A Performance Study of Epoch-based Commit Protocols in Distributed OLTP Databases

J. Waudby[1], P. Ezhilchelvan[1], I. Mitrani[1], J. Webber[2]

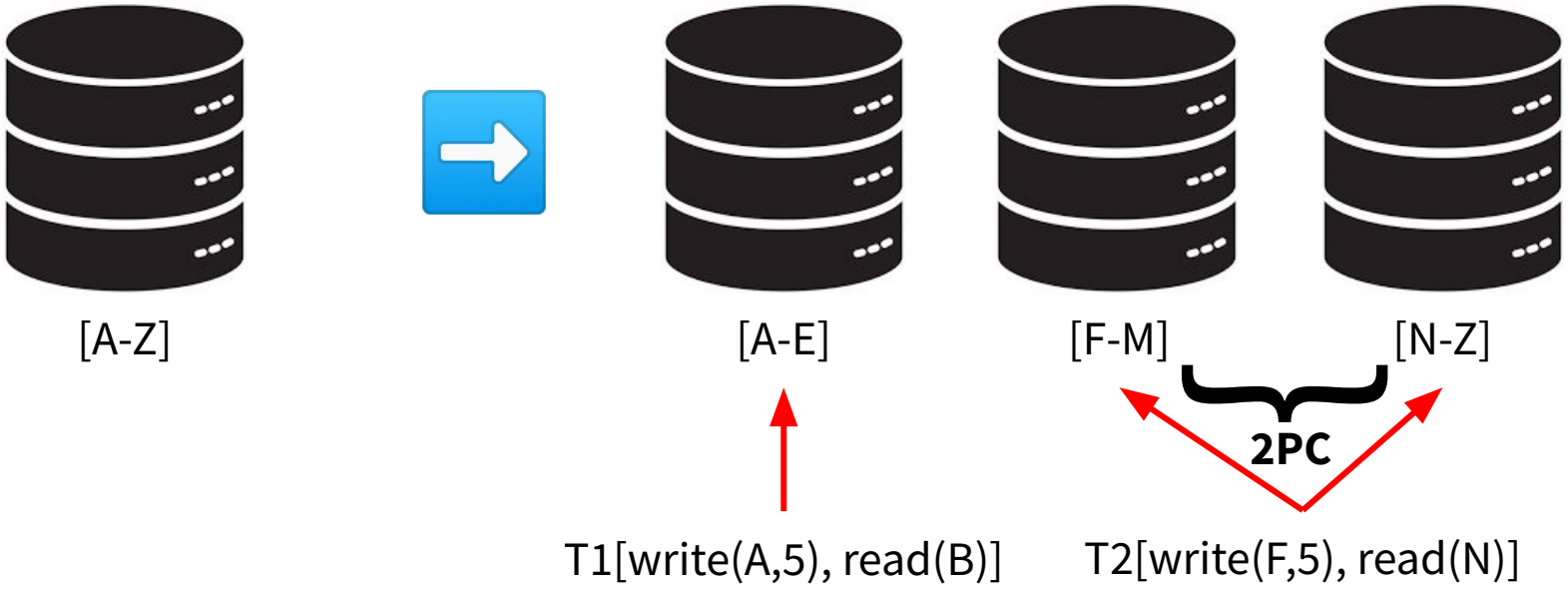
22 Sept 2022

1. Newcastle University
