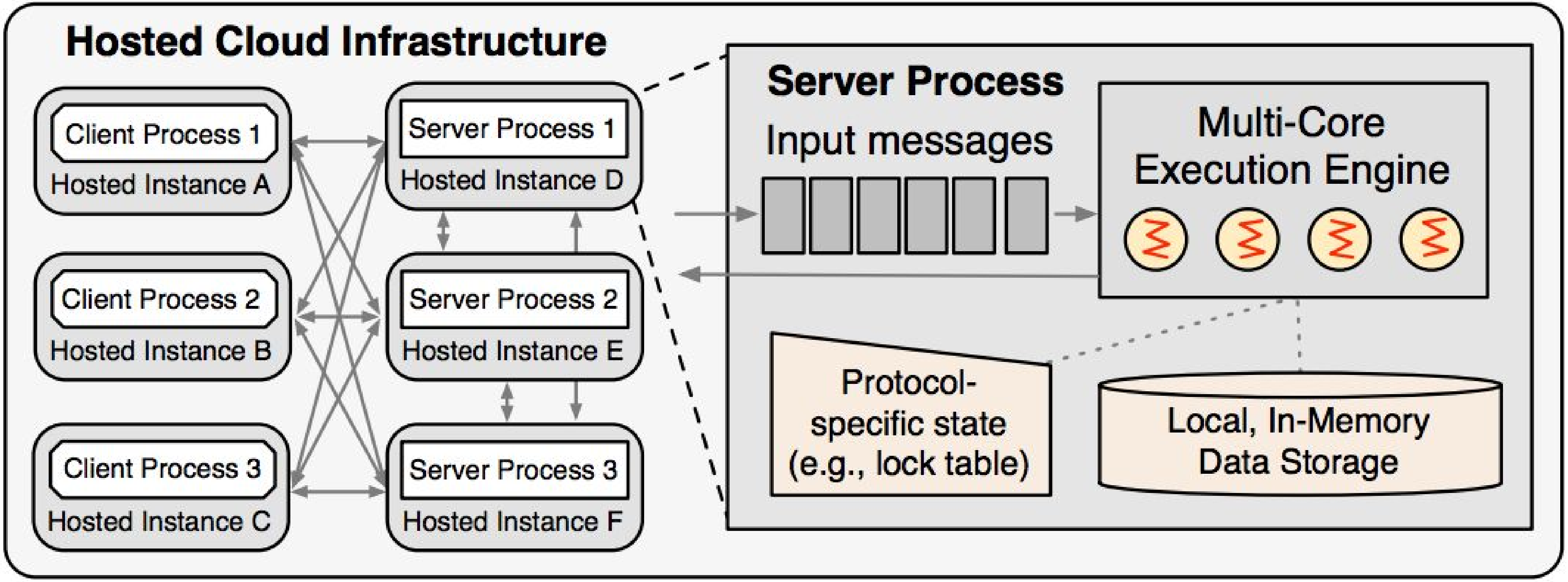
Transaction Model

- Deneva uses the concept of stored procedures to model transactions.
  - No client stalls in-between transaction logical steps
- Support for protocols (e.g. CALVIN) that require READ-SET and WRITE-SET to be known in-advanced
  - DBMS needs to compute that.
    - Simplest way: run transaction without any CC measures

