Wait-Hit: A high-performance concurrency control protocol for any scale

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

2 Dec 2021

1. Newcastle University
2. Neo4j

Database Concurrency Control

- Vital component for maintaining data integrity and achieving high performance
- Well researched, 3 main categories;
  - Lock-based, 2PL
  - Timestamp-based, TO/MVCC
  - Validation-based, OCC
- Point 1: scale poorly in many-core and shared-nothing DBMSs
- Point 2: users desire seamlessly scaling and to avoid re-architecting their systems
- Motivates the development of a protocol that performs well across multiple scale points

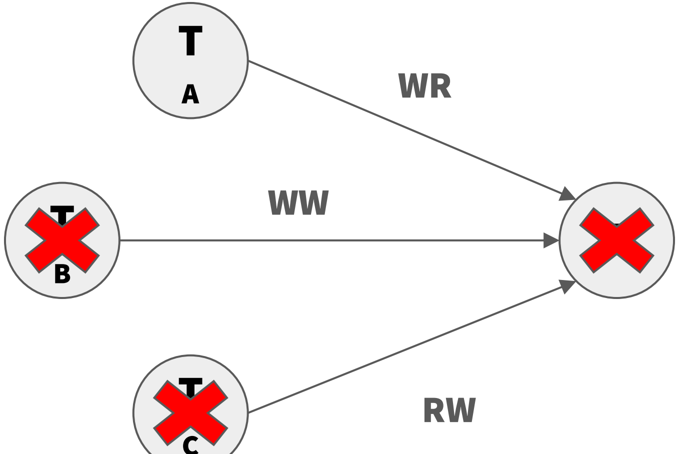
Serialization Graph Testing (SGT)

- **State-of-the-art protocol → Serialization Graph Testing**
  - ***"No False Negatives: Accepting All Useful Schedules in a Fast Serializable Many-Core System", Durner and Neumann, ICDE, 2019***
- **Theory: an execution of transactions is serializable iff its corresponding conflict graph is acyclic;**
  - **Two transactions conflict if both access the same data and at least 1 is write**
- **Protocol;**
  - **Transactions execute, annotating records with access metadata**
  - **Based on this detect conflicts and add edges to the serialization graph**
  - **At commit time, check if committing keeps the graph acyclic**
- **Benefit: best theoretical properties of accepting all valid schedules**
- **Summary: historically SGT deemed infeasible due to cycle checking, this paper refutes this in the context of a many-core system outperforming classical and modern protocols**

Wait-Hit Protocol

- **Problem: SGT cycle checking becomes infeasible again in distributed shared-nothing system**
- **How can we simplify cycle checking?**
  - **If there is no incoming edge there can not be a cycle**

T wants to commit...



Wait Phase, if T_A is;
- **Committed then continue**
- **Aborted then abort T**
- **Active then abort T**

Hit Phase:
- **Abort T_B and T_C**
- **Add T_B, T_C to Hit List**

Wait-Hit Protocol

- Ensures conflict serializability → enforces no incoming edges from active/aborted transactions via aborting self or predecessor
- Optimistic approach;
  - Transaction collects conflicting predecessors
  - Two-phased validation: wait phase and hit phase
  - Transaction commits/aborts
- 3 variants;
  - Basic
  - Optimised (many-core)
  - Distributed (shared nothing)

Basic Algorithm

3 data structures;

- **Hit list (HL); list of transactions that if allowed to commit may result in non-serializable behaviour**
- **Terminated list (TL); list of completed (aborted or committed) transactions**
- **ID generator (ID); generates unique IDs**

2 per-transaction data structures;

- **Predecessor upon read list (PR); when reading store id of transaction that wrote that value - can include transactions that have made uncommitted writes (at the time of reading)**
- **Predecessor upon write list (PW); when writing stores the id of all transactions that wrote and read the record before it**

Basic Algorithm cont.