2. Neo4j

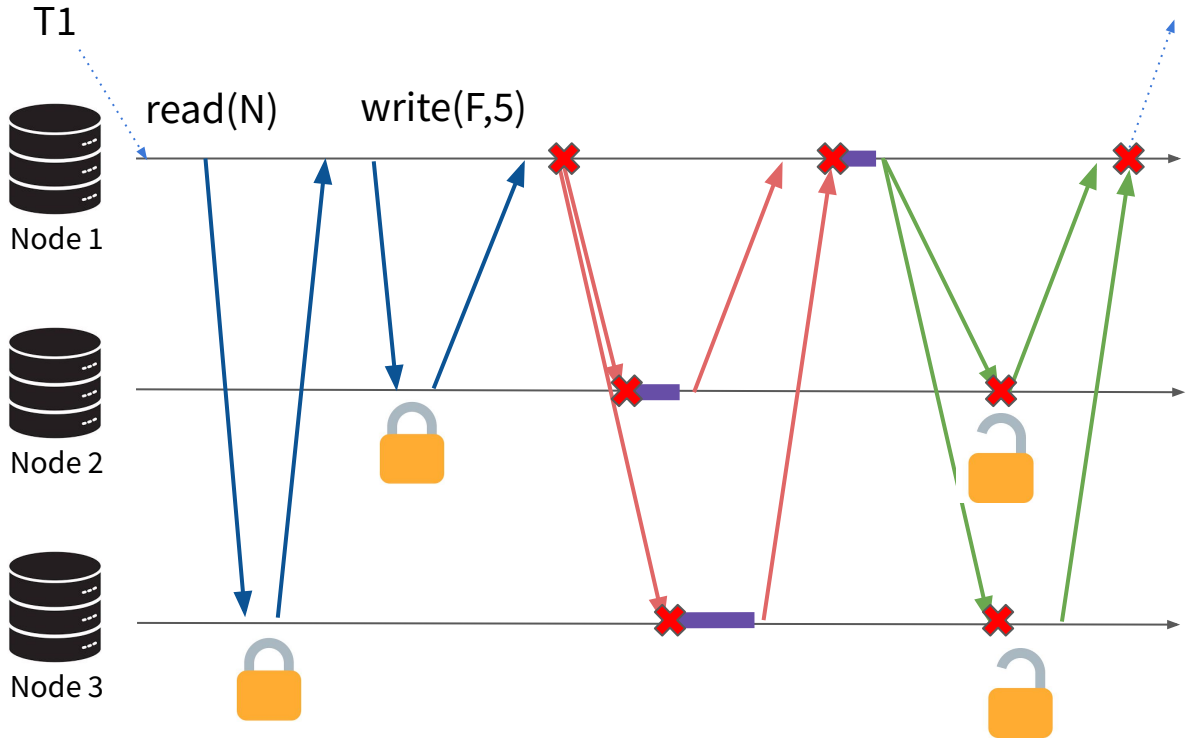
Distributed Transaction Processing (OLTP) Databases

Single-node database

Distributed database



The Problem with 2PC



N1 coordinating distributed txn

Performance killer:

2 sequential network round trips → expensive compared to single-node transaction

Multiple durable writes: disk flush can be 10us - 10ms → increased latency

Factors combine to cause higher contention

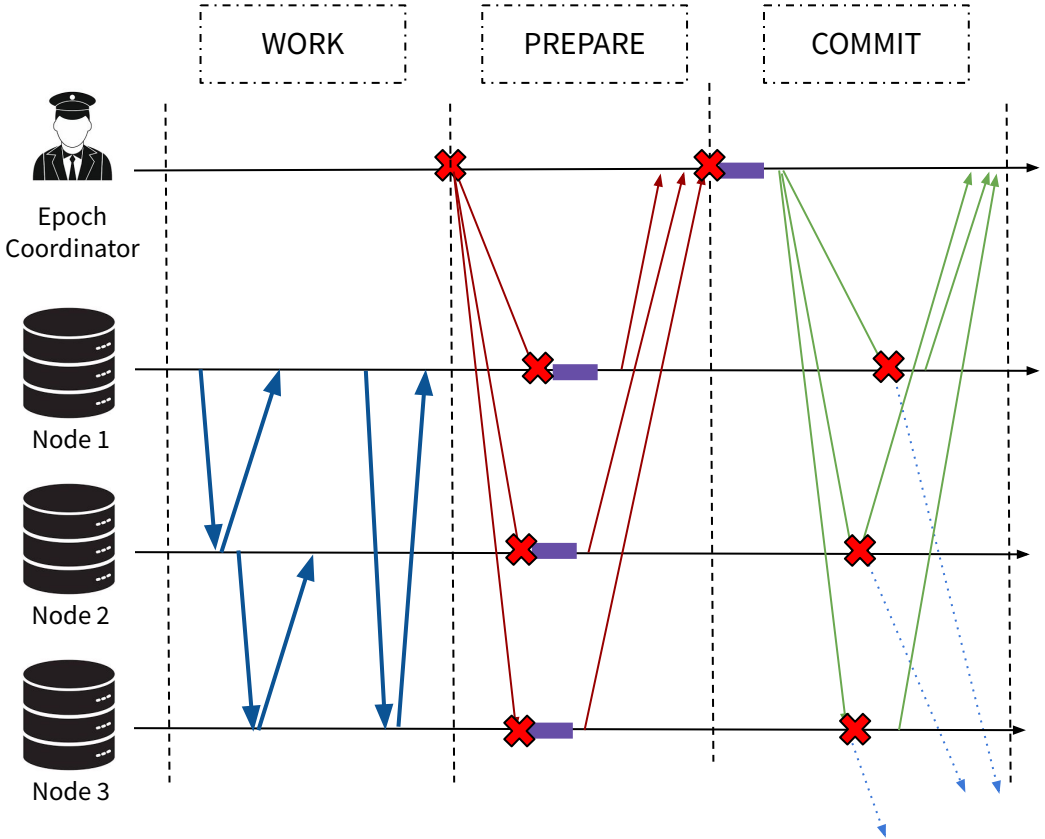
2PC: Research Strikes Back

- Minimise #distributed transactions via workload-driven partitioning [Schism] → *impossible to avoid completely*
- Eliminate by dynamic re-partitioning [G-Store, LEAP]→ *costs to move data around*
- Deterministic databases [Calvin, SLOG] → *require pre-declared read/write set, often infeasible*
- Amortise costs;
  - Combine replication and commitment protocols [OceanVista, MDCC, Janus, Helios]
  - **Epoch-based commit [COCO, STAR, Obladi]**

Epoch-based Commit

- **Paper: “***Epoch-based commit and replication in distributed OLTP databases”*, Lu et al., VLDB, 2021
- **Observation:** Failures are rare on modern hardware so invoking 2PC per transaction is coordination overkill
- **Key Idea:** Execute 2PC once for all transactions that arrive and processed within a time interval called an epoch → amortize cost of one 2PC over many transactions

Epoch-based Commit Cycle



WORK PHASE

- Nodes execute transactions

PREPARE PHASE

- Epoch timeouts
- Send PREPARE to nodes
- Receive PREPARE
- Force log *prepared record*:
  - Epoch #
  - All TIDs in epoch
  - All writes of TIDs
- Send P-ACK

COMMIT PHASE

- Receive P-ACKs
- Force log *commit record* (epoch #) and increment
- Send COMMIT
- Receive COMMIT
- Release results to clients
- Send COMMIT-ACK

Epoch-based Commit Summary

- **Advantages;**
  - High throughput (4x increase, Lu et al.)
  - 2PC per-transaction is overkill
- **Disadvantages;**
  - Increased latency per transaction
  - Sensitive to imbalanced workloads, 1 long-running transaction can block the complete epoch
  - All transactions executed in an epoch aborted if 1 nodes fails (wasted work)

For a given workload, how do we determine the optimal epoch size? 🤔

Contribution #1: analytical solutions for estimating throughput and average latency

Contribution #1: Analytical Models

- Derive analytical solutions in terms of epoch length and system/load parameters;
  - Maximum attainable throughput → assumes node never idle
  - Average latency → assumes transactions arrive at some rate into an external queue;
    - Upper bound & lower bound estimates
- Allow epoch length to be chosen for maximum throughput or minimum latency or seeking a trade-off between the two
- Makes few assumptions, e.g., failed node recovers before another node fails
- See paper for further details…

Epoch-based Commit Summary

- **Advantages;**
  - High throughput (4x increase, Lu et al.)
  - Performs well if all transactions are short-lived
  - 2PC per-transaction is overkill
- **Disadvantages;**
  - Increased latency per transaction
  - Sensitive to imbalanced workloads, 1 long-running transaction can block the complete epoch
  - All transactions executed in an epoch aborted if 1 nodes fails (wasted work)

For a given workload, how do we determine the optimal epoch size? 🤔

Contribution #1: analytical solutions for estimating throughput and average latency