High Level System Architecture

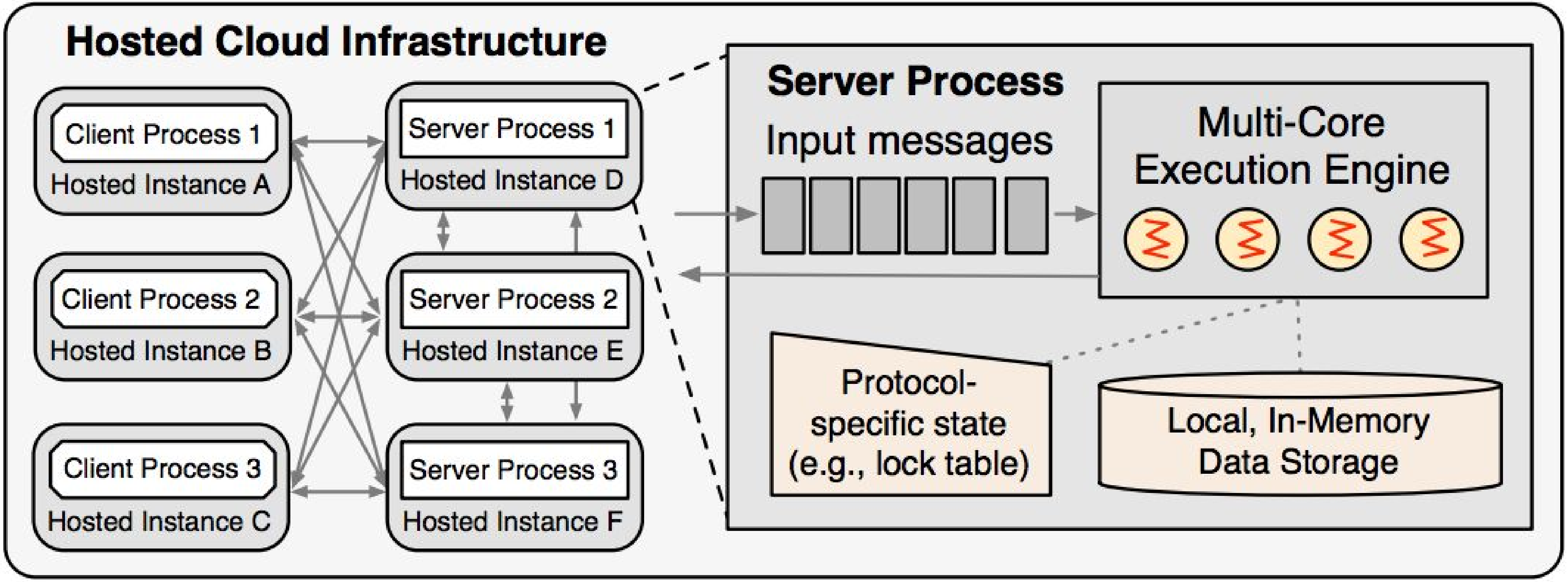


High Level System Architecture

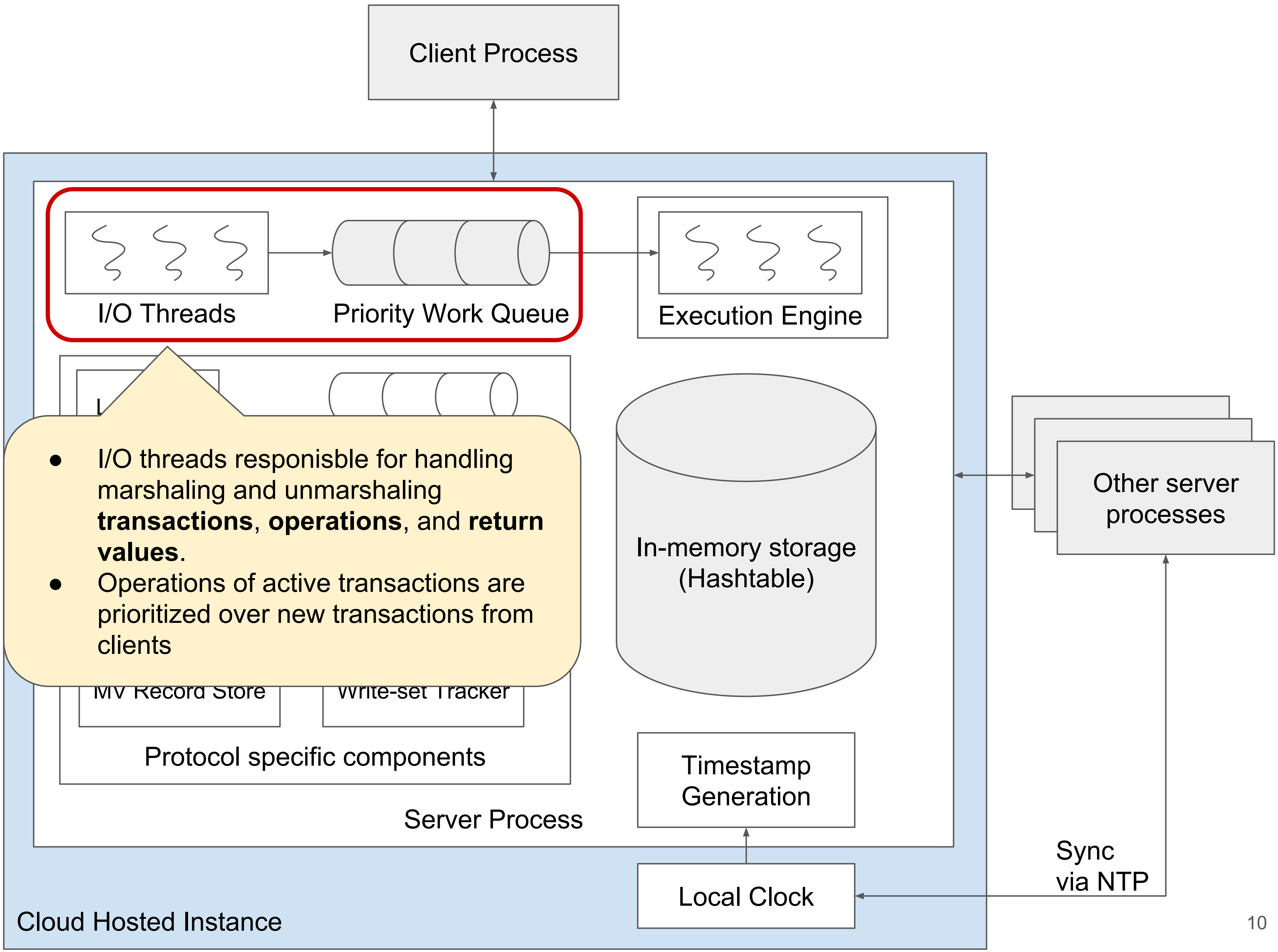


Client and Server processes are deployed on different hosted cloud instance

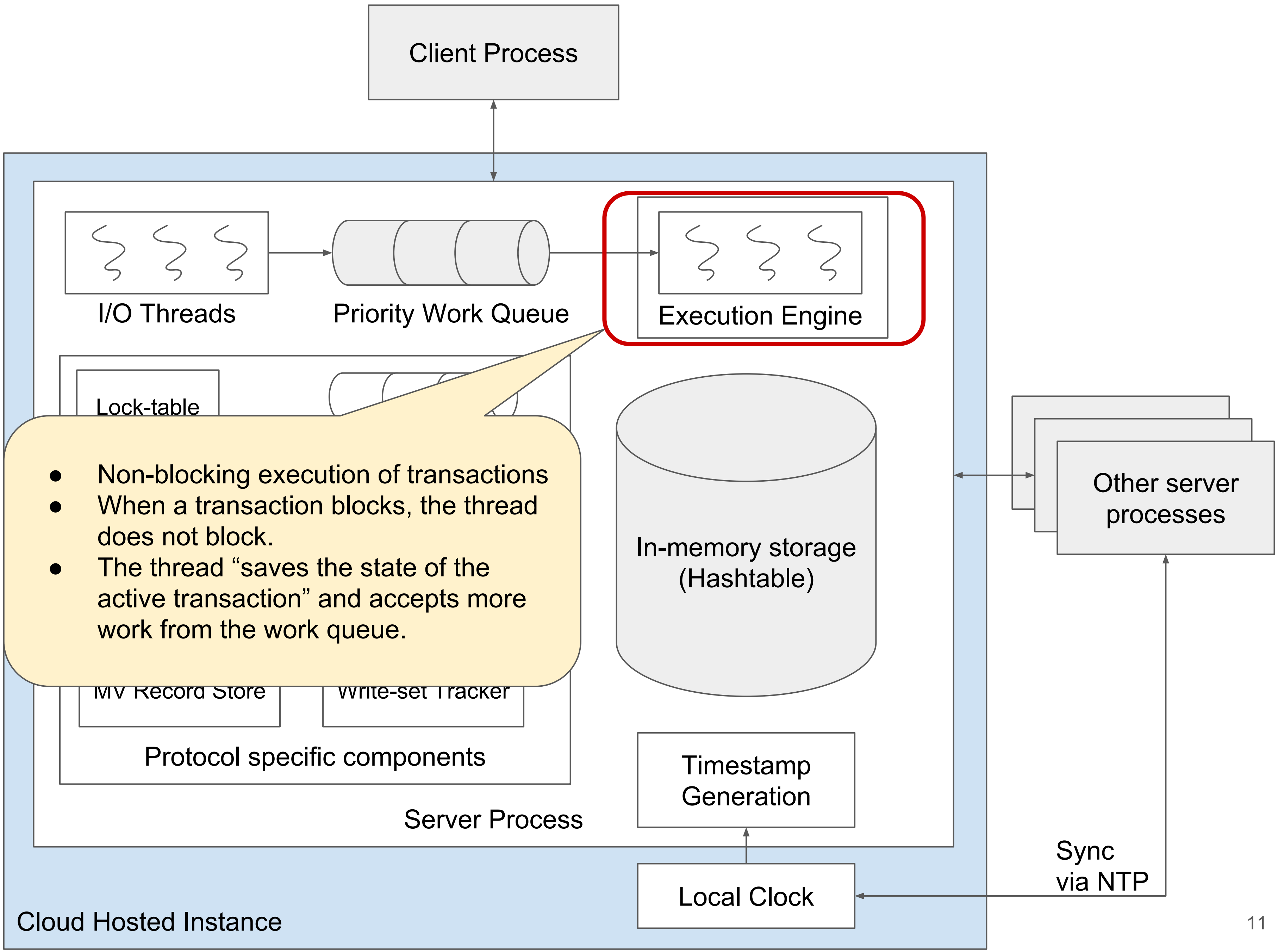
High Level System Architecture

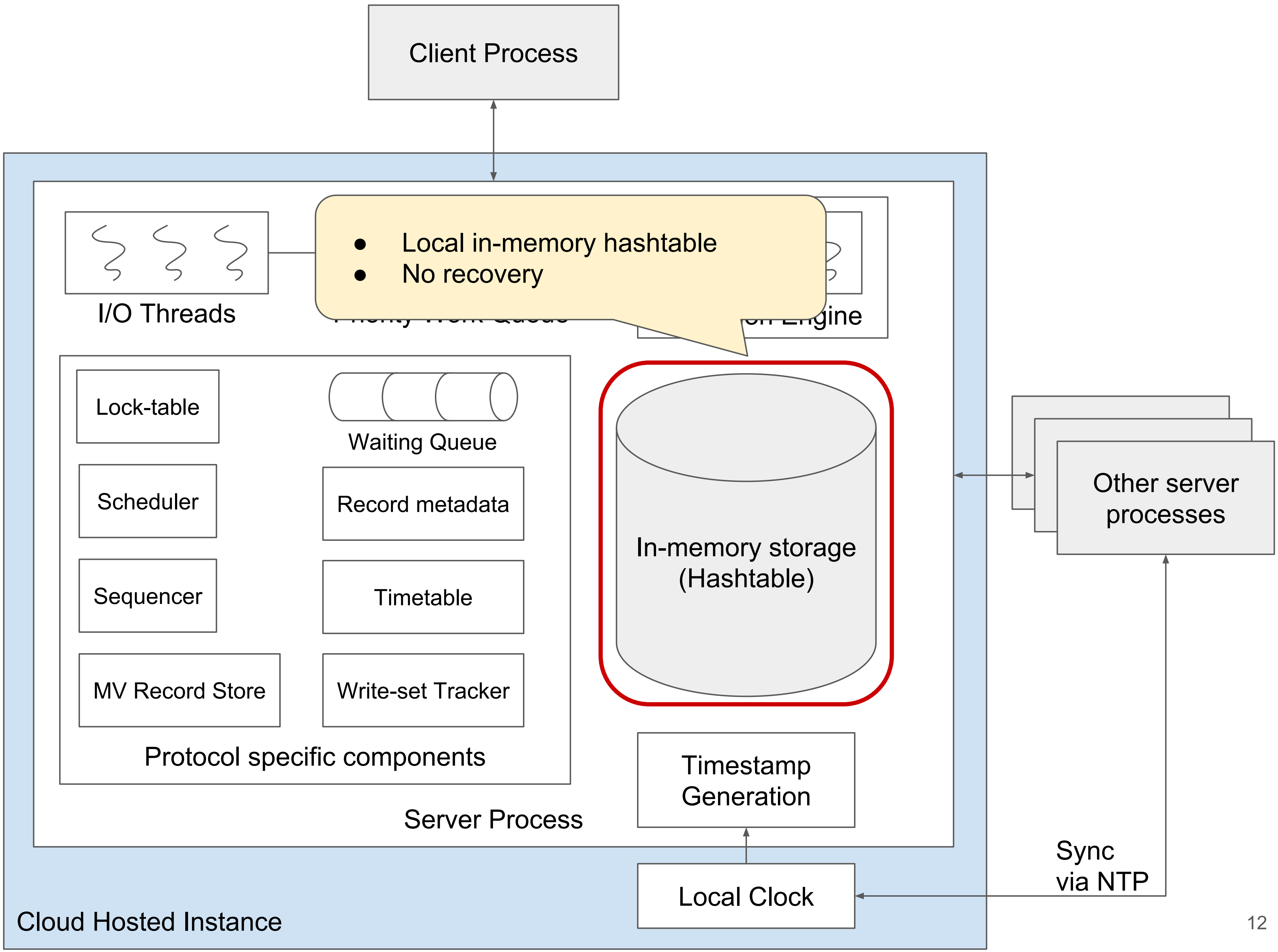


Communication among processes uses nanomsg socket library

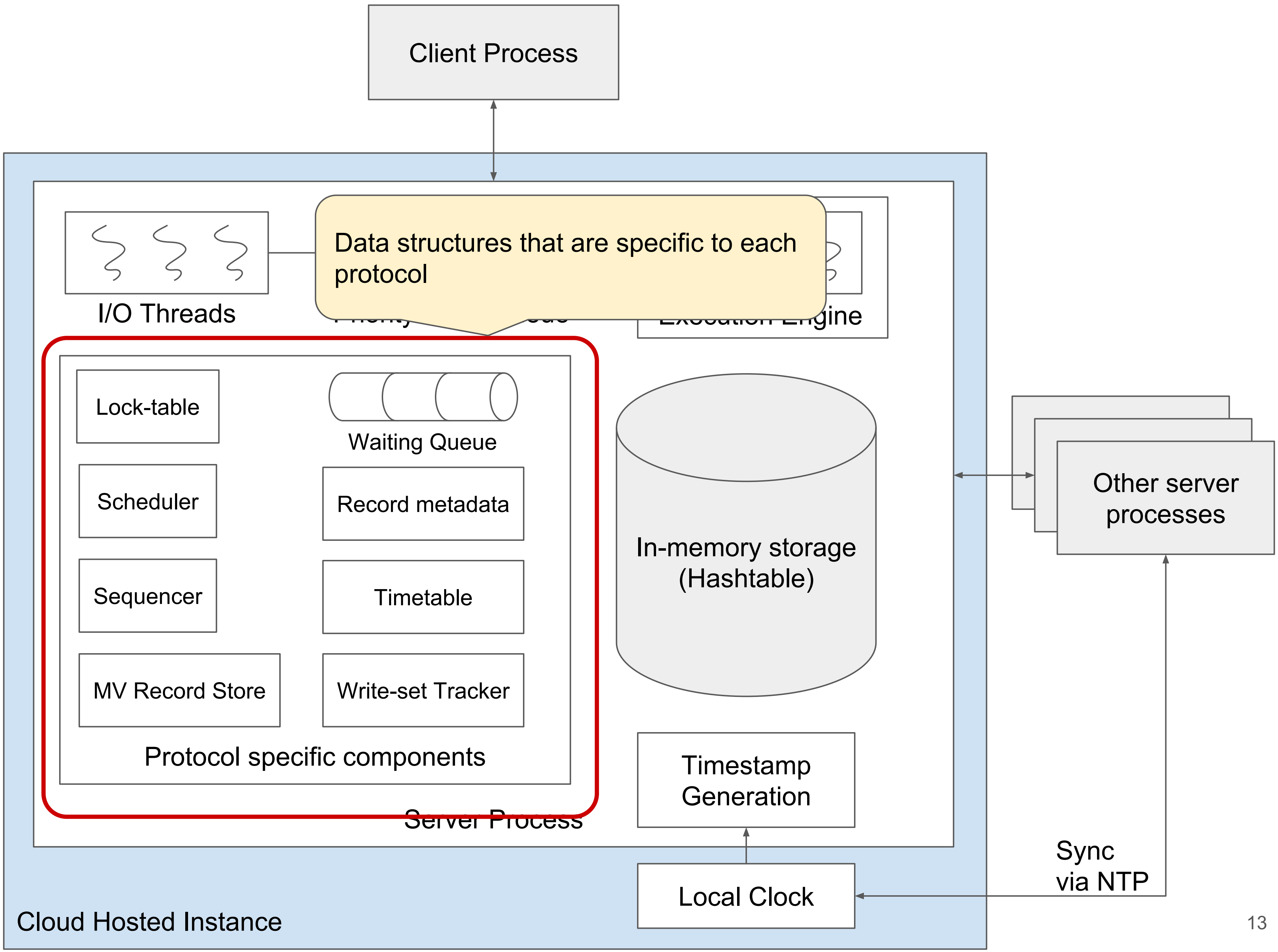


Client Process
I/O Threads
Priority Work Queue
Execution Engine
I/O threads responisble for handling marshaling and unmarshaling **transactions**, **operations**, and **return values**.
Operations of active transactions are prioritized over new transactions from clients
MV Record Store
Write-set Tracker
Protocol specific components
In-memory storage (Hashtable)
Other server processes
Timestamp Generation
Server Process
Sync via NTP
Local Clock
Cloud Hosted Instance

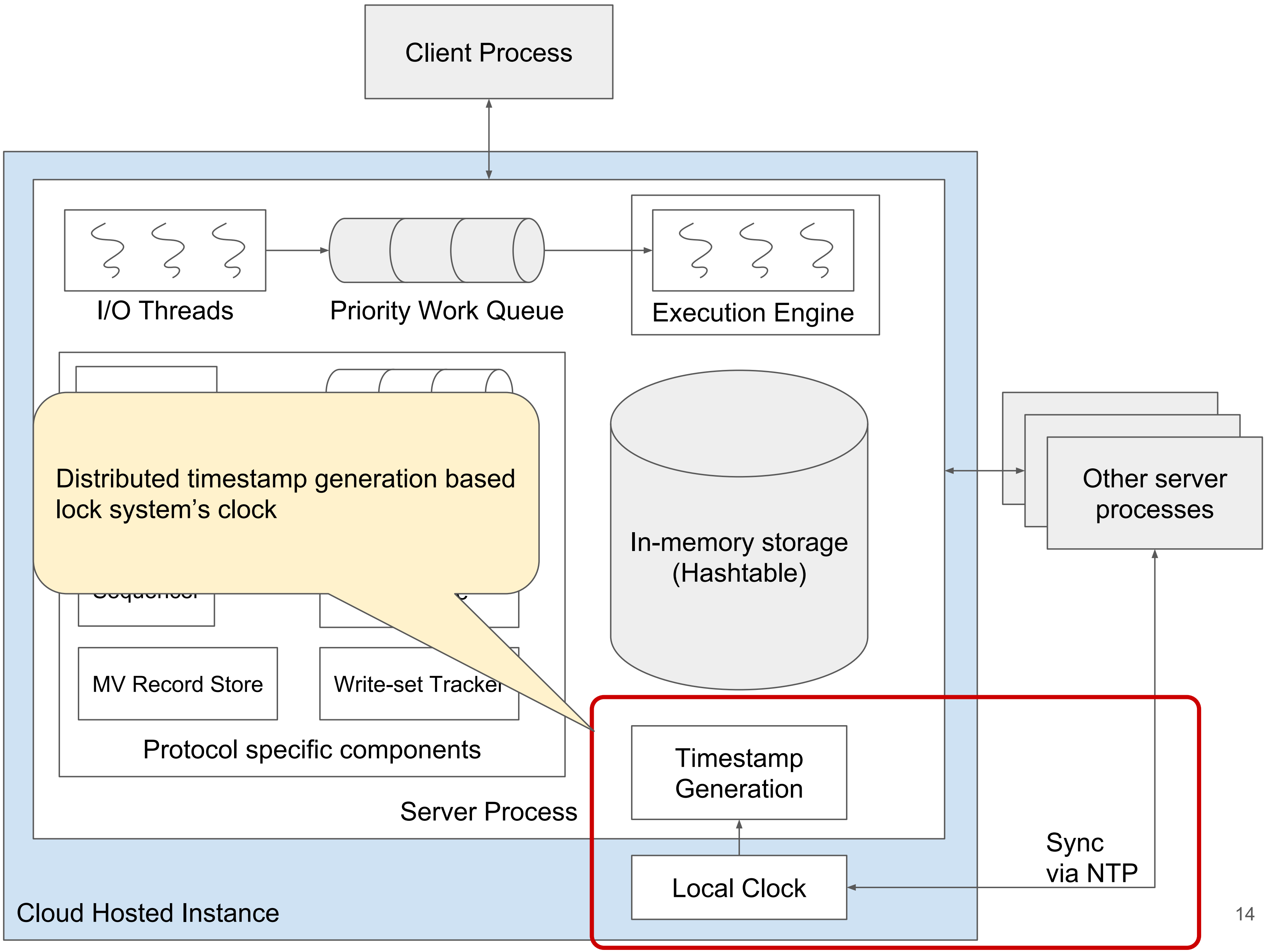

Client Process
I/O Threads
Priority Work Queue
Execution Engine
Lock-table
Non-blocking execution of transactions
When a transaction blocks, the thread does not block.
The thread “saves the state of the active transaction” and accepts more work from the work queue.
MV Record Store
Write-set Tracker
Protocol specific components
In-memory storage (Hashtable)
Other server processes
Timestamp Generation
Server Process
Local Clock
Sync via NTP
Cloud Hosted Instance



Client Process
Local in-memory hashtable
No recovery
I/O Threads
Lock-table
Waiting Queue
Scheduler
Record metadata
Sequencer
Timetable
MV Record Store
Write-set Tracker
Protocol specific components
In-memory storage (Hashtable)
Timestamp Generation
Server Process
Other server processes
Sync via NTP
Local Clock
Cloud Hosted Instance



Client Process
Data structures that are specific to each protocol
I/O Threads
Lock-table
Waiting Queue
Scheduler
Record metadata
Sequencer
Timetable
MV Record Store
Write-set Tracker
Protocol specific components
Server Process
In-memory storage (Hashtable)
Other server processes
Timestamp Generation
Local Clock
Sync via NTP
Cloud Hosted Instance


Client Process
I/O Threads
Priority Work Queue
Execution Engine
Distributed timestamp generation based lock system's clock
In-memory storage (Hashtable)
Other server processes
MV Record Store
Write-set Tracker
Protocol specific components
Timestamp Generation
Server Process
Local Clock
Sync via NTP
Cloud Hosted Instance

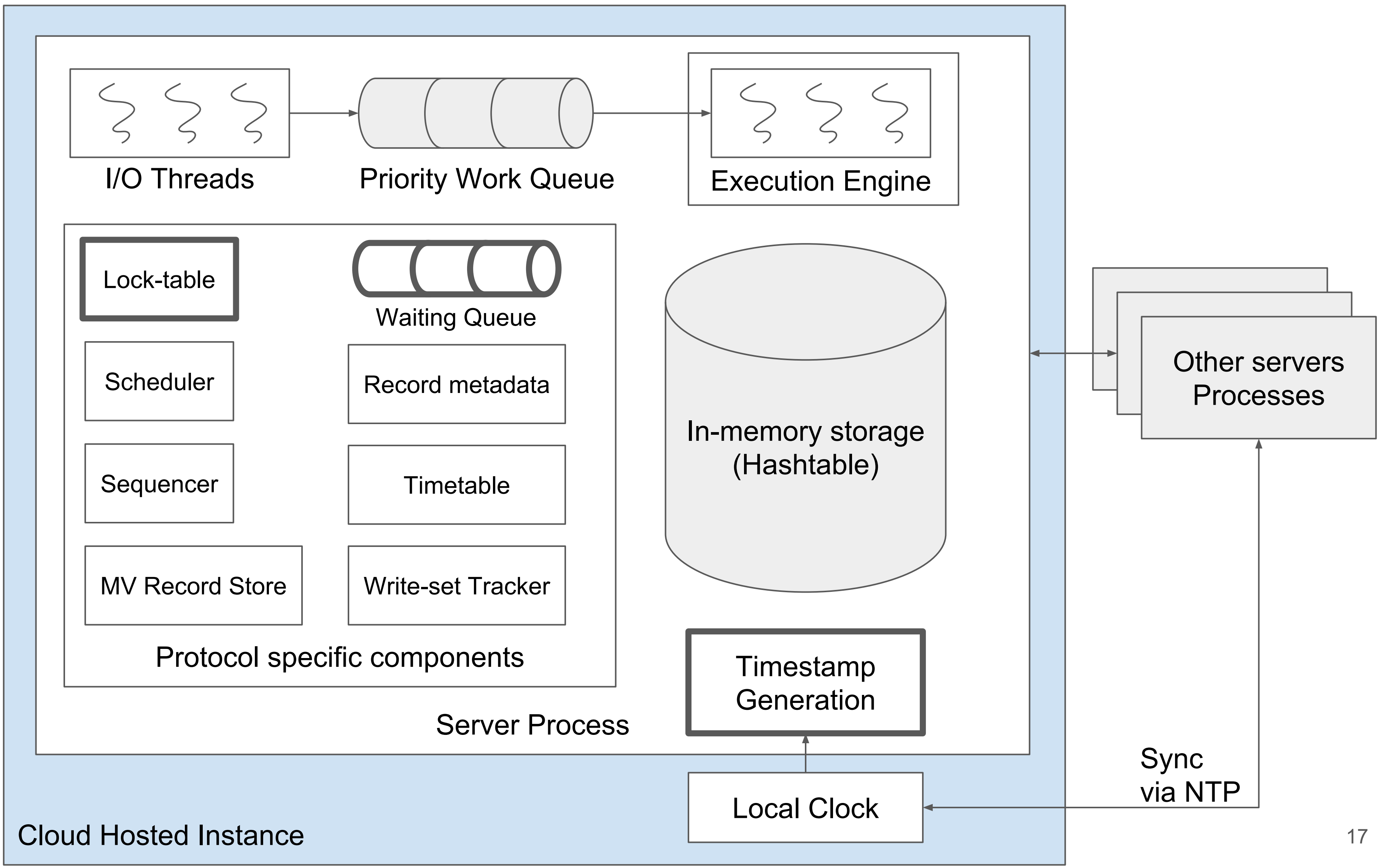
Transaction Protocols

- Concurrency Control
  - Two-phase Locking (2PL)
    - NO_WAIT
    - WAIT_DIE
  - Timestamp Ordering (TIMESTAMP)
  - Multi-version concurrency control (MVCC)
  - Optimistic concurrency control (OCC)
  - Deterministic (CALVIN)
- Commitment Protocols
  - Two-phase Commit (2PC)