- **Initialisation; assign ID and initialise PR/PW**
- **Execution; execute reads and writes, detecting conflicts, and inserting into PR/PW**
- **Commit Procedure for T;**
  - **Wait phase; for each *p* in PR;**
    - **If *p* is committed; continue**
    - **If *p* is aborted; abort T, append to TL, remove from HL (if exists)**
    - **Else *p* is active; employ zero-wait policy and abort T, append to TL, remove from HL (if exists)**
  - **Hit phase;**
    - **If T is in HL; abort T, append to TL, remove from HL**
    - **Else, commit T; merge PW into HL and append T to TL**
- **Epoch-based garbage collector ensures that TL does not grow over time**

Optimised Algorithm

Assume *n* cores each with a thread pinned to it, each has 2 thread-local data structures;

- **ID generator (ID); generates unique IDs (sequence number + thread id)**
- **Termination list (TL); list of transactions executed by this thread along with its state (active, aborted, or committed)**

Additionally, each thread has 2 per-transaction data structures;

- **Predecessor upon read list (PR); when reading store id of transaction that wrote that value - can include transactions that have made uncommitted writes (at the time of reading)**
- **Predecessor upon write list (PW); when writing stores the id of all transactions that wrote and read the record before it**

Optimised Algorithm cont.

- **Initialisation; thread receives T assigns ID and initialises PR/PW**
- **Execution; execute reads and writes, detecting conflicts, and inserting into PR/PW**
- **Commit Procedure for T;**
  - **If T has been hit; then abort T**
  - **While T is active; for each *p* in PW**
    - **If *p* is terminated; then continue**
    - **Else *p* is active; so hit *p***
  - **If T has been hit; then abort T**
  - **While T is active; for each *p* in PR**
    - **If *p* is committed; then continue**
    - **Else; abort T**
  - **Commit T**
- **Epoch-based garbage collector ensures TL on a thread does not grow over**

Distributed Algorithm - System Model

- Database consists of S shards
- Each shard S has T threads split into disjoint sets:
  - $\mathrm{T_H}$ coordinates **home transactions**; transactions that begin locally
  - $\mathrm{T_R}$ handles transactions that begin at a different server but operate on local data
  - $\mathrm{T_R}$ is managed by a **surrogate** process G
- Each transaction has a unique home shard (**coordinator**) and 0 or more remote shards (**validating shards**)

Distributed Algorithm - Data Structures

- **At shard *S*,**
  - **$\forall \tau \in \mathrm{T}_H$;**
    - **Transaction ID generator; [shard id, thread id, sequence number]**
    - **Terminated list; indexed by transaction ID**
    - **PuR/W lists for each transaction; storing local conflicts**
  - **Surrogate *G*;**
    - **Thread pool; containing T_R**
    - **Remote transaction status; the shard's local view of transaction termination status**
    - **PuR/W lists for each remote transaction; storing local conflicts**

Distributed Wait-Hit Protocol Context

1. **Initialisation;**
    a. **Transaction is assigned a unique ID and data structures are initialised**
2. **Execution;**
    a. **Transaction optimistically execute and PuR/W lists are populated**
3. **Commitment;**
    a. **Preparation**
    b. **Verification**
    c. **Commit**

Distributed Algorithm - Initialisation

- Coordinator (shard S_i);
  - Receives BEGIN_TRANSACTION
  - Assign to some $\tau_i \in T_H$
  - Assign ID = [S_i,τ_i,i], set TL(i) = 0, and initialise PUR/W(i)
  - Sends REMOTE_TRANSACTION(ID,operations) to validating shards
- Validating shards (shard S_j);
  - Receives REMOTE_TRANSACTION(ID,operations)
  - Surrogate G_j,
    - Inserts [ID,0] into its remote map and initialises PUR/W(ID)
    - Assigns a thread τ_j from T_R to execute operations

Distributed Algorithm - Execution

- Coordinator (shard S_i);
  - τ_i executes operations on local data and updates local PuR/W(i)
  - Receives **REMOTE_RESULTS(ID)** from validating shards
- Validating shards (shard S_j);
  - τ_j executes operations on local data and updates local PuR/W(ID)
  - Send **REMOTE_RESULTS(ID)** to coordinator

Distributed Algorithm - Commitment (Preparation)

- **Coordinator;**
  - **Send GET_READY(T_i) to all validating shards**
  - **While TL(i) ≠ -1 ∨ PuW(i) ≠ ∅; for each T_j ∈ PuW(i);**
    - **If TL(j) = 0; then set TL(j) = -1**
    - **Else; remove TL(j) from PuW(i)**
  - **If TL(i) ≠ -1; then wait for READY(T_i) from each validating shard**
  - **Else; send ABORT(T_i) to each validating shard**
- **Validating shards;**
  - **Receives GET_READY(T_i) from coordinator**
  - **While TL(i) ≠ -1 ∨ PuW(i) ≠ ∅; for each T_j ∈ PuW(i);**
    - **If TL(j) = 0; then set TL(j) = -1**
    - **Else; remove TL(j) from PuW(i)**
  - **If TL(i) ≠ -1; then send READY(T_i) to coordinator**
  - **Else; send ABORT(T_i) to coordinator**