Overly pessimistic? 🤔
Contribution #2: epoch-based multi-commit

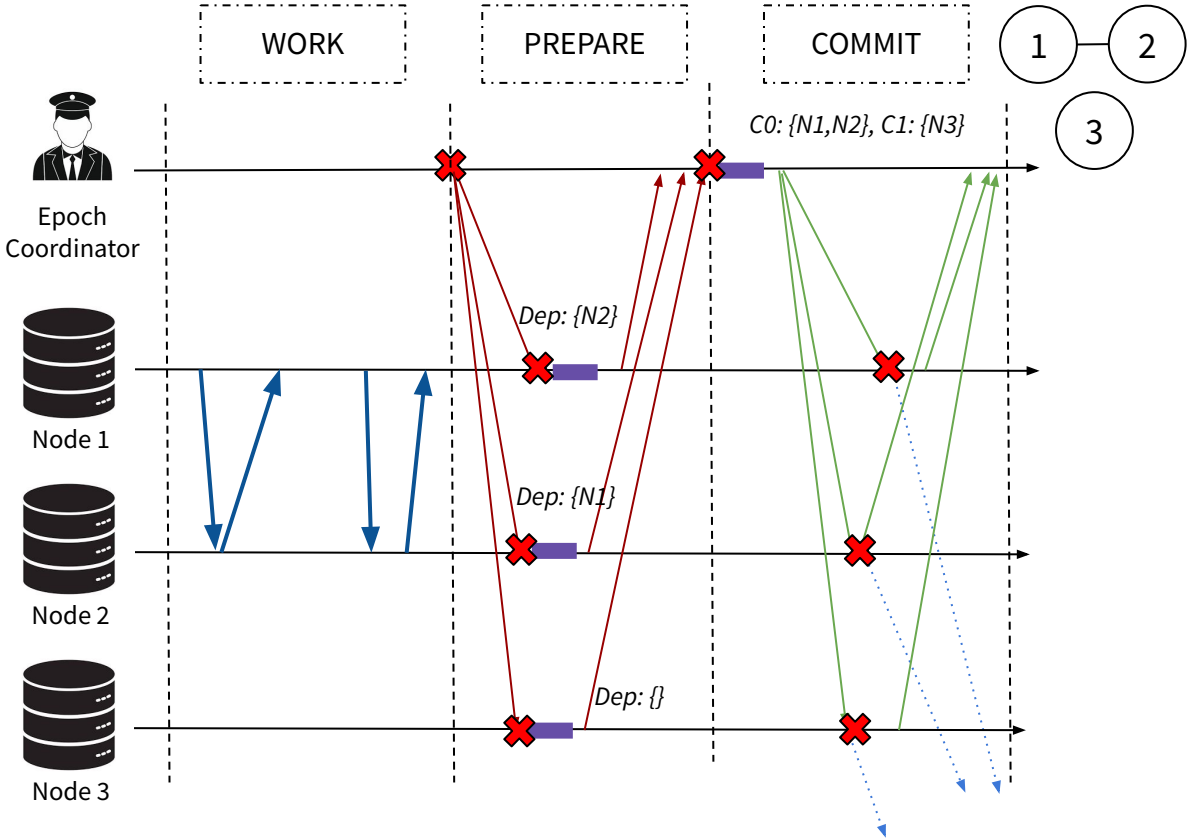
Contribution #2: Minimising Wasted Work

- **Epoch-based commit:** all transactions executed in an epoch abort if 1 nodes fails
- **Assumption:** each node has directly/indirectly accessed the failed node during the epoch → all transactions it executed accessed some data now lost due to failure **(pessimistic)**
- **Motivating example:**
  - Node fails shortly after it starts its work interval
  - Unlikely that each operative node executes a distributed transaction uses uncommitted data held by this node that failed
- **Epoch-based multi-commit:** avoids (where possible) aborting all transactions, thus improving throughput and reduce average latency

Epoch-based Multi-Commit

- Monitor node interactions called **epoch dependencies**
  - If T1 coordinated by N1 requires data stored on N2 then there will be an epoch dependency between N1 and N2
- Epoch dependencies can be direct or transitive
  - T1 coordinated by N1 updates an item, T2 coordinated by N2 reads update by T1 on N1, then updates item on N2, T3 coordinated by N3 reads update by T2 on N2
  - Information flow: N1 → N2 → N3
  - If N1 crashes, N2 and N3 must abort their transactions
- Epoch coordinator uses epoch dependencies to create **commit groups**
  - Form a undirected graph; vertices are database nodes and edges are epoch dependencies
  - Computes commit groups using connected components
- Commit groups are disjoint sets of nodes that can fail independently (no data dependencies)

Epoch-based Multi-Commit Cycle



WORK PHASE

- Nodes execute transactions

PREPARE PHASE

- Epoch timeouts
- Send PREPARE to nodes
- Receive PREPARE
- Force log *prepared record*:
  - Epoch #
- Receive P-ACKs
- Force log *commit record* (epoch #) and increment
- **Compute commit groups**
- Send COMMIT
- Receive COMMIT
- Release results to clients
- Send COMMIT-ACK