Two-phase Locking (2PL)

- Two phase:
  - Growing phase: lock acquisition (no lock release)
  - Shrink phase: lock release (no more acquisition)
- NO_WAIT
  - Aborts and restarts the transaction if lock is not available
  - No deadlocks (suffers from excessive aborts)
- WAIT_DIE
  - Utilizes timestamp
  - Older transactions wait, younger transactions abort
  - Locking in shared mode bypasses lock queue (which contains waiting writers)

2PL

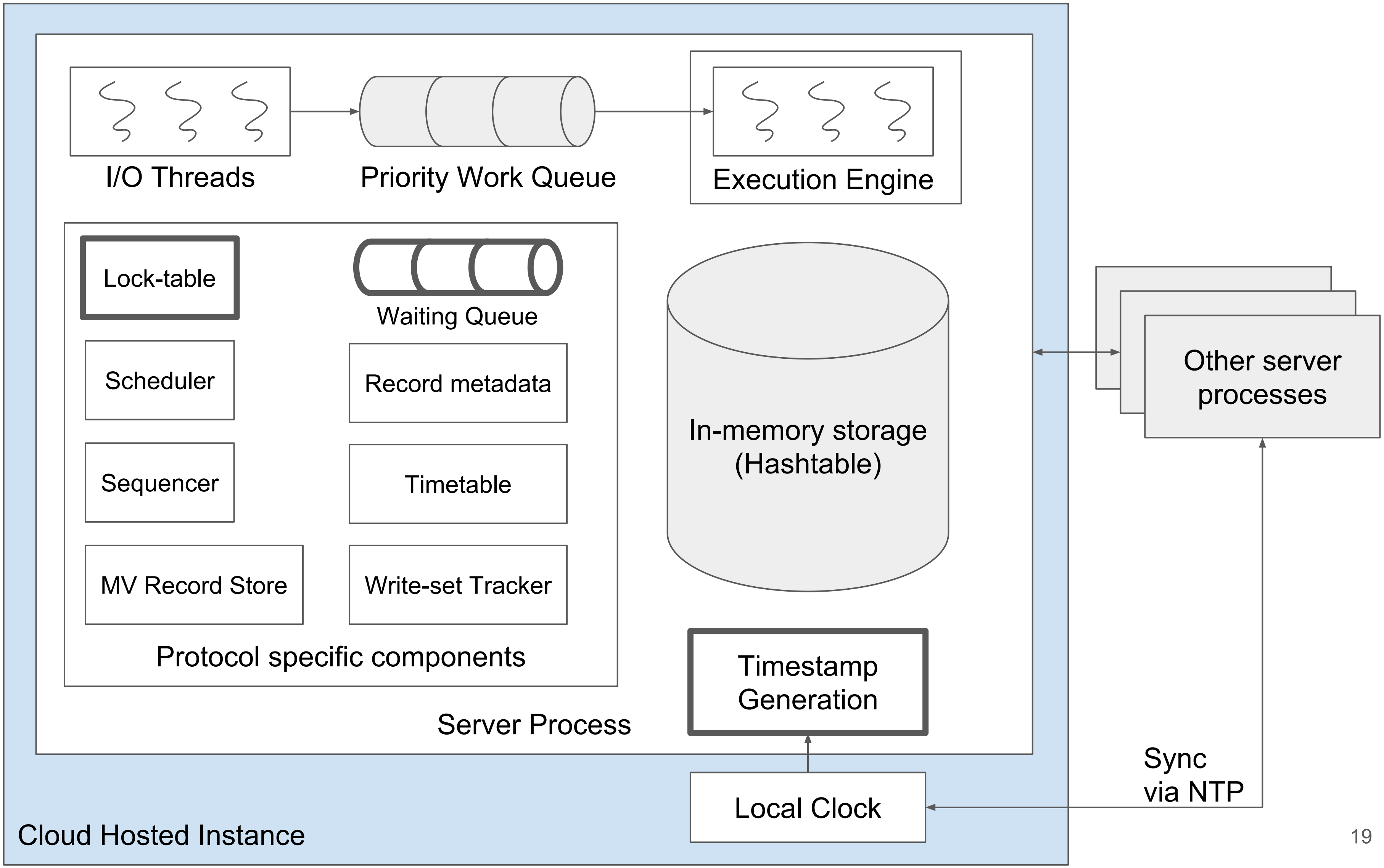


I/O Threads
Priority Work Queue
Execution Engine
Lock-table
Waiting Queue
Scheduler
Record metadata
Sequencer
Timetable
MV Record Store
Write-set Tracker
Protocol specific components
In-memory storage
(Hashtable)
Timestamp
Generation
Server Process
Other servers
Processes
Sync
via NTP
Local Clock
Cloud Hosted Instance

Timestamp Ordering (TIMESTAMP)

- Executes transactions based on the assigned timestamp order
- No bypassing of wait queue
- Avoids deadlocks by aborting older transactions when they conflict with transactions holding records exclusively

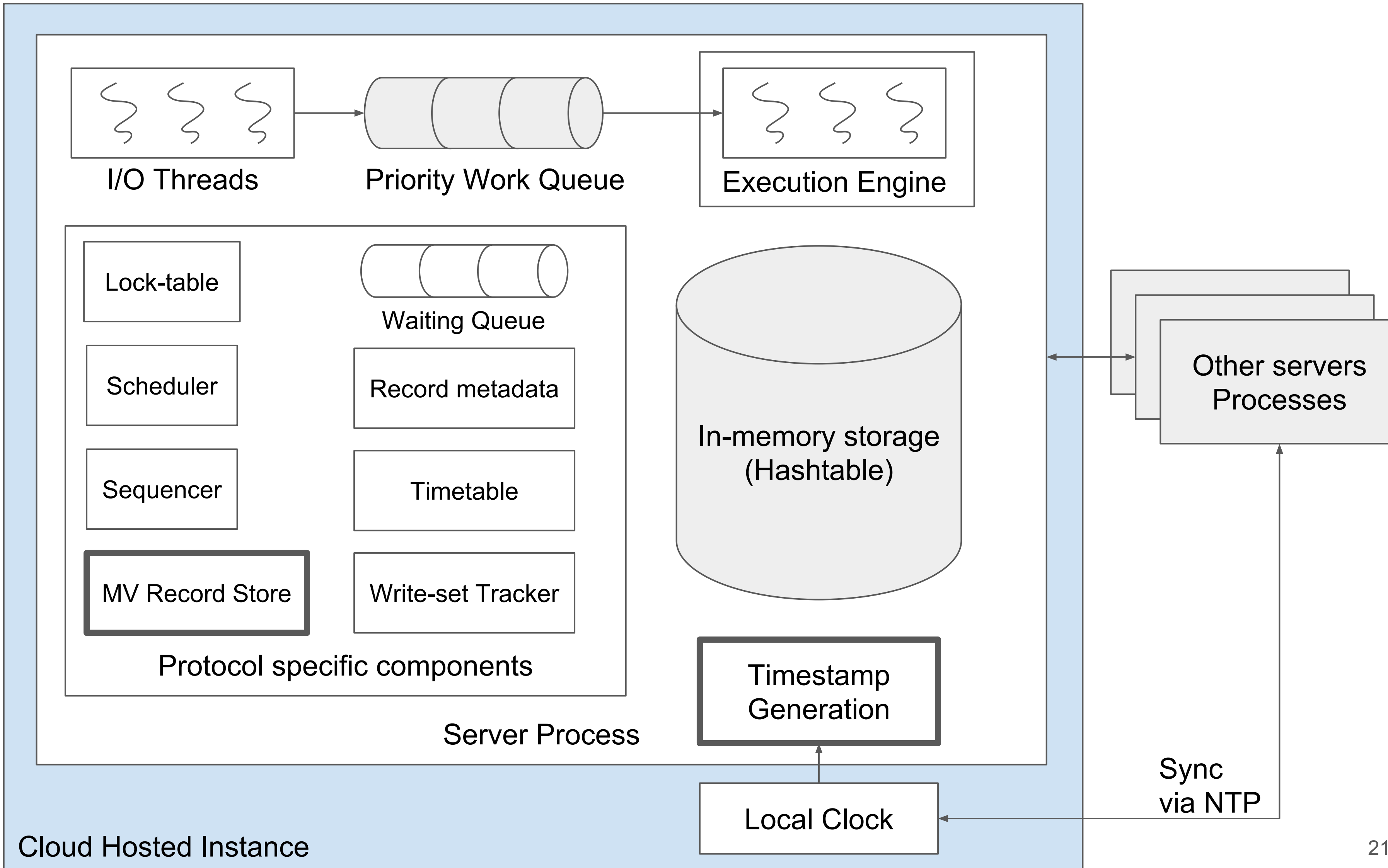
TIMESTAMP


I/O Threads
Priority Work Queue
Execution Engine
Lock-table
Waiting Queue
Scheduler
Record metadata
Sequencer
Timetable
MV Record Store
Write-set Tracker
Protocol specific components
In-memory storage
(Hashtable)
Other server
processes
Timestamp
Generation
Server Process
Sync
via NTP
Local Clock
Cloud Hosted Instance

Multi-version Concurrency Control (MVCC)

- Maintain multiple timestamped copies of each record
- Minimizes conflict between reads and writes
- Limit the number of copies stored
- Abort transactions that try to access records that have been garbage collected

MVCC

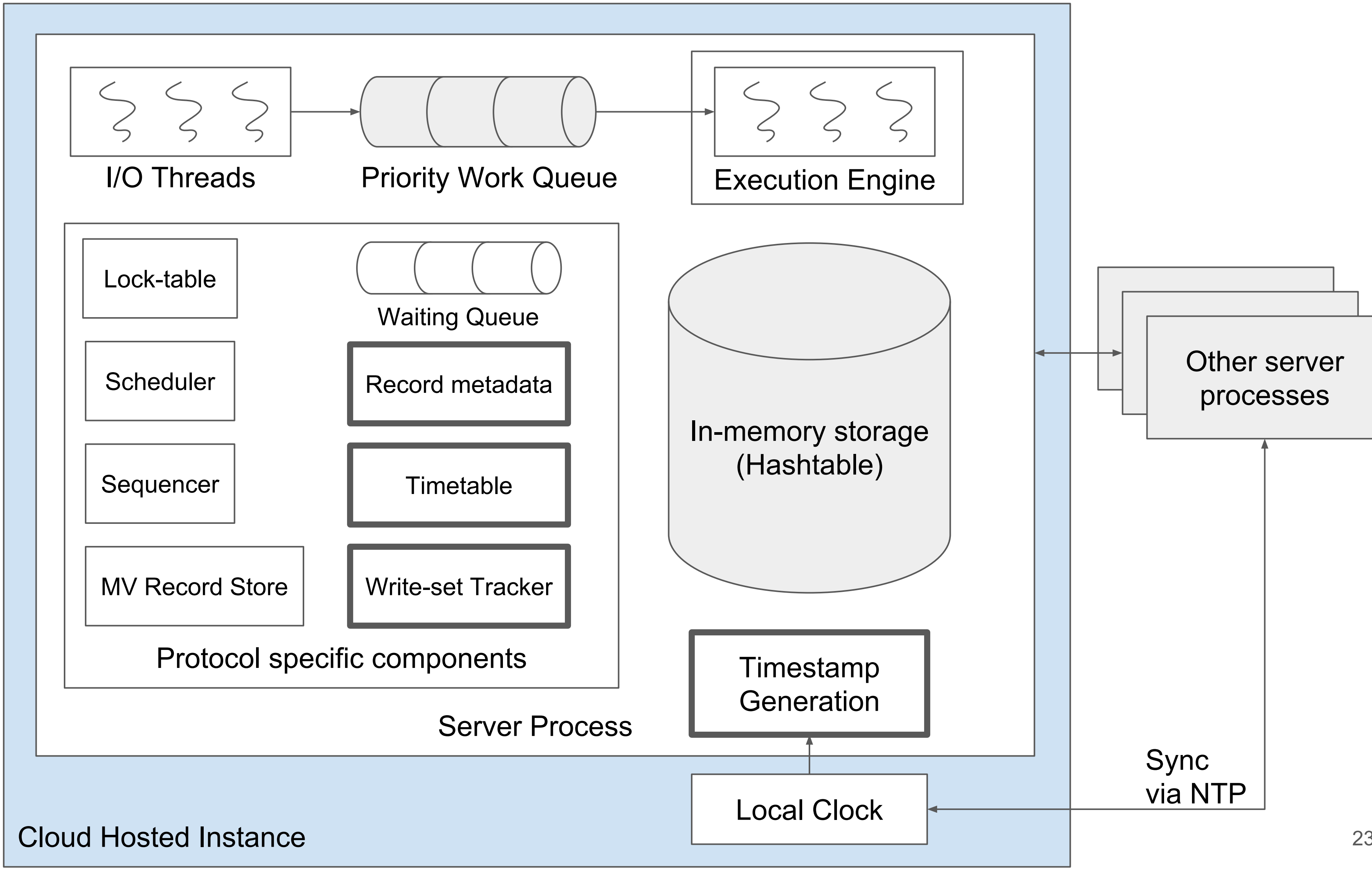


I/O Threads
Priority Work Queue
Execution Engine
Lock-table
Waiting Queue
Scheduler
Record metadata
Sequencer
Timetable
MV Record Store
Write-set Tracker
Protocol specific components
In-memory storage
(Hashtable)
Timestamp
Generation
Server Process
Other servers
Processes
Sync
via NTP
Local Clock
Cloud Hosted Instance

Optimistic Concurrency Control (OCC)

- Based on MaaT [Mahmoud et. al, MaaT protocol, VLDB'14]
- Strong-coupling with 2PC:
  - CC's Validation == 2PC's Prepare phase
- Maintains time ranges for each transaction
- Validation by constraining the time range of the transaction
  - If time range is valid => COMMIT
  - Otherwise => ABORT

