Database Replication in Tashkent

CSEP 545 Transaction Processing
Sameh Elnikety
Replication for Performance

Expensive
Limited scalability
DB Replication is Challenging

• Single database system
  – Large, persistent state
  – Transactions
  – Complex software

• Replication challenges
  – Maintain consistency
  – Middleware replication
Background

Standalone DBMS
Background

Load Balancer

Replica 1

Replica 2

Replica 3
Read Tx

Read tx does not change DB state
Update Tx 1/2

Load Balancer

- Replica 1
- Replica 2
  - ws
- Replica 3

Update tx
changes
DB state
Update Tx 1/2

Apply (or commit) T everywhere

T

Load Balancer

Example:
T1: { set x = 1 }
Update Tx 2/2

Load Balancer

Replica 1

Update tx changes DB state

Replica 2

Ordering

Replica 3
Commit updates in order

Replica 1
- Update tx changes DB state

Replica 2
- Example:
  - T1: \{ set x = 1 \}
  - T2: \{ set x = 7 \}

Replica 3

Load Balancer

Ordering
- ws
- ws
- ws
- ws
Sub-linear Scalability Wall
This Talk

• General scaling techniques
  – Address fundamental bottlenecks
  – Synergistic, implemented in middleware
  – Evaluated experimentally
Super-linear Scalability

Single: 1 X
Base: 7 X
United: 12 X
MALB: 25 X
UF: 37 X
Big Picture: Let’s Oversimplify

Standalone DBMS

- reading
- update
- logging

R
U
Big Picture: Let’s Oversimplify

- Standalone DBMS:
  - Reading
  - Update
  - Logging

- Replica 1/N (traditional):
  - Reading
  - Update
  - Logging

Tasks flow as follows:
- R (read) to Standalone DBMS
- U (update) to Standalone DBMS
- N.R (read) to Replica 1/N
- N.U (update) to Replica 1/N
Big Picture: Let’s Oversimplify

Replica 1/N (traditional)

Replica 1/N (optimized)

Standalone DBMS
Big Picture: Let’s Oversimplify

- **Standalone DBMS**
  - Reading
  - Update
  - Logging

- **Replica 1/N (traditional)**
  - Reading
  - Update
  - Logging
  - \(N-R\) \(->\) \(R\), \(N-U\) \(->\) \(U\), \((N-1).ws\)

- **Replica 1/N (optimized)**
  - Reading
  - Update
  - Logging
  - \(N-R\) \(->\) \(R^*\), \(N-U\) \(->\) \(U^*\), \((N-1).ws^*\)

**MALB**
Update Filtering
Uniting O & D
Key Points

1. Commit updates in order
   – Perform serial synchronous disk writes
   – Unite ordering and durability

2. Load balancing
   – Optimize for equal load: memory contention
   – MALB: optimize for in-memory execution

3. Update propagation
   – Propagate updates everywhere
   – Update filtering: propagate to where needed
Roadmap

Load Balancer

Replica 1

Replica 2

Replica 3

Ordering

Load balancing

Update propagation

Commit updates in order
Key Idea

• Traditionally:
  – Commit ordering and durability are separated

• Key idea:
  – Unite commit ordering and durability
All Replicas Must Agree

- All replicas agree on
  - which update tx commit
  - their commit order

- Total order
  - Determined by middleware
  - Followed by each replica
Order Outside DBMS