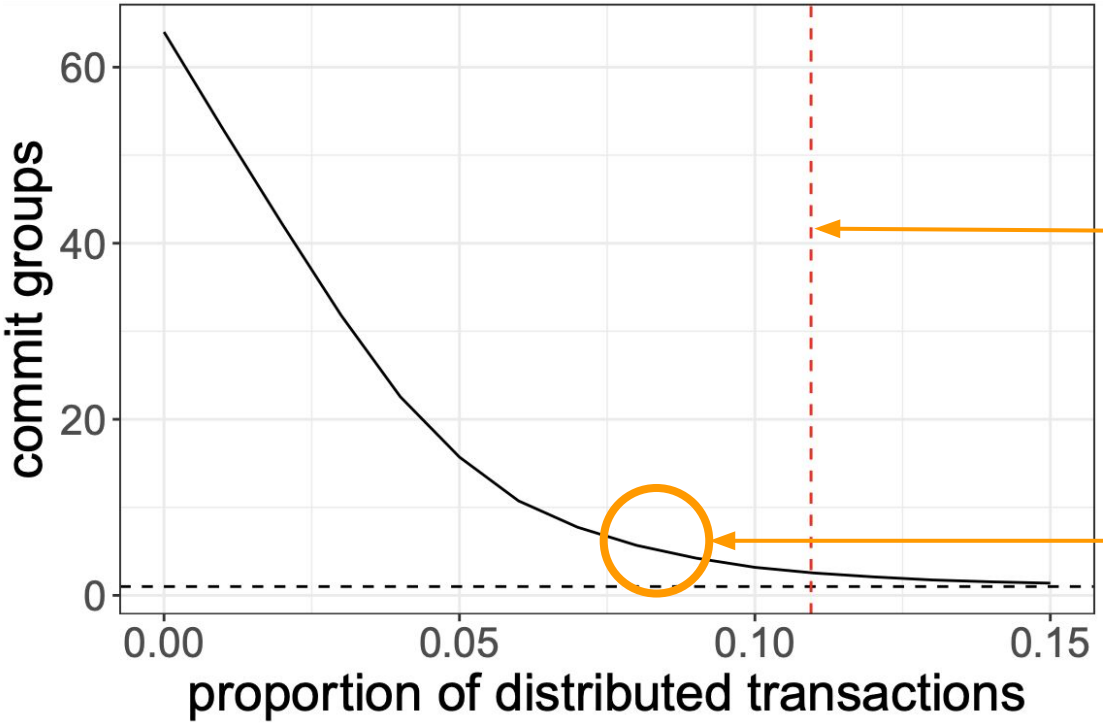
How useful is this? Would we always just end up with a single commit group 🤔

TPC-C Case Study

- TPC-C is the canonical OLTP database benchmark modeling warehouse order-processing application
- 5 transaction types, only *Payment* and *NewOrder* involve accessing remote nodes
- To minimise aborts multiple commit groups should emerge
- Simulated TPC-C workload varying the proportion of distributed transactions and counted commit groups formed;
  - Simulated 10K epochs
  - Fixed epoch size to 10ms
  - 300K transaction per second throughput
  - 64 server cluster

TPC-C Case Study cont.