OCC

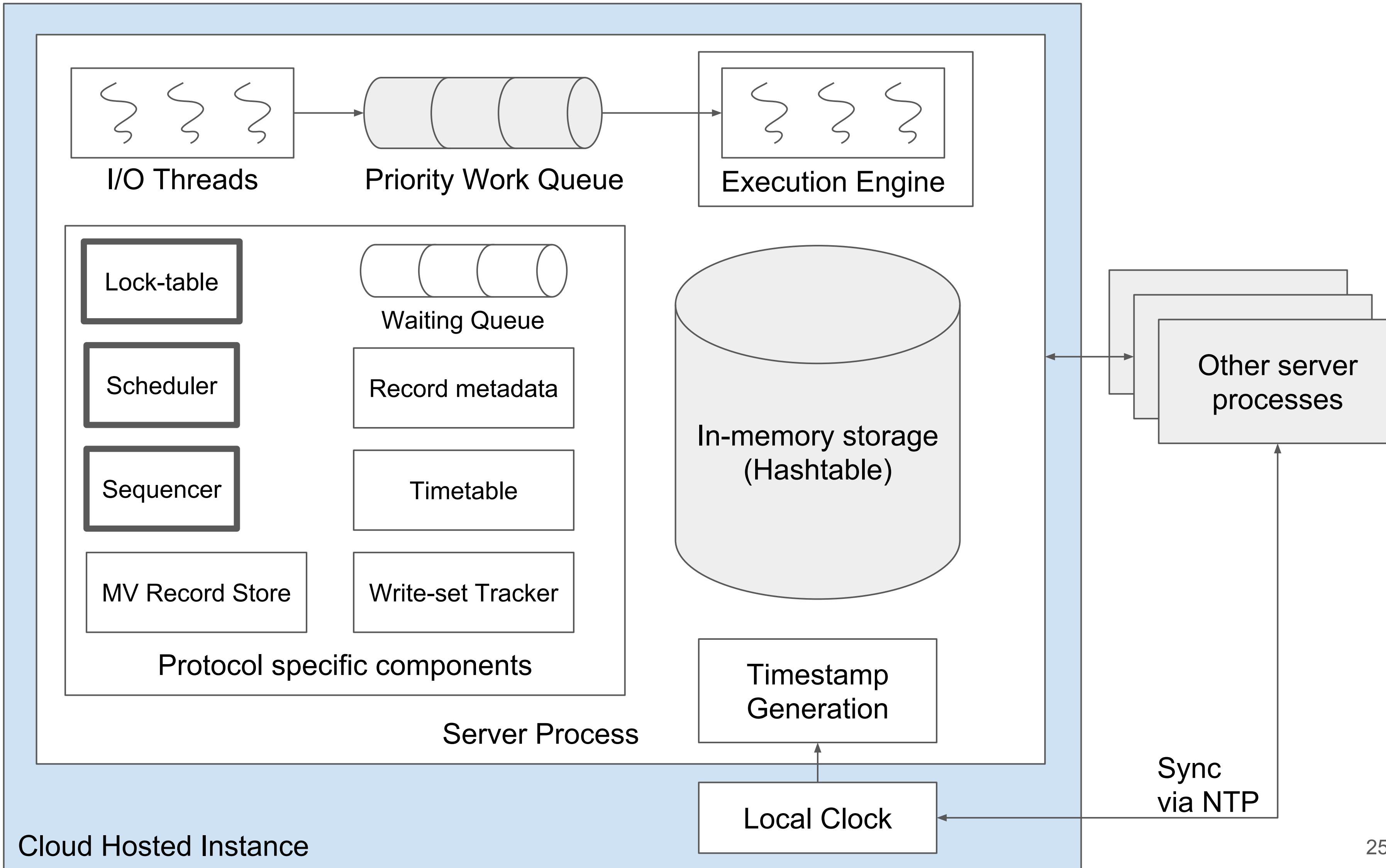


I/O Threads
Priority Work Queue
Execution Engine
Lock-table
Waiting Queue
Scheduler
Record metadata
Sequencer
Timetable
MV Record Store
Write-set Tracker
Protocol specific components
In-memory storage
(Hashtable)
Timestamp
Generation
Server Process
Other server
processes
Sync
via NTP
Local Clock
Cloud Hosted Instance

Deterministic (CALVIN)

- Discussed in previous class
- Key idea: impose a deterministic order on a batch of transactions
- Avoids 2PC
- Unlike others, requires READ_SET and WRITE_SET of transactions to be known a priori, otherwise needs to be computed before starting the execution of the transaction
- In Deneva, a dedicated thread is used for each of sequencer and scheduler.

CALVIN



I/O Threads
Priority Work Queue
Execution Engine
Lock-table
Waiting Queue
Scheduler
Record metadata
Sequencer
Timetable
MV Record Store
Write-set Tracker
Protocol specific components
In-memory storage
(Hashtable)
Timestamp
Generation
Server Process
Other server
processes
Sync
via NTP
Local Clock
Cloud Hosted Instance

Evaluation "Hardware"

- Amazon EC2 instances (m4.2xlarge)

M4 instances are the latest generation of General Purpose Instances. This family provides a balance of compute, memory, and network resources, and it is a good choice for many applications.

Features:

- 2.3 GHz Intel Xeon® E5-2686 v4 (Broadwell) processors or 2.4 GHz Intel Xeon® E5-2676 v3 (Haswell) processors
- EBS-optimized by default at no additional cost
- Support for Enhanced Networking
- Balance of compute, memory, and network resources

Model	vCPU	Mem (GiB)	SSD Storage (GB)	Dedicated EBS Bandwidth (Mbps)
m4.large	2	8	EBS-only	450
m4.xlarge	4	16	EBS-only	750
m4.2xlarge	8	32	EBS-only	1,000
m4.4xlarge	16	64	EBS-only	2,000

Evaluation Methodology

- Table partitions are loaded on each server before each experiment
- Number of open client connections: 10K
- 60 seconds warmup
- 60 seconds measurements
- Throughput measure as the number of successfully completed
- Restart an aborted transaction (due to CC) after a penalization period

Evaluation Workload

- YCSB
- TPC-C: warehouse order processing system
- Product-Part-Supplier

Evaluation Workload

- **YCSB**
  - Single table with 1 primary key and 10 columns of 100B each
    - ~ 16 million records per partition => 16GB per node
  - Each transaction accesses 10 records with independent read and write operation in random order
  - Zipfian distribution of access with theta in [0,0.9]
- TPC-C: warehouse order processing system
- Product-Part-Supplier

Evaluation Workload

- YCSB
- **TPC-C: warehouse order processing system**
  - 9 tables partitioned by warehouse_id
  - Item table is read-only and replicated at every server
  - Implemented two transaction of TPCC specs (88% of workload)
    - Payment: 15% chance to access a different partition
    - NewOrder: ~10% are multi-partition transactions
- Product-Part-Supplier

Evaluation Workload

- YCSB
- TPC-C: warehouse order processing system
- **Product-Part-Supplier**
  - 5 tables: 1 for each products, parts and suppliers. 1 table maps products to parts. 1 table maps partos to suppliers
  - Transactions:
    - Order-Product (MPT): reads parts of a product and decrement the stock quantity of the parts
    - LookupProduct (MPT): (read-only) retrieve parts and their stock quantities
    - UpdateProductPart (SPT): updates product-to-parts mapping

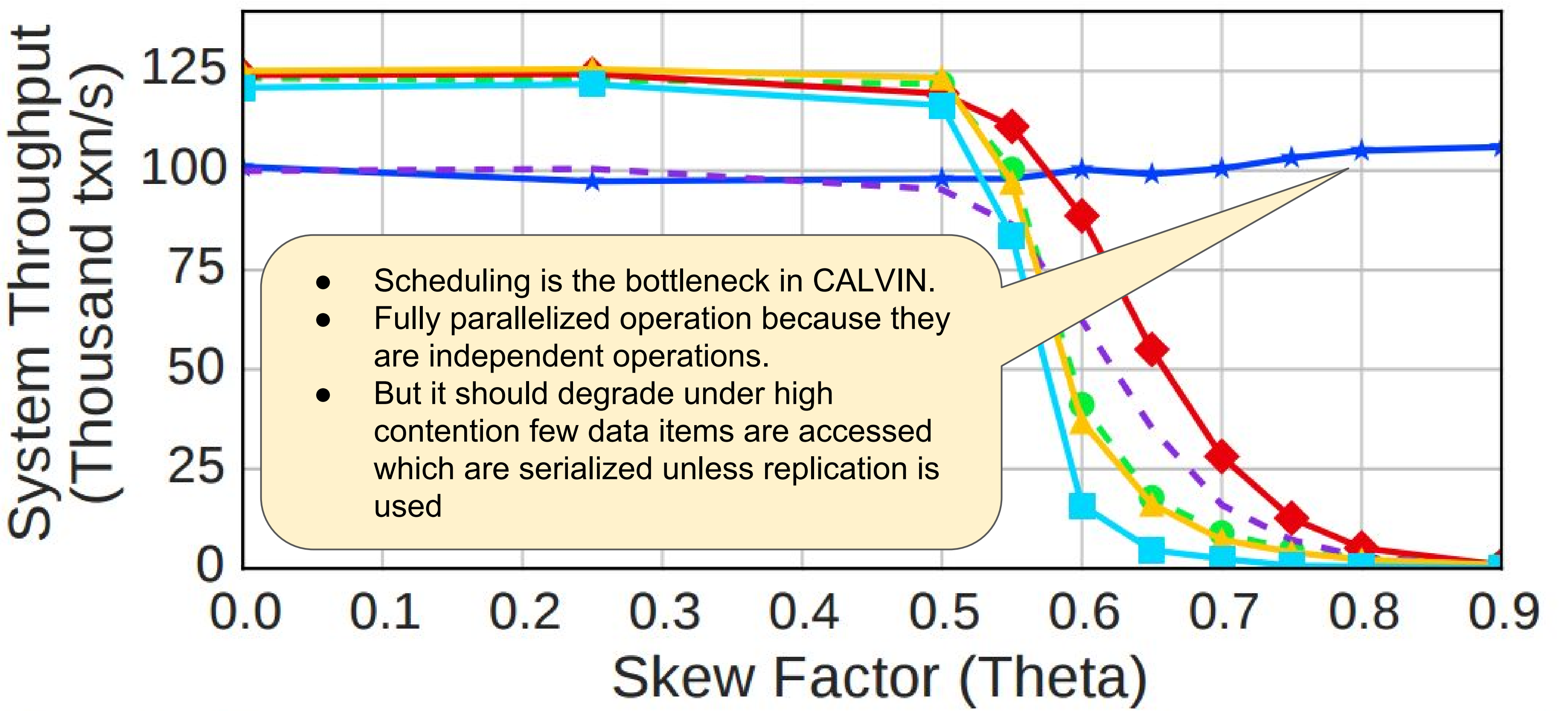


Figure 2: Contention – The measured throughput of the protocols on 16 servers when varying the skew factor in the YCSB workload.

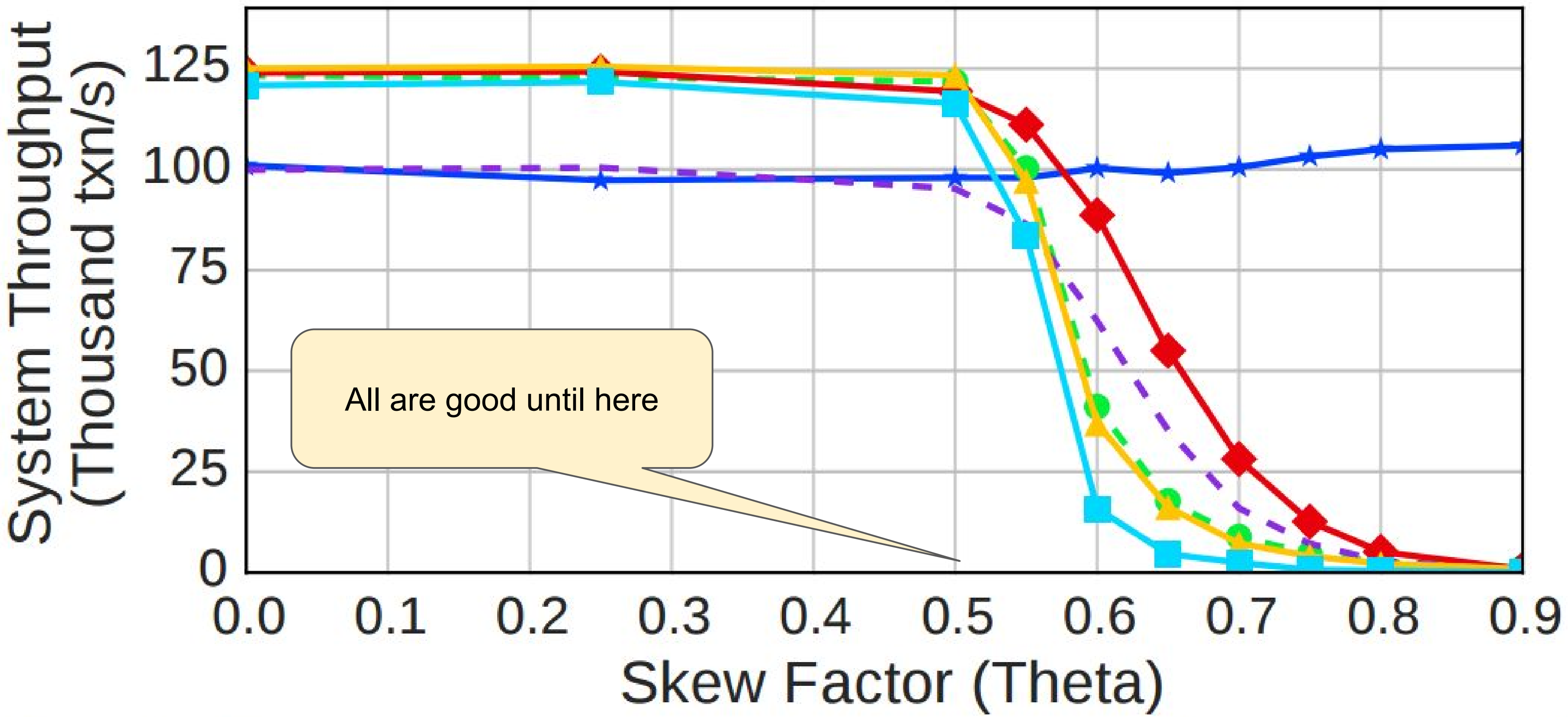


Figure 2: Contention – The measured throughput of the protocols on 16 servers when varying the skew factor in the YCSB workload.

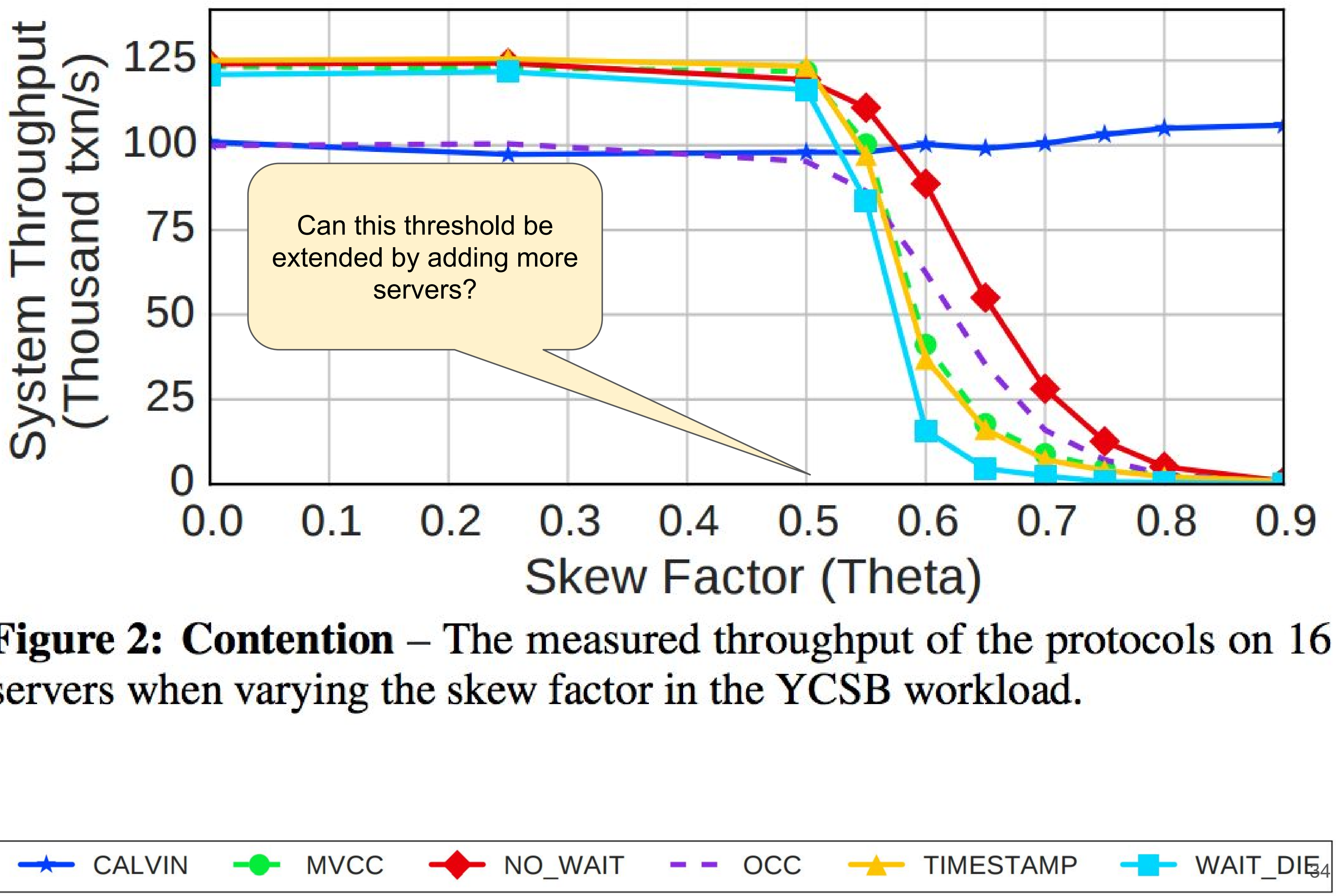


Figure 2: Contention – The measured throughput of the protocols on 16 servers when varying the skew factor in the YCSB workload.