Distributed Algorithm - Commitment (Verification)

- Coordinator;
  - Receives **READY(T_i)** from all validating shards
  - Sends **VERIFY(T_i)** to validating shards
  - While TL(i) ≠ -1 ∨ PuR(i) ≠ ∅; for each T_j ∈ PuR(i);
    - If TL(j) = 1; then remove TL(j) from PuR(i)
    - Else; set TL(j) = -1
  - If TL(i) ≠ -1; then wait for **VERIFIED(T_i)** from each validating shard
  - Else; send **ABORT(T_i)** to each validating shard
- Validating shards;
  - Receives **VERIFY(T_i)** from coordinator
  - While TL(i) ≠ -1 ∨ PuR(i) ≠ ∅; for each T_j ∈ PuR(i);
    - If TL(j) = 1; then remove TL(j) from PuR(i)
    - Else; set TL(j) = -1
  - If TL(i) ≠ -1; then send **VERIFIED(T_i)** to coordinator
  - Else; send **ABORT(T_i)** to coordinator

Distributed Algorithm - Commitment (Commit)

- Coordinator;
  - Receives **VERIFIED(T_i)** from all validating shards
  - If TL(i) ≠ -1; set TL(i) = 1 and send **COMMIT(T_i)** to each validating shard
  - Else; send **ABORT(T_i)** to each validating shard
- Validating shards;
  - Receives **COMMIT(T_i)** from coordinator
  - Receives **ABORT(T_i)** to coordinator