Epoch size fixed to 10ms
300K TPS throughput
64 server cluster



After red line only single commit group

Commit groups are numerous when the proportion does not exceed 8%

Performance Evaluation

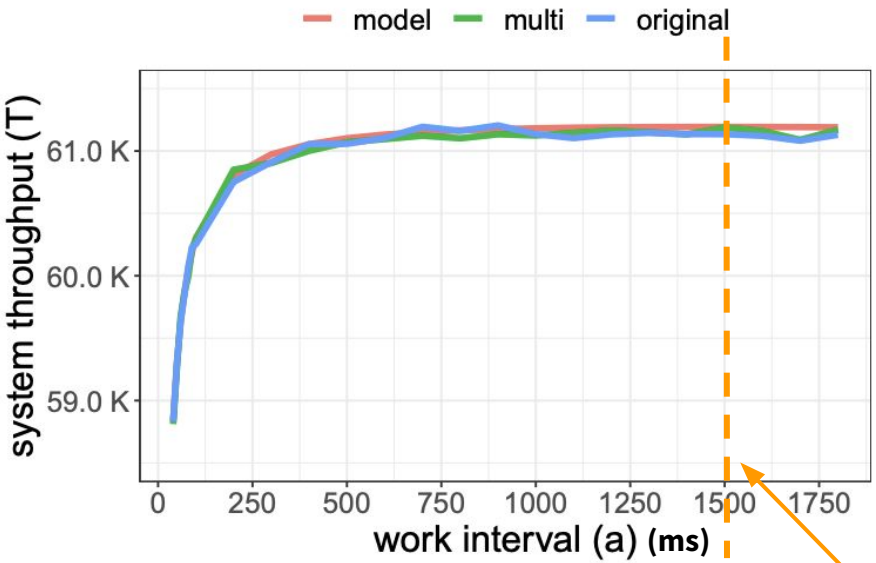
- **Discrete event-based simulations aims:**
  - Assess analytical models accuracy
  - Explore circumstances when multi-commit can outperform epoch-based commit
- Each simulation run took approximately 12 hours to complete ≈ 100 day operational period ≈ thousands of cycles with node failures; a = 40 ms, 11K/207M cycles contained failure
- **Metrics:**
  - *System throughput:* #transactions committed per second
  - *Average response time:* measured from the point when a transaction enters the system, to the point when it departs, after potentially several retries
  - *Committed transactions during failures:* average #transactions committed in cycles with failures
  - *Operational commit groups:* #commit groups not containing a failed node, given failure occurred

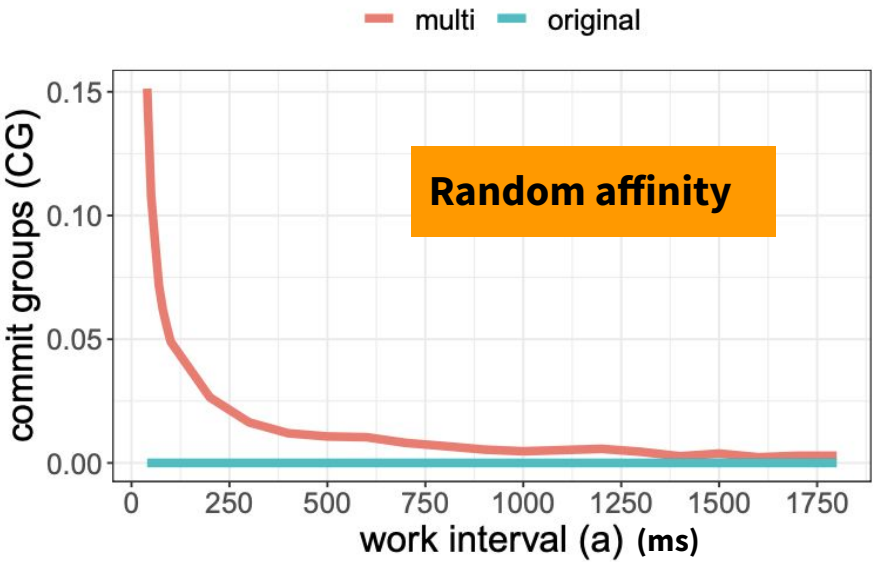
Parameters

Symbol	Meaning	Values
N	Number of participant nodes	64
a	Work interval (ms)	4-1800
b	Mean time to commit (ms)	1.7
μ	Node's transaction service rate (txn/ms)	1
$1/\xi$	Mean time between failure (hr)	12
$1/\eta$	Mean time to repair (min)	30
$E(\kappa)$	Remote servers accessed by transactions	0.1
$\lambda^{\dagger}$	Transaction arrival rate (txn/s)	30000

† Average response time model only.