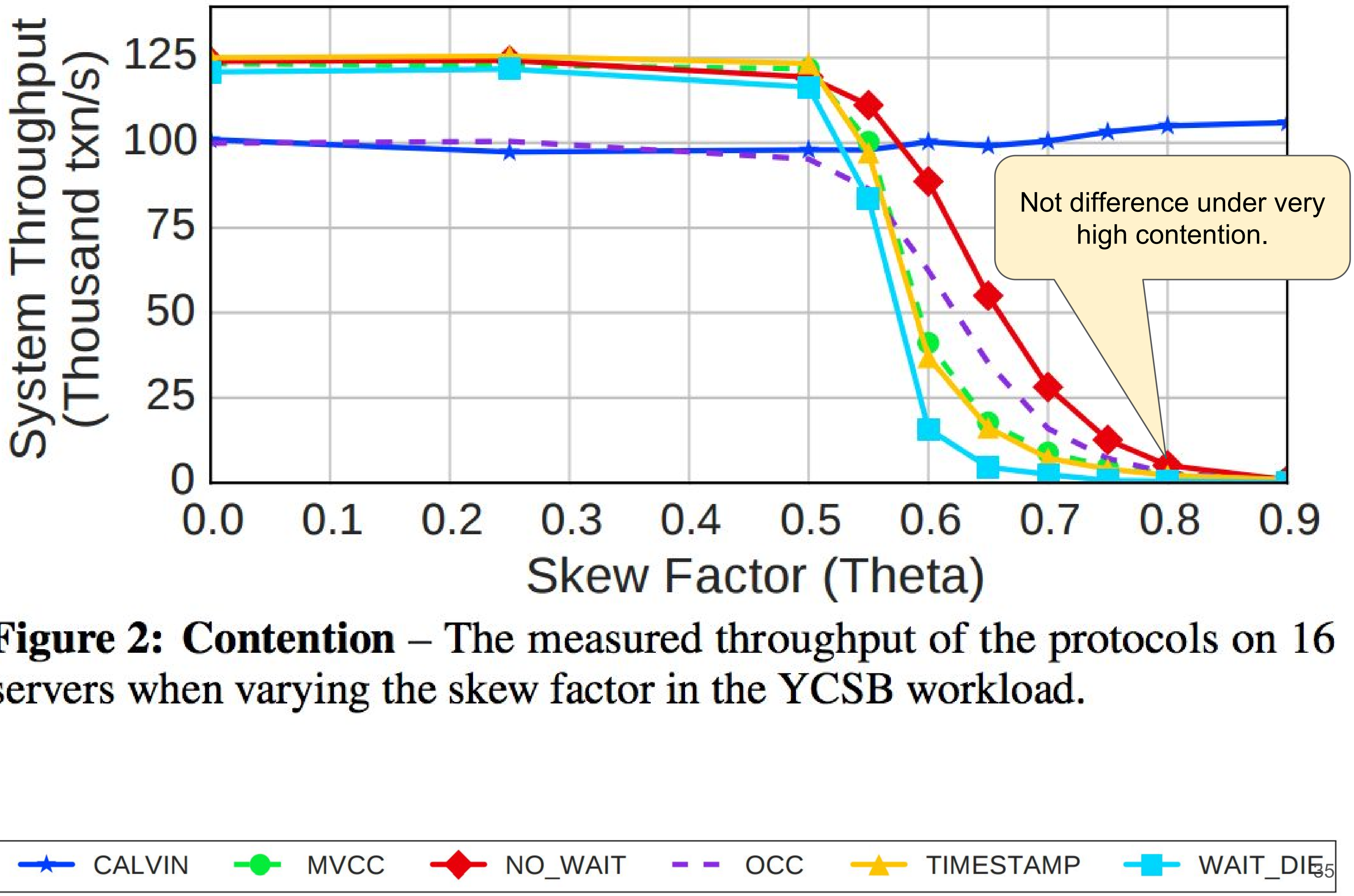


Figure 2: Contention – The measured throughput of the protocols on 16 servers when varying the skew factor in the YCSB workload.

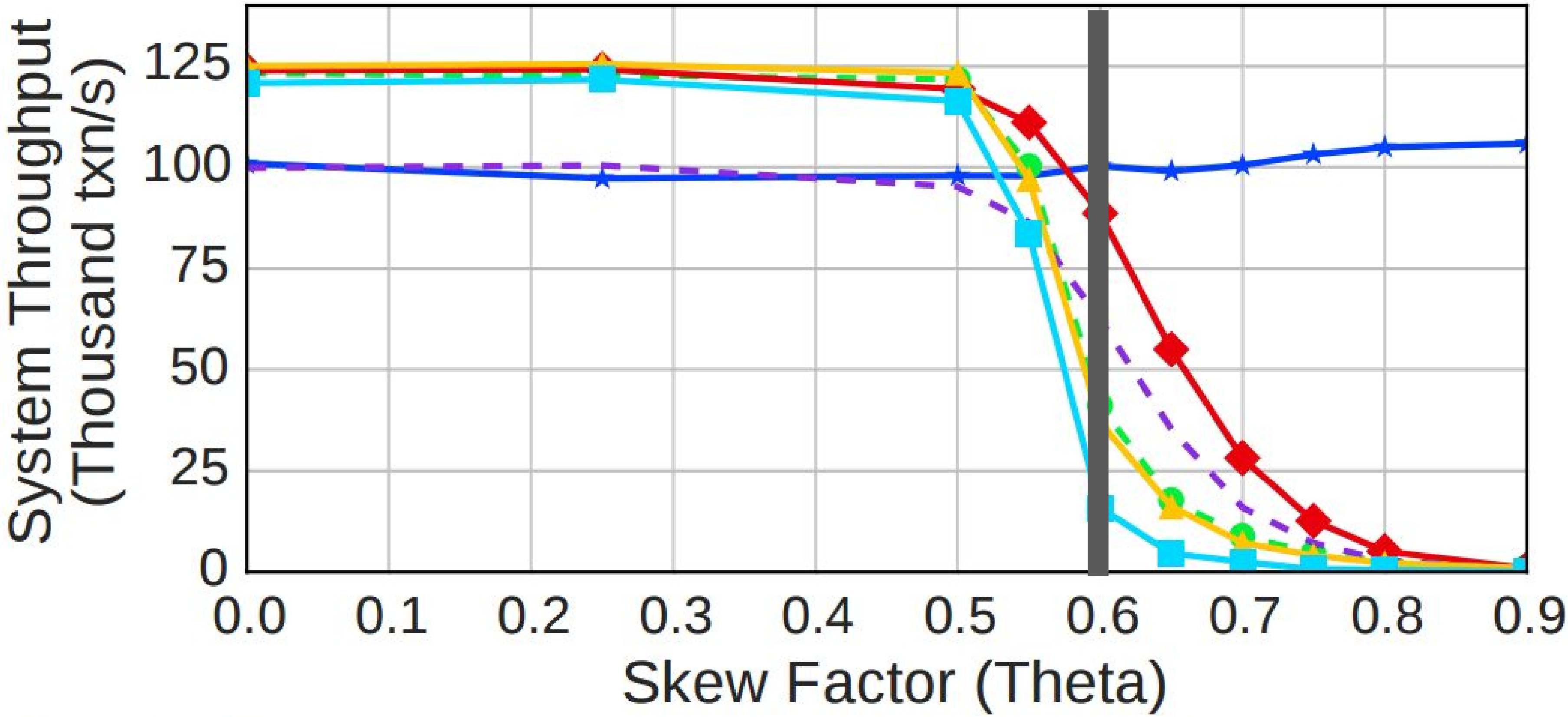


Figure 2: Contention – The measured throughput of the protocols on 16 servers when varying the skew factor in the YCSB workload.

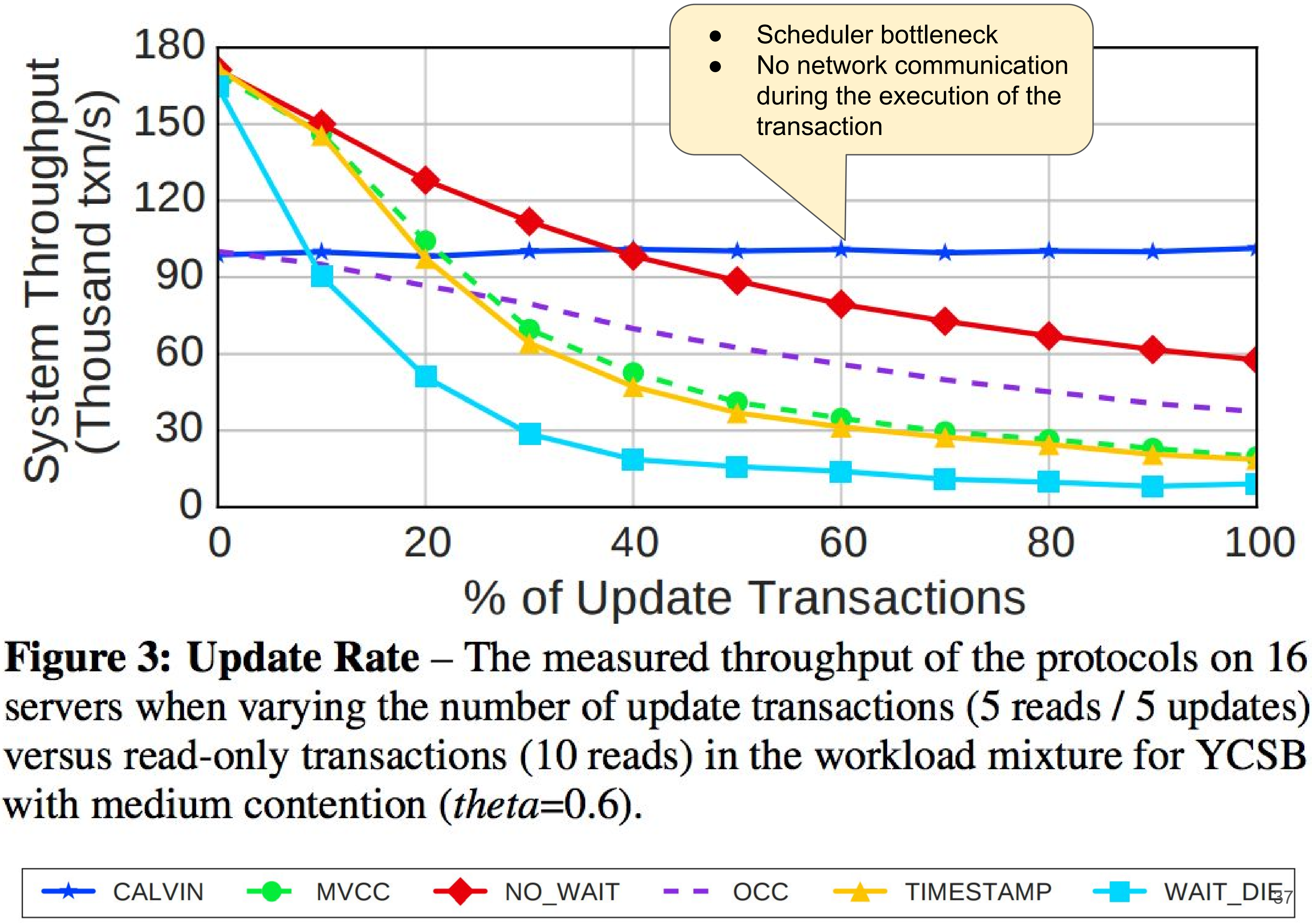


Figure 3: Update Rate – The measured throughput of the protocols on 16 servers when varying the number of update transactions (5 reads / 5 updates) versus read-only transactions (10 reads) in the workload mixture for YCSB with medium contention (*theta*=0.6).

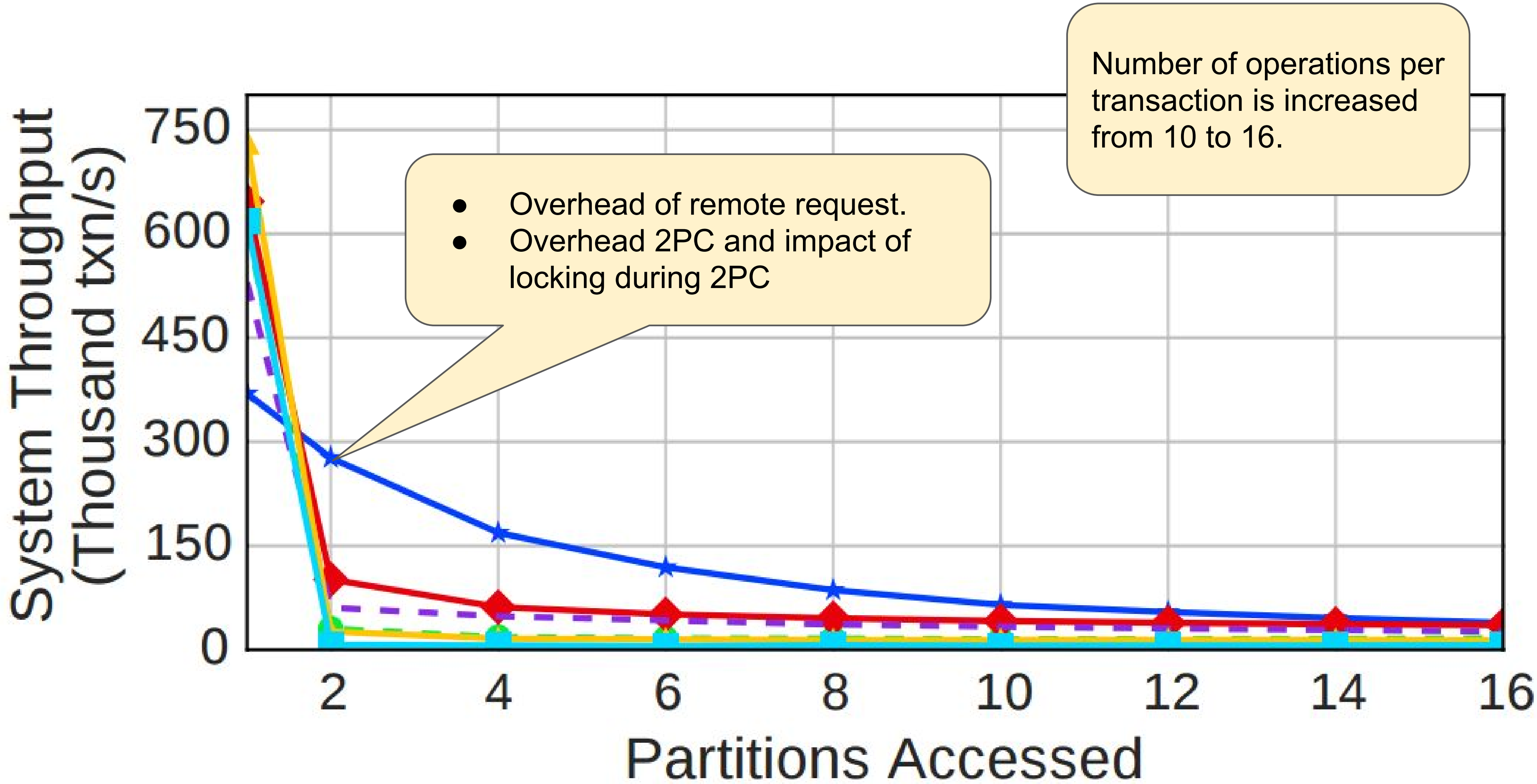
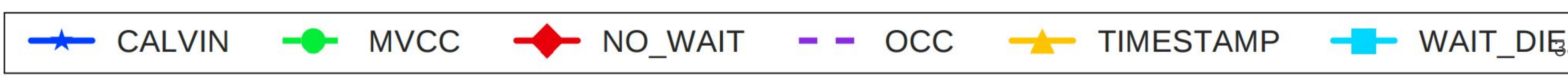


Figure 4: Multi-Partition Transactions – Throughput with a varying number of partitions accessed by each YCSB transaction.