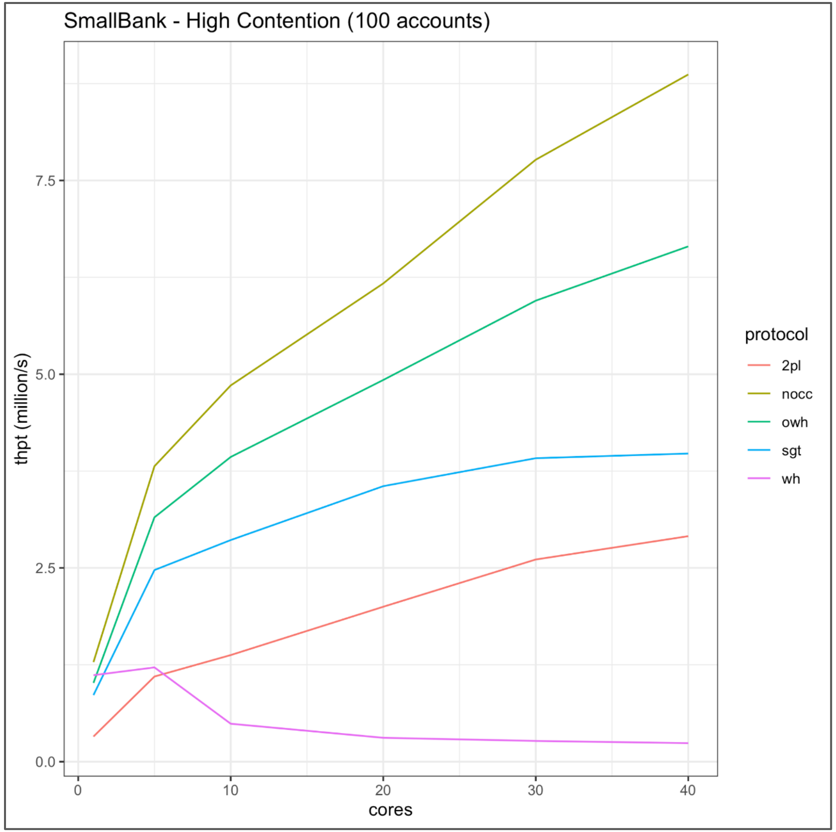
Evaluation Framework

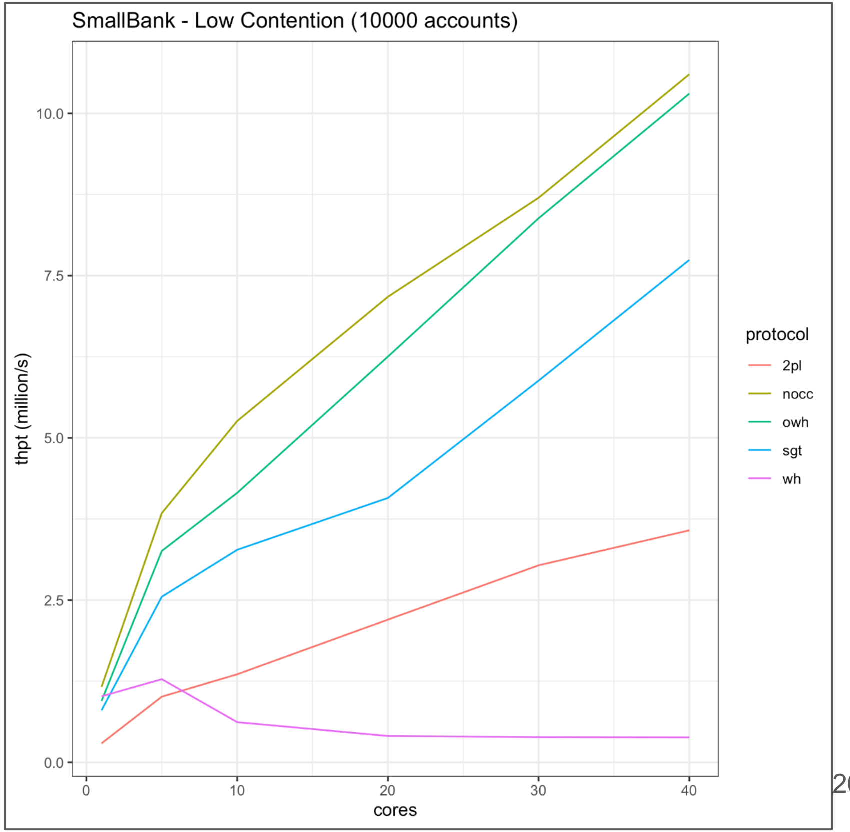
- Key components;
  - In-memory single versioned storage layer
  - Modular transaction scheduler
  - Extendable for multiple workloads; parameter generator, loader, and stored procedures
  - Each core acts as independent client generating transactions
- Testing; validated using LDBC’s property-based ACID test suite
- Workloads; SmallBank, TATP
- Metrics; throughput, av. latency, abort rate

Evaluation Framework cont.

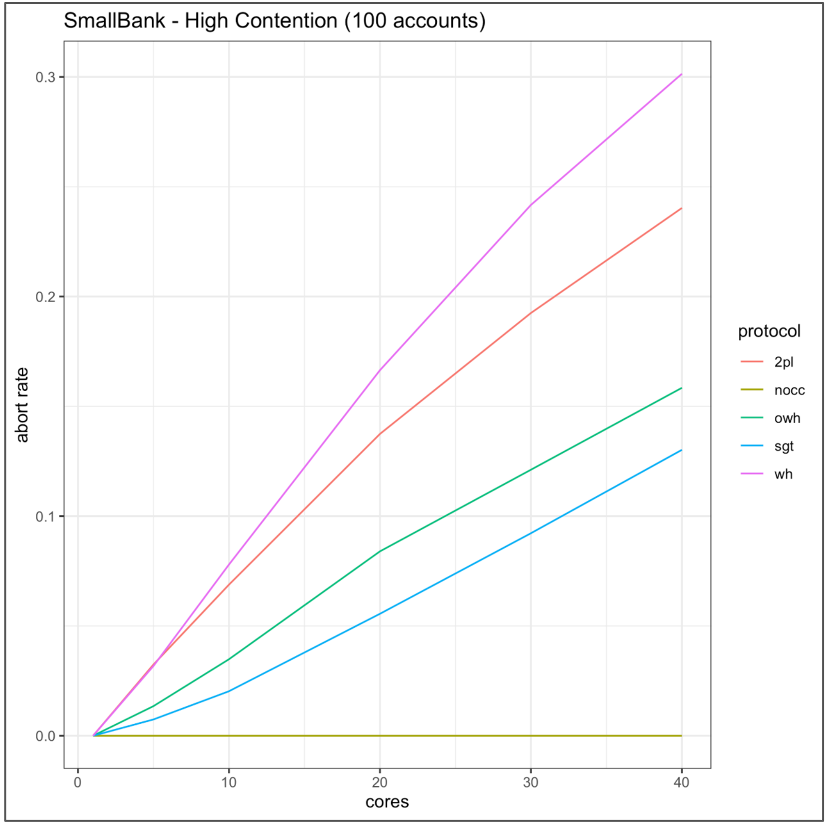
- **Protocols;**
  - **2PL: single-versioned, strict (locks held until commit point), read/write locks (no predicate locks)**
  - **SGT: faithful attempt to implement that described in Durner et al (2019)**
  - **WH: Wait-hit protocol with epoch-based garbage collector**
  - **OWH: Optimised wait-hit protocol with epoch-based garbage collector**
  - **NOCC: No concurrency control**
- **Hardware: Azure Standard_D48_v3 instance with 48 cores and 192GB RAM**