Replica 1

Replica 2

Replica 3

durability

durability

durability

Ordering
Order Outside DBMS

Tx A

Replica 1

durability
A → B

A → B

Tx B

Replica 2

durability
A → B

A → B

Tx B

Replica 3

durability
A → B

A → B

Ordering

A → B
Enforce External Commit Order

Ordering
A \rightarrow B

Replica 3

Proxy

SQL interface

durability

DBMS

Task A

Task B

Tx A

Tx B
Enforce External Commit Order

Ordering
A → B

Replica 3

Proxy

DBMS

durability
B → A

Task A

Task B

SQL interface

Tx A

Tx B
Enforce External Commit Order

Cannot commit A & B concurrently!
Enforce Order = Serial Commit

Ordering
A → B

Replica 3

Proxy

DBMS

SQL interface

durability

Task A

Task B

Tx A

Tx B
Enforce Order = Serial Commit

Ordering
A \rightarrow B

Replica 3

Proxy

Tx A

Tx B

Task A

Task B

durability
A \rightarrow B

DBMS

SQL interface
Commit Serialization is Slow

Ordering
A → B → C

Proxy

DBMS
durability

Commit order
A → B → C

CPU

Durability
A

Ack A

Commit A

CPU

Durability
A → B

Commit B

Ack B

CPU

Durability
A → B → C

Commit C

Ack C
Commit Serialization is Slow

Problem:
Durability & ordering separated → serial disk writes
Unite D. & O. in Middleware

Ordering
A → B → C

Durability
A → B → C

Commit order
A → B → C

Proxy

DBMS

durability OFF

CPU

Commit A

Ack A

Commit B

Ack B

Commit C

Ack C
Unite D. & O. in Middleware

Solution:
Move durability to MW
Durability & ordering in middleware → group commit
Implementation: Uniting D & O in MW

• Middleware logs tx effects
  – Durability of update tx
    • Guaranteed in middleware
    • Turn durability off at database

• Middleware performs durability & ordering
  – United → group commit → fast

• Database commits update tx serially
  – Commit = quick main memory operation
Uniting Improves Throughput

- **Metric**
  - Throughput

- **Workload**
  - TPC-W Ordering (50% updates)

- **System**
  - Linux cluster
  - PostgreSQL
  - 16 replicas
  - Serializable exec.
Roadmap

Load Balancer

Load balancing

Update propagation

Replica 1

Replica 2

Replica 3

Ordering

Commit updates in order
Key Idea

Equal load on replicas

Load Balancer

Replica 1

Mem

Disk

Replica 2

Mem

Disk
Key Idea

Equal load on replicas

Load Balancer

MALB: (Memory-Aware Load Balancing) Optimize for in-memory execution
How Does MALB Work?

Database

Workload

Memory

A → 1 2
B → 2 3
Read Data From Disk

Least Loaded

A → 1 2
B → 2 3

A, B, A, B

Replica 1

Mem

Disk

Replica 2

Mem

Disk
Read Data From Disk

A → 1 2
B → 2 3

Least Loaded

A, B, A, B

Replica 1
Mem
1 2 3
Slow
Disk
1 2 3

Replica 2
Mem
1 2 3
Slow
Disk
1 2 3
Data Fits in Memory

MALB

A → 1 2
B → 2 3

A, B, A, A, A
B, B, B, B

Replica 1
Mem
Disk
1 2 3

Replica 2
Mem
Disk
1 2 3
Data Fits in Memory

MALB

A → [1, 2]
B → [2, 3]

Memory info?
Many tx and replicas?

A, A, A, A

Fast

Replica 1
Mem
Disk

Replica 2
Mem
Disk

B
Estimate Tx Memory Needs

• Exploit tx execution plan
  – Which tables & indices are accessed
  – Their access pattern
    • Linear scan, direct access

• Metadata from database
  – Sizes of tables and indices
Grouping Transactions

• Objective
  – Construct tx groups that fit together in memory

• Bin packing
  – Item: tx memory needs
  – Bin: memory of replica
  – Heuristic: Best Fit Decreasing

• Allocate replicas to tx groups
  – Adjust for group loads
MALB in Action
MALB in Action

Memory needs for A, B, C, D, E, F

MALB
MALB in Action

Memory needs for A, B, C, D, E, F

Group A

Group B C

Group D E F
MALB in Action

Memory needs for A, B, C, D, E, F

MALB

Group A

Replica A
Disk

Group B C

Replica B C
Disk

Group D E F

Replica D E F
Disk
MALB Summary

• Objective
  – Optimize for in-memory execution

• Method
  – Estimate tx memory needs
  – Construct tx groups
  – Allocate replicas to tx groups
Experimental Evaluation