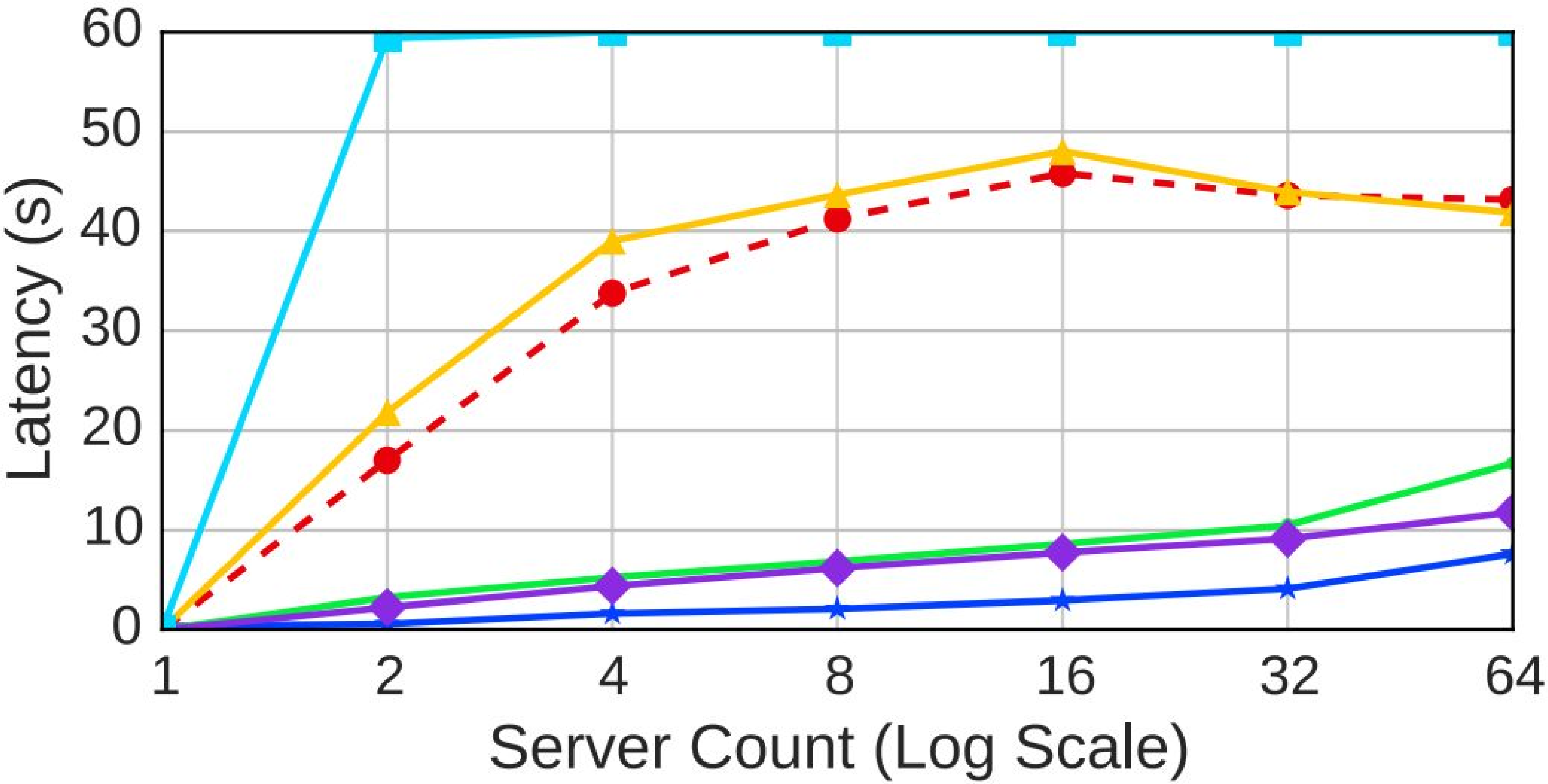


Figure 7: 99%ile Latency – Latency from a transaction's first start to its final commit for varying cluster size.

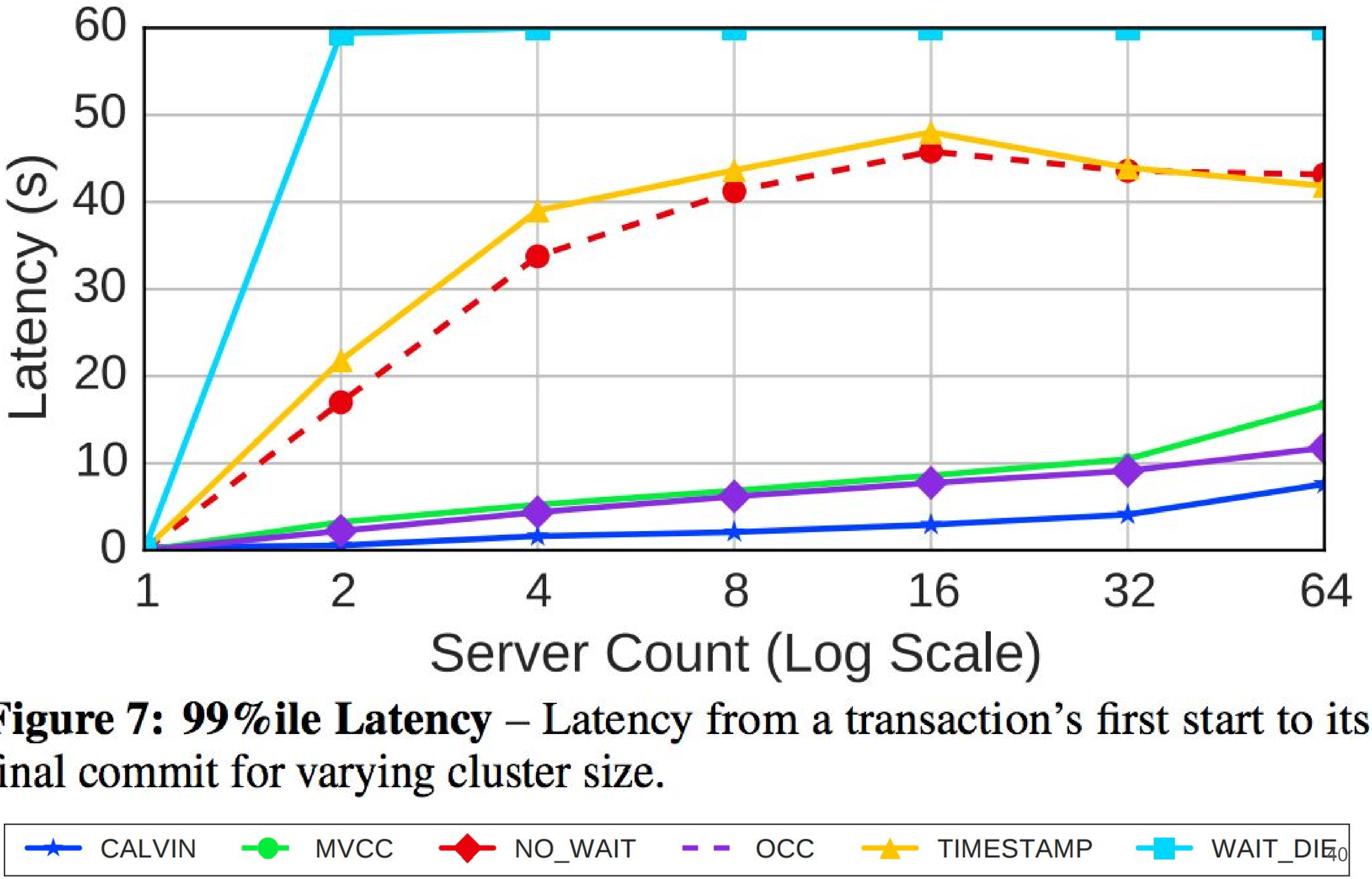
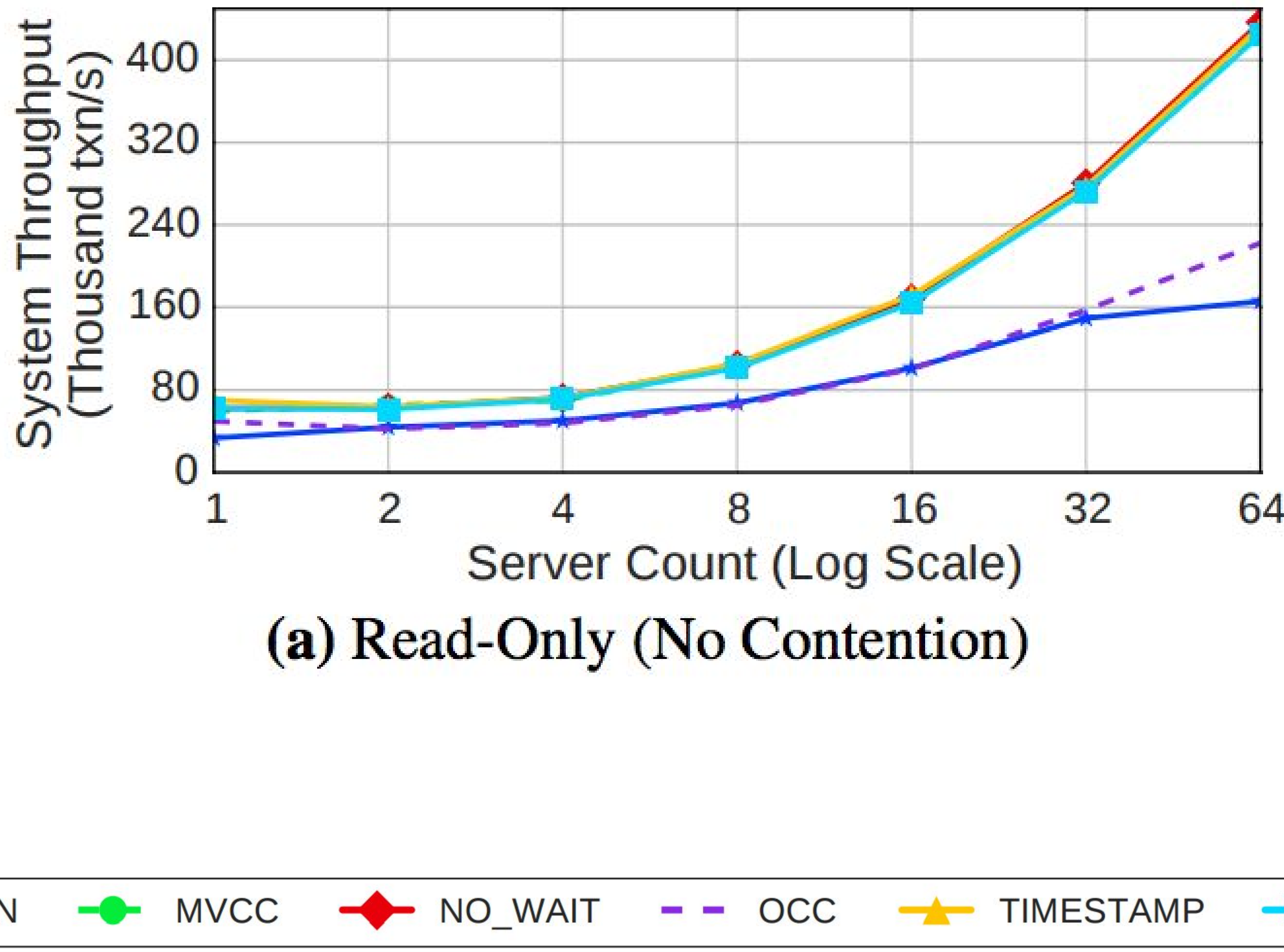


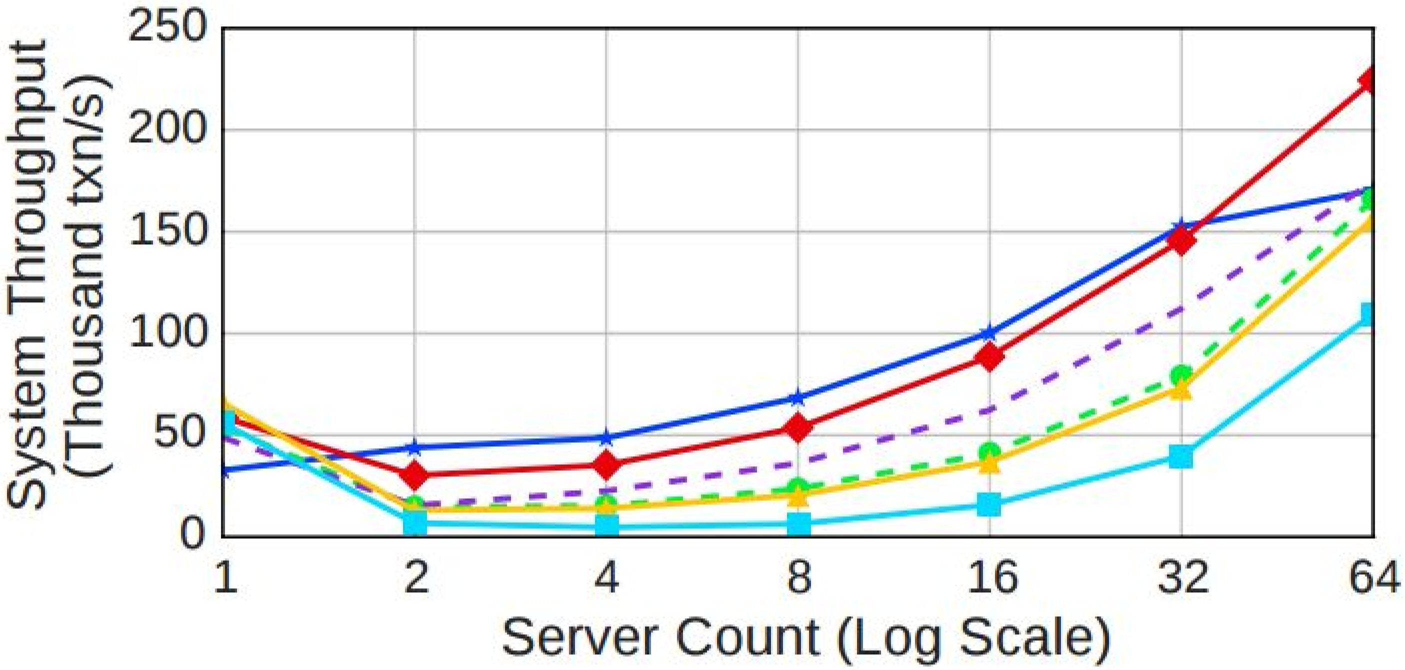
Figure 7: 99%ile Latency – Latency from a transaction's first start to its final commit for varying cluster size.