Throughput

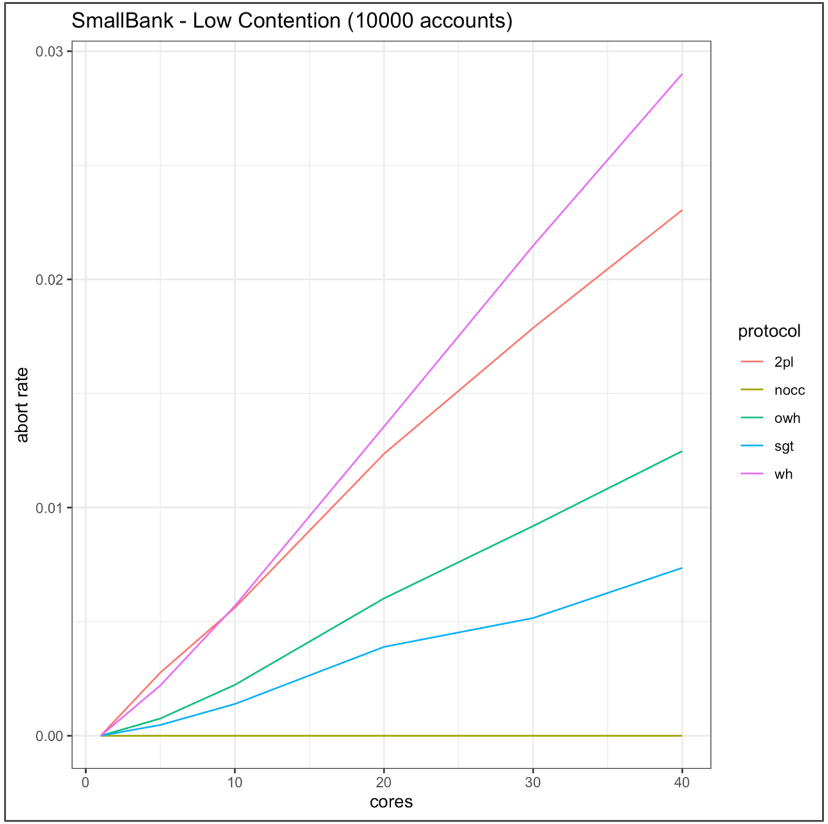

SmallBank - High Contention (100 accounts)
thpt (million/s)
0.0
2.5
5.0
7.5
0
10
20
30
40
cores
protocol
2pl
nocc
owh
sgt
wh


SmallBank - Low Contention (10000 accounts)
thpt (million/s)
0.0
2.5
5.0
7.5
10.0
0
10
20
30
40
cores
protocol
2pl
nocc
owh
sgt
wh