• Implementation
  – No change in consistency
  – Still middleware

• Compare
  – **United**: efficient baseline system
  – **MALB**: exploits working set information

• Same environment
  – Linux cluster running PostgreSQL
  – Workload: TPC-W Ordering (50% update txs)
MALB Doubles Throughput

TPC-W Ordering
16 replicas
MALB Doubles Throughput

105% increase

United

MALB

Read I/O, normalized

Single

Base

United

MALB

1 X

7 X

12 X

25 X
Big Gains with MALB

- Big DB Size: 12%, 75%, 182%
- Small DB Size: 29%, 0%, 4%

- Big Mem Size: 45%, 105%, 48%
- Small Mem Size: 12%
Big Gains with MALB

- Run from memory
- Run from disk

- 29%
- 45%
- 75%
- 105%
- 182%
- 48%
- 0%
Roadmap

- Load Balancer
- Load balancing
- Update propagation
- Replica 1
- Replica 2
- Replica 3
- Ordering
- Commit updates in order
Key Idea

• **Traditional:**
  – Propagate updates everywhere

• **Update Filtering:**
  – Propagate updates to where they are needed
Update Filtering Example

A → 1 2
B → 2 3

A, B, A, B

MALB UF

Replica 1
Mem
Disk 1 2 3

Replica 2
Mem
Disk 1 2 3
Update Filtering Example

A → 1 2
B → 2 3

MALB
UF

A, B, A, B

Group A

Replica 1
Mem
1 2
Disk
1 2 3

Group B

Replica 2
Mem
2 3
Disk
1 2 3
**Update Filtering Example**

Group A

A → 1 2
B → 2 3

Group B

MALB UF

A, B, A, B

Update table 1

Replica 1

Disk

Mem

Replica 2

Disk

Mem

59
Update Filtering Example

Group A

A → 1 2
B → 2 3

MALB UF

Group B

A, B, A, B

Update table 1

A → B → 2 1 2 3
Update Filtering Example

Group A

A → 1 2
B → 2 3

Update table 1

MALB UF

A, B, A, B

Group B

Replica 1

Disk

Replica 2

Mem

Mem
Update Filtering Example

Group A
A → 1 2
B → 2 3

Replica 1
Mem
1 2
Disk
1 2 3
Update table 1

Group B
Replica 2
Mem
2 3
Disk
1 2 3
Update table 3

MALB UF
A, B, A, B

A → B → 2 1 2

A, B, A, B
Update Filtering Example

Group A

A → 1 2
B → 2 3

Update table 1

Group B

A, B, A, B

Update table 3

MALB UF
Update Filtering Example

Group A

A → 1 2
B → 2 3

Update table 1

Group B

A, B, A, B

MALB UF

Update table 3
Update Filtering in Action
Update Filtering in Action

Update to red table
Update Filtering in Action

Update to red table

Update to green table

UF
Update Filtering in Action

Update to red table

Update to green table

UF
Update Filtering in Action

Update to red table

Update to green table
MALB+UF Triples Throughput

TPC-W Ordering 16 replicas
MALB+UF Triples Throughput

TPS

Prop. Updates

Single  Base  United  MALB  UF

1 X  7 X  12 X  25 X  37 X

MALB  UF

15  7

49%
Filtering Opportunities

5% Browsing Mix

50% Ordering Mix

Updates

Ratio MALB+UF / MALB

MALB    MALB+UF

1.49    1.02
0.5    1
1    1.5
1.5    2
2

MALB    MALB+UF

1.02    1.49
0    1
0.5    1.5
1    2
2

Percentage:
- 50% Ordering Mix
- 50% Filtering Opportunities
Conclusions

1. Commit updates in order
   – Perform serial synchronous disk writes
   – Unite ordering and durability

2. Load balancing
   – Optimize for equal load: memory contention
   – MALB: optimize for in-memory execution

3. Update propagation
   – Propagate updates everywhere
   – Update filtering: propagate to where needed