Scalability (no contention)



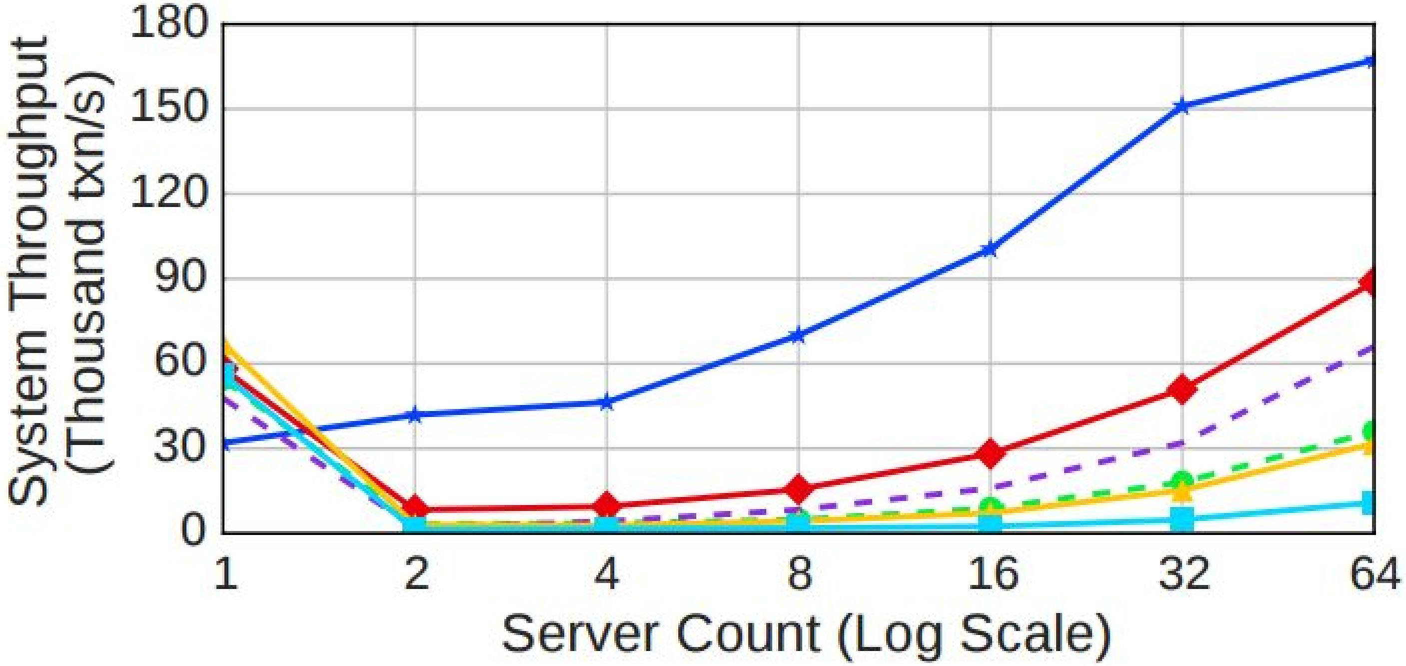
(a) Read-Only (No Contention)

Scalability (medium contention)



(b) Read-Write (Medium Contention)

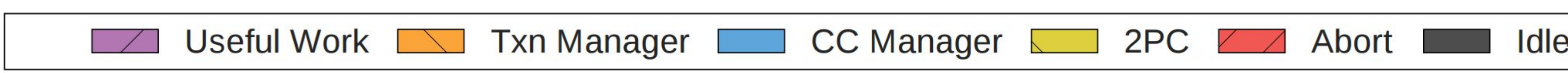
Scalability (high contention)



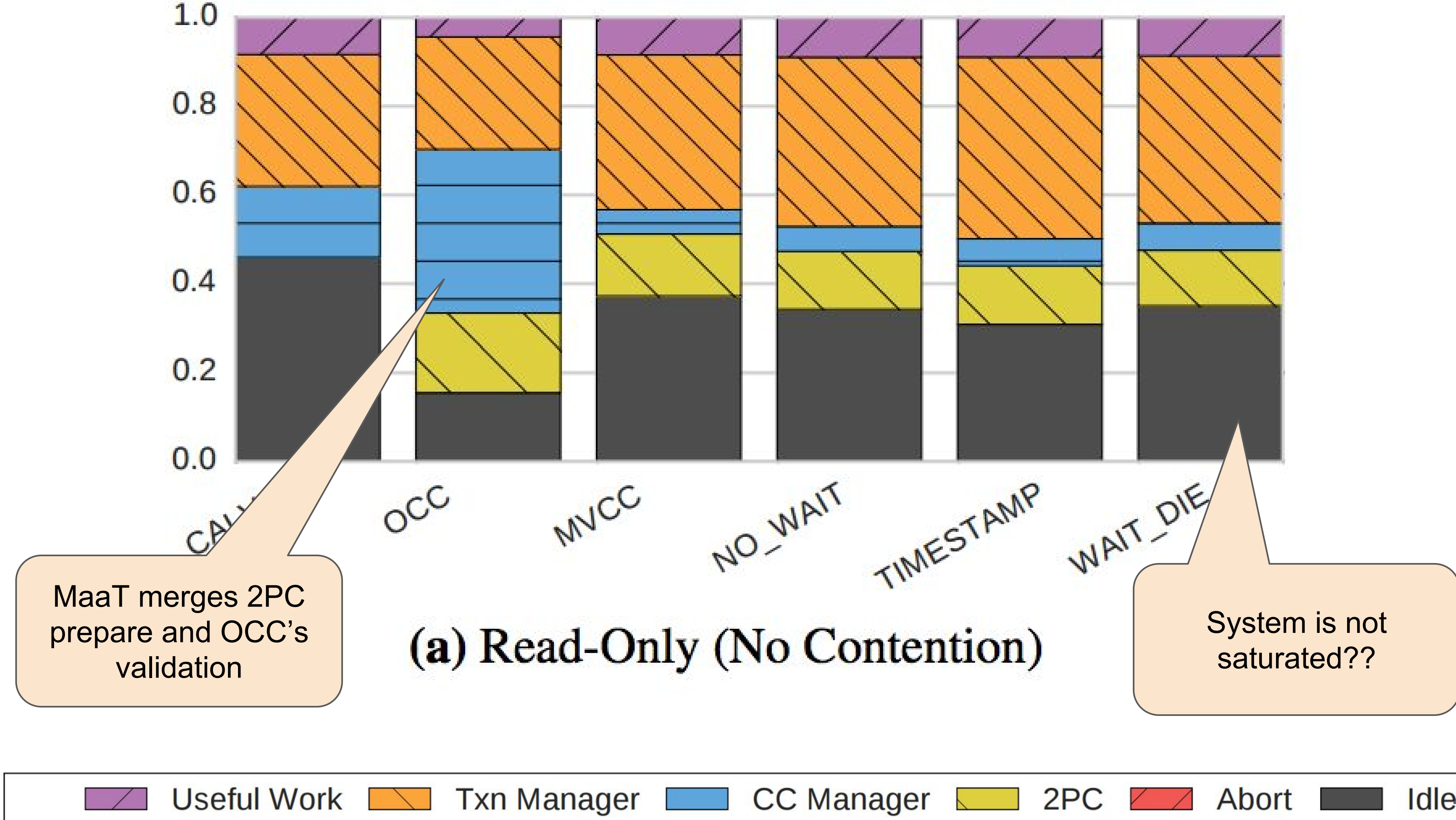
(c) Read-Write (High contention)

Scalability (Breakdown)

- **USEFUL WORK**: All time that the workers spend doing computation on behalf of read or update operations.
- **TXN MANAGER**: The time spent updating transaction metadata and cleaning up committed transactions.
- **CC MANAGER**: The time spent acquiring locks or validating as part of the protocol. For CALVIN, this includes time spent by the sequencer and scheduler to compute execution orders.
- **2PC**: The overhead from two-phase commit.
- **ABORT**: The time spent cleaning up aborted transactions.
- **IDLE**: The time worker threads spend waiting for work.

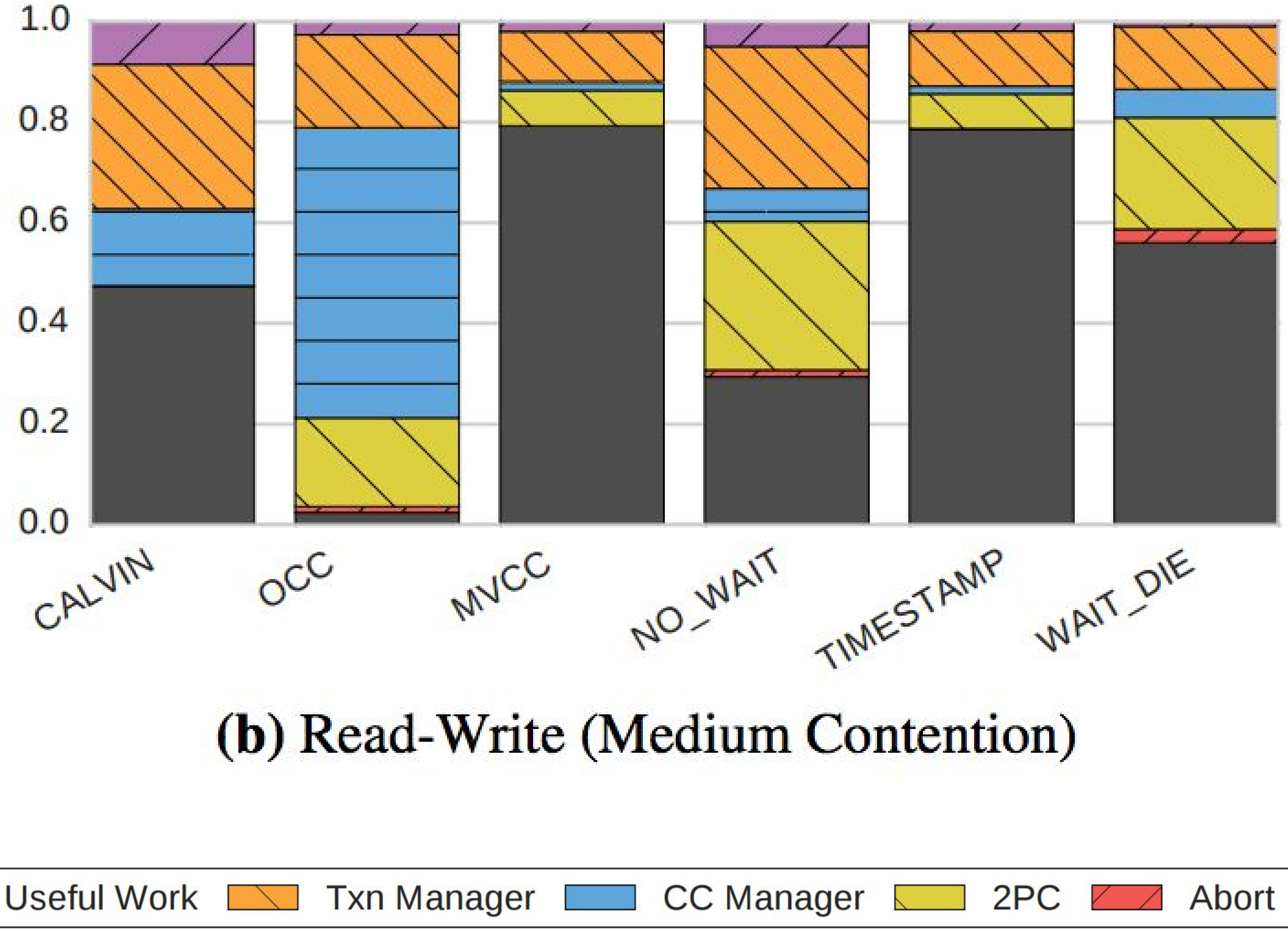


Scalability (Breakdown - no contention)



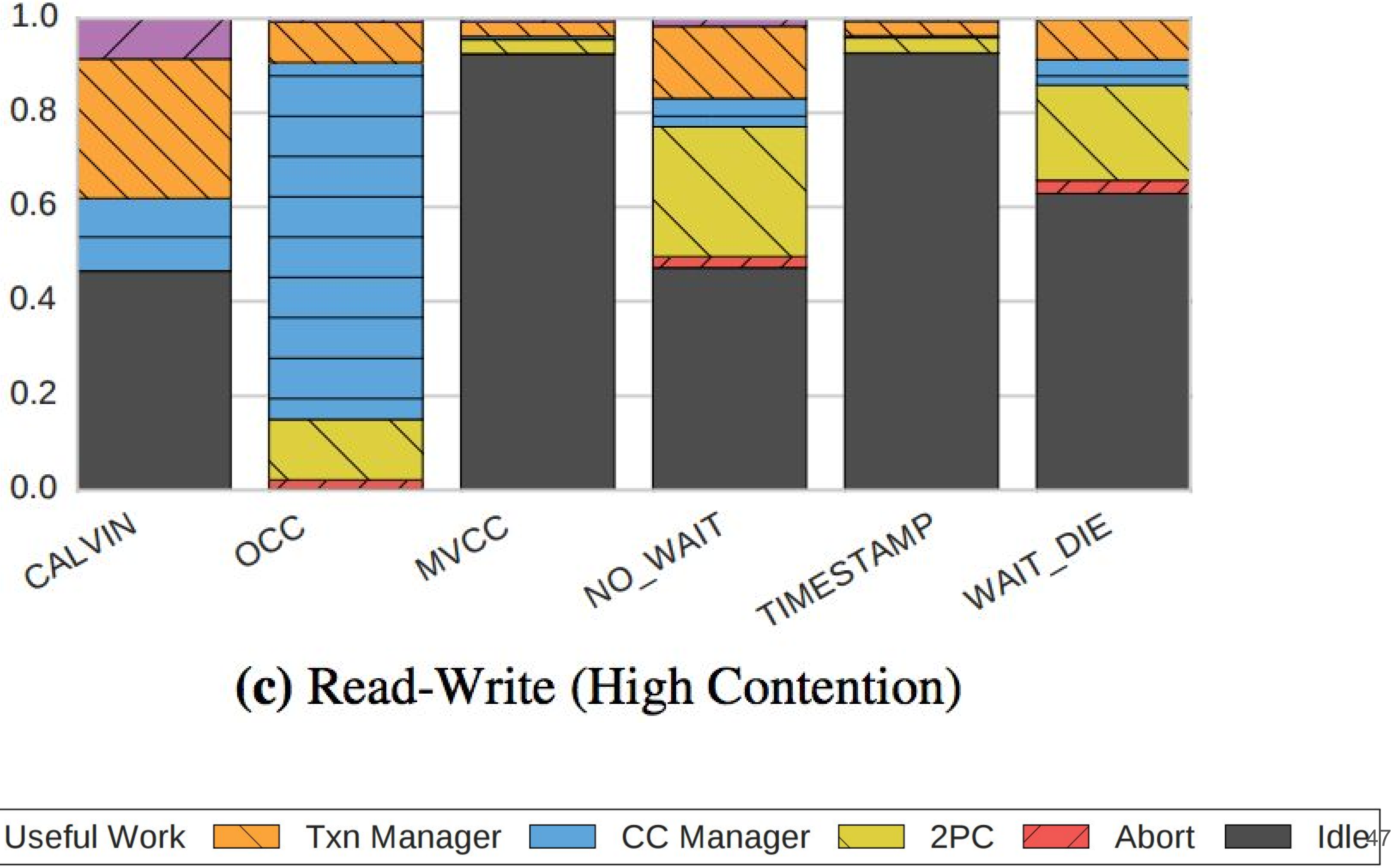
(a) Read-Only (No Contention)