Abort Rate

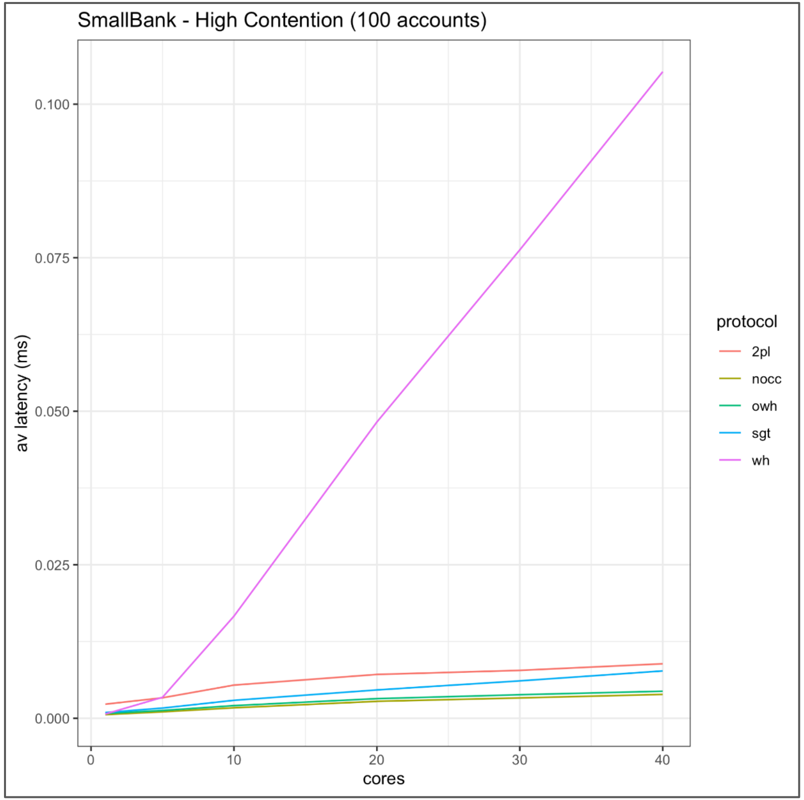


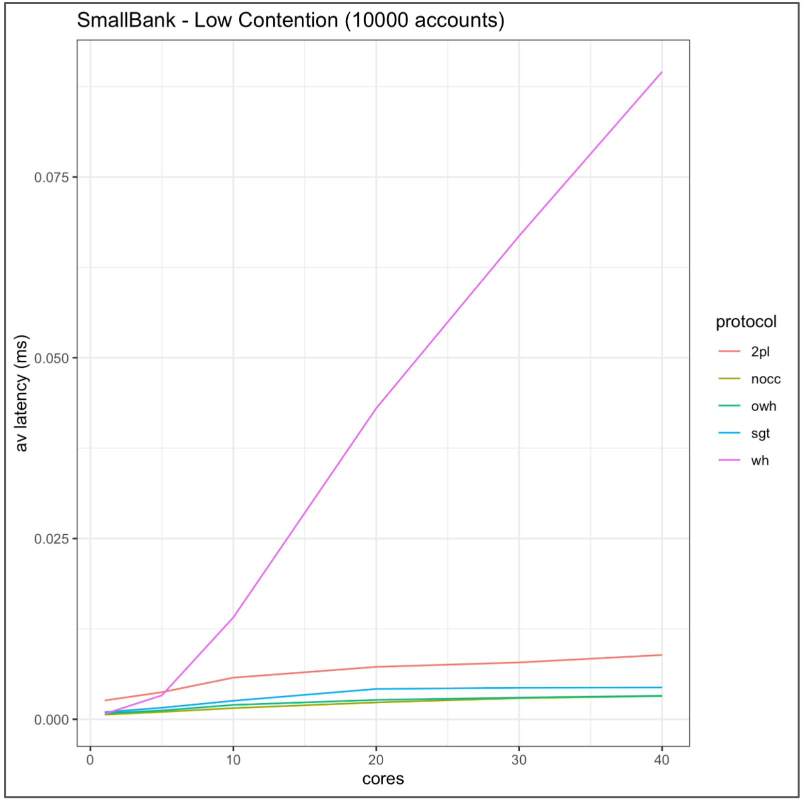
SmallBank - High Contention (100 accounts)
abort rate
0.0
0.1
0.2
0.3
0
10
20
30
40
cores
protocol
2pl
nocc
owh
sgt
wh



SmallBank - Low Contention (10000 accounts)
abort rate
0.00
0.01
0.02
0.03
0
10
20
30
40
cores
protocol
2pl
nocc
owh
sgt
wh

Average Latency





Future Work

- Extend framework to evaluate performance in a distributed shared-nothing setting
- Investigate techniques to amortise 2PC costs;
  - Epoch-based commit [COCO]
  - Parallel commits [CockroachDB]
  - Determinism [Calvin]
- Investigate how to make the protocol Neo4j friendly 23
- Proof of correctness (first order logic)

Ω