Experiment #1: Maximum Throughput



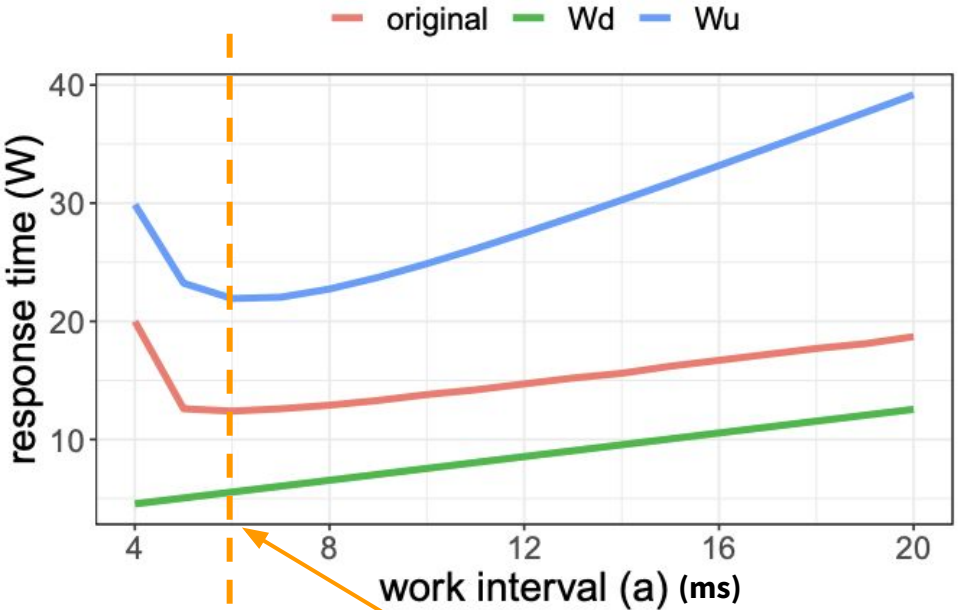


#1 Estimated throughput close to simulated values (max 3.8%)

#2 Model identifies a_T that achieves max throughput

#3 Throughput of both protocols are nearly identical

Experiment #2: Average Response Time



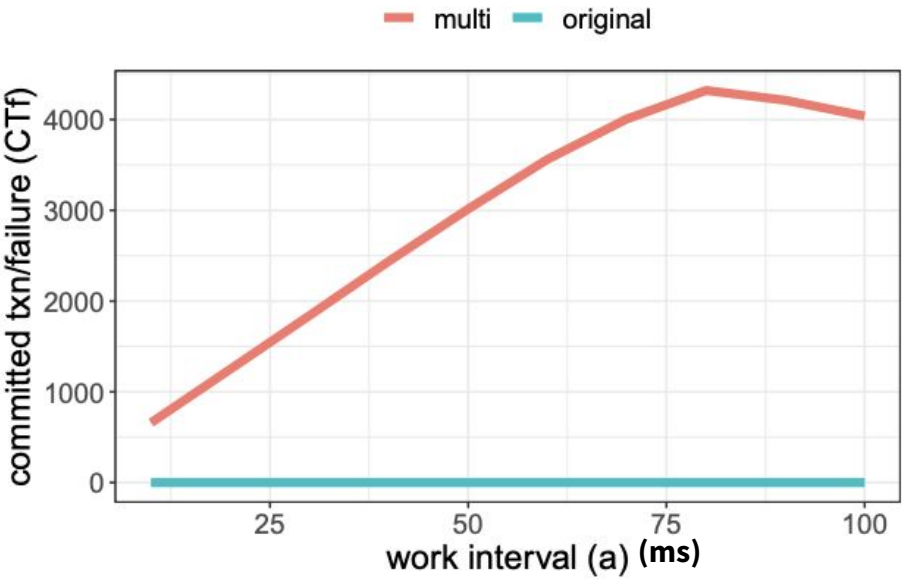
#1 Simulations well within upper/lower bound estimates

#2 Wu identifies a* = 6 reasonably accurately

#3 Wd response times closer to simulation (max 6ms)

Experiment #3: Paired Affinity

- Paired affinity attempts to capture correlations in data access
- Example; data on N1 and N2 is frequently accessed together more than N1 and N3
- Under paired affinity, distributed transactions accesses pair 90% of the time



#1 Multi-commit avoids a significant number of aborts

Conclusion

- Developed 2 analytical models which allow DBAs to explore trade-off between throughput and latency
- Model accuracy validated through simulations of 64 node cluster over 100 days
- Developed epoch-based multi-commit aiming to minimize transaction aborts in the event of node failures;
  - Performs better than original protocol when distributed transactions originating at a node tend to access specific other nodes in their remote interactions
  - Performs identically to the original protocol under other circumstances
- **We offer alternative to epoch-based commit and analytical solutions to efficiently tune the parameter of epoch-based commit protocols in practical settings**

Thanks for listening

Check out my podcast!

- Email: j.waudby2@newcastle.ac.uk
- Twitter: jwaudberry
- LinkedIn: jack-waudby