Scalability (Breakdown - medium contention)



(b) Read-Write (Medium Contention)

Scalability (Breakdown - high contention)



(c) Read-Write (High Contention)

Latency breakdown

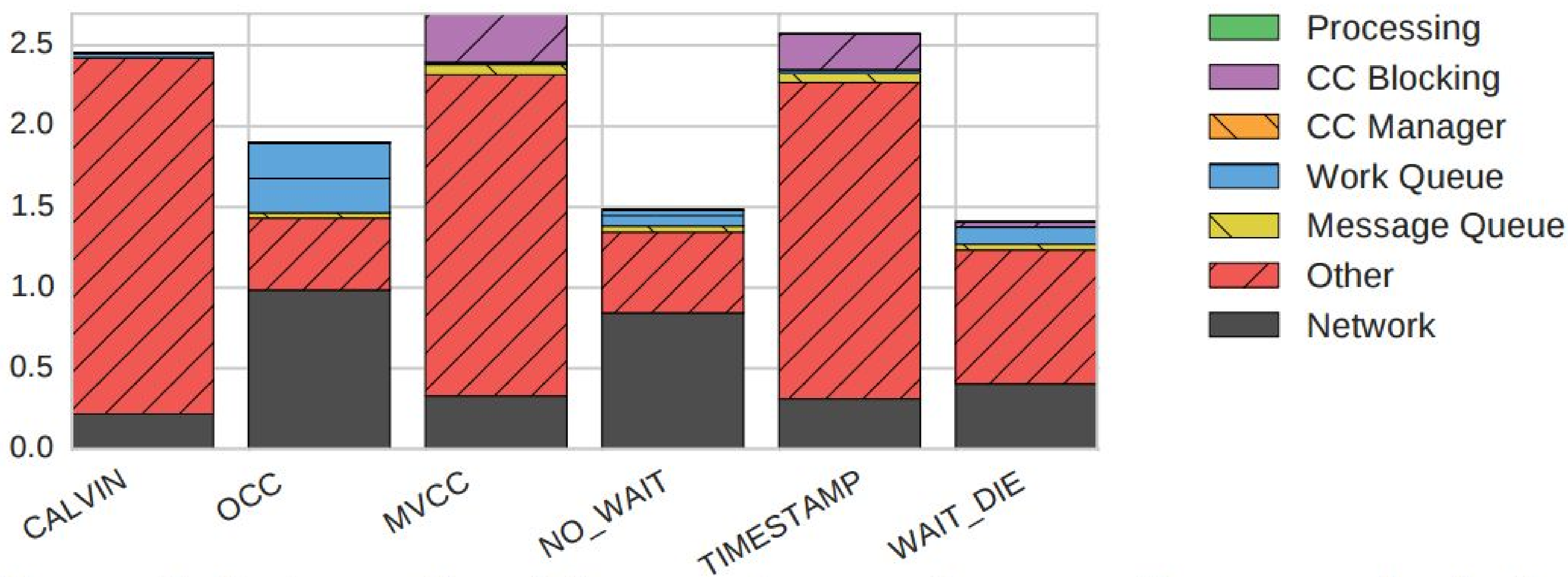


Figure 8: Latency Breakdown – Average latency of a transaction's final execution before commit.

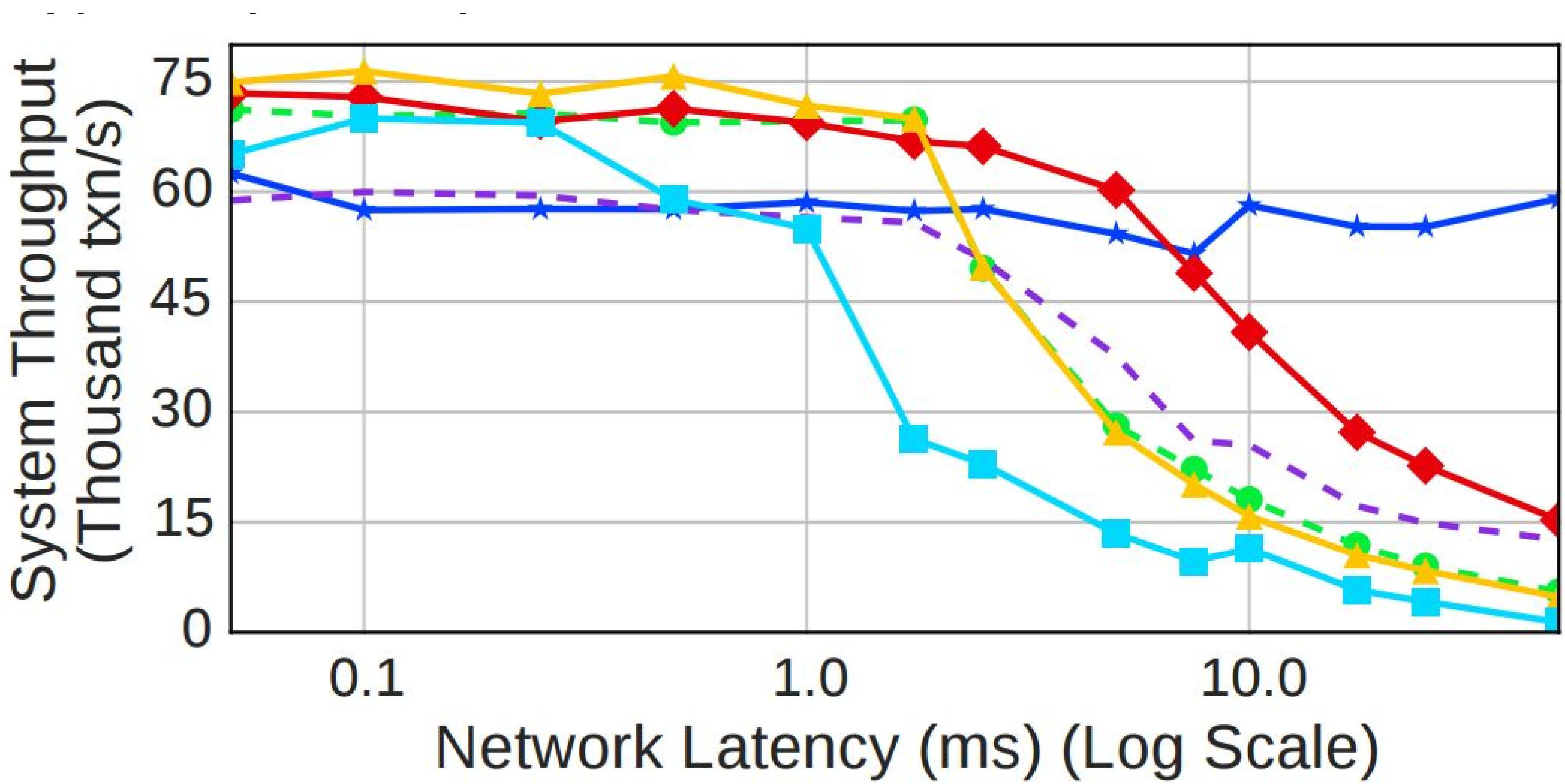


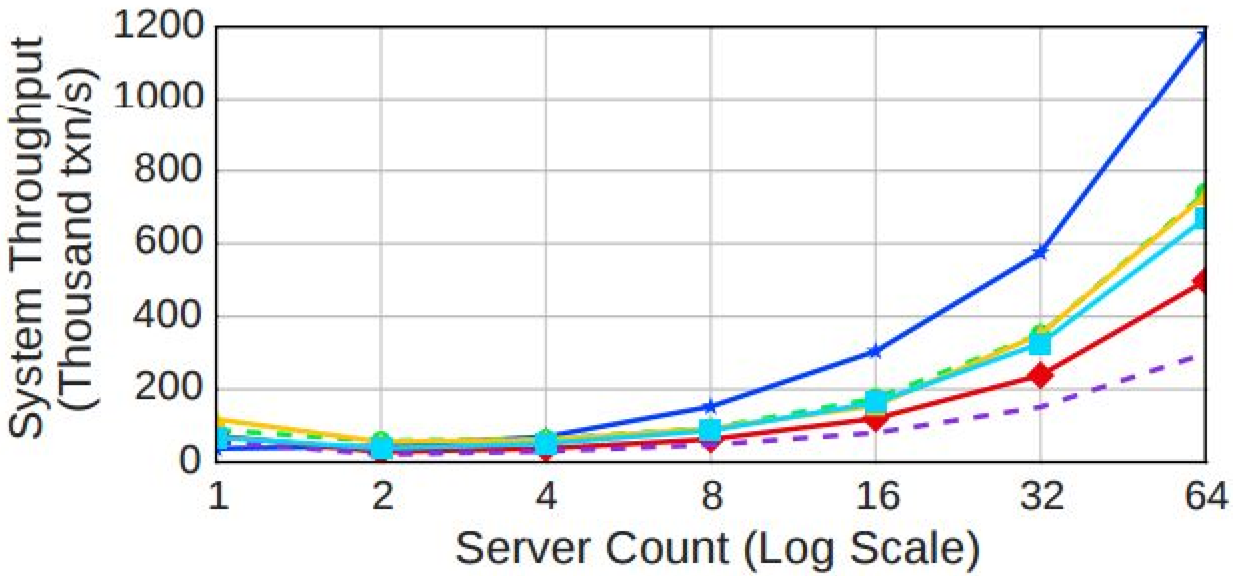
Figure 9: Network Speed – The sustained throughput measured for the concurrency protocols for YCSB with artificial network delays.

Table 2: Multi-Region Cluster – Throughput of a 2-node cluster with servers in AWS US East and US West regions.

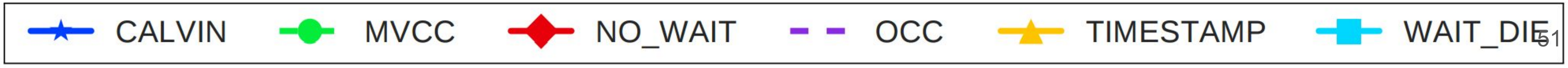
Algorithm	CALVIN	OCC	MVCC
Throughput	8,412	11,572	5,486

Algorithm	NO_WAIT	TIMESTAMP	WAIT_DIE
Throughput	15,921	4,635	4,736

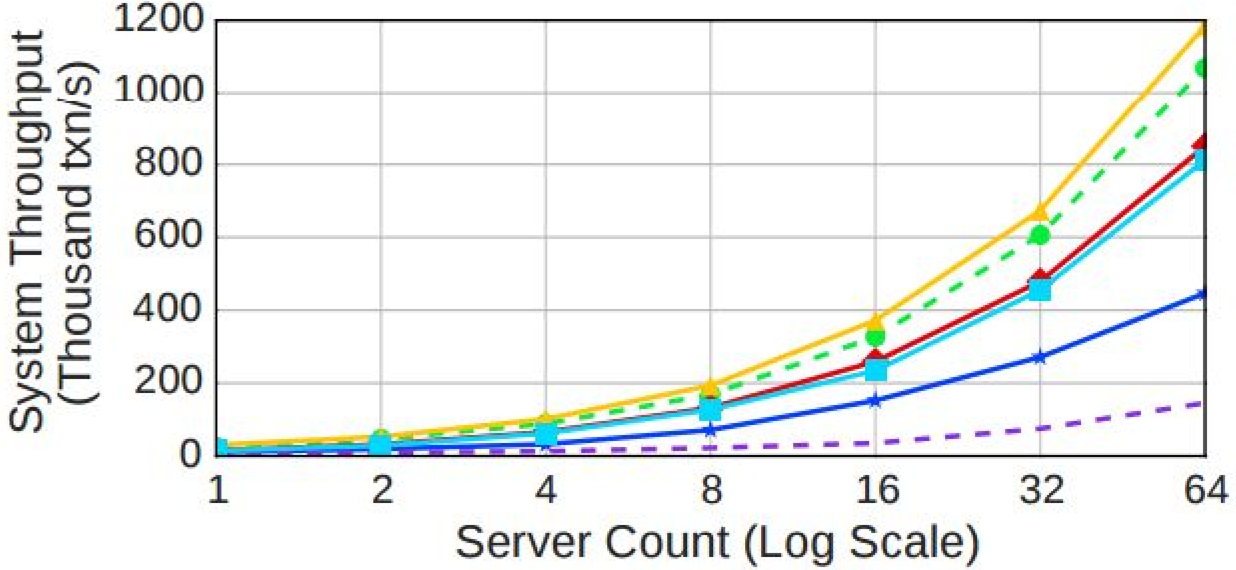
Scalability - TPCC - Payment transaction



(a) Payment Transaction



Scalability - TPCC - NewOrder transaction



(b) NewOrder Transaction

Data dependant aborts

- YCSB operation are independent
- Modified YCSB transction to have conditional abort based a value read.
- 36% decrease in performance compared to 2%-10% descease on other protocols.
    - theta=0.6 , 50% updates
- CALVIN performs worse with higher contention (drops 73K to 19K txn/s)

Results Summary

Class	Algorithm	2PC delay	MPT	Low Contention	High Contention
Locking	NO_WAIT, WAIT_DIE	B	B	A	B
Timestamp	TIMESTAMP, MVCC	B	B	A	B
Optimistic	OCC	B	B	B	A
Deterministic	CALVIN	NA	B	B	A

Bottlenecks in DDBMS

- According to the paper, it boils down to the following bottlenecks:
- 2PC delay
  - CALVIN is designed to eliminate that but in case a transaction will need to abort. It needs to pay the cost of broadcasting the abort decision
- Data access contention
  - Read-only contention can be trivially solved by replication
  - Write contention is difficult

Further research and additional potential solutions

- Authors mentions many aspects for future research and solutions:
  - Impact of recovery mechanisms
  - Leverage better network technologies (e.g. RDMA)
  - Automatic repartitioning [Schism, H-Store]
  - Force a data model adaptation on application developers
    - (e.g. entity group- Helland CIDR'07, G-Store)
  - Semantic based concurrency control methods

- Is there a way to generalize CC protocols into a framework that admits different configurations and yield different CC protocols implementation?
  - e.g. Similar to GiST generalizes search tree for indexes, and SP-GiST generalizes space-partitioning trees.

- Contention-aware adaptive concurrency control
  - 2PL or Timestamp under low contention and switch to OCC or CALVIN under high contention

- Evaluating abort rate