L-Store Concurrency Control: QueCC

Slides are adopted from Qadah, Sadoghi

QueCC - A Queue-Oriented, Control-Free Concurrency Architecture, ACM Middleware 2018

ECS 165A – Winter 2022

Mohammad Sadoghi
Exploratory Systems Lab
Department of Computer Science

UC DAVIS
UNIVERSITY OF CALIFORNIA

ExpoLab
Creativity Unfolded

ResilientDB
Hardware Trends

Large core counts

Large main-memory

HPE Superdome Server
144 physical cores
6TB of RAM

Popularity of Key-value Stores

- No multi-statement transactions
- Weak consistency
- Weak isolation
High-Contention Workloads

Challenge ???

High number of contented operations
State-of-the-Art Concurrency Control Protocols

- Optimized for multi-core hardware and main-memory databases
- Non-deterministic

<table>
<thead>
<tr>
<th>CC</th>
<th>Class</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>SILO</td>
<td>Optimistic CC</td>
<td>SOSP ’13</td>
</tr>
<tr>
<td>TICTOC</td>
<td>Timestamp Ordering</td>
<td>SIGMOD ’16</td>
</tr>
<tr>
<td>FOEDUS-MOCC</td>
<td>Optimistic CC</td>
<td>VLDB ’16</td>
</tr>
<tr>
<td>ERMIA</td>
<td>MVCC</td>
<td>SIGMOD ’16</td>
</tr>
<tr>
<td>Cicada</td>
<td>MVCC</td>
<td>SIGMOD ’17</td>
</tr>
</tbody>
</table>
Performance Under High-Contention

Optimize-for-multi-core concurrency control techniques suffer under high-contention due to increasing abort rate.
Performance Under High-Contention

Under high-contention: Non-deterministic aborts dominates
Performance Under High-Contention

Under high-contention: Non-deterministic aborts dominates
2PL - NoWait

Abort Count: 0

Client Transactions

- \( w_4(b) \)
- \( w_3(b) \)
- \( w_2(b) \)
- \( r_1(a) \)
- \( r_4(d) \)
- \( r_3(c) \)
- \( r_2(a) \)
- \( w_1(b) \)

Each color presents a transaction

Worker Thread #1

Worker Thread #2

Database:

- a
- b
- c
- d
Client Transactions

2PL - NoWait

Abort Count: 0

Worker Thread #1

Worker Thread #2

Worker Thread #1

Worker Thread #2

Client Transactions

w₄(b) w₃(b) r₄(d) r₃(c)

Worker Thread #1

r₁(a) w₁(b)

Worker Thread #2

w₂(b) r₂(a)

(1) w₁(b)

(2) w₁(b)

(3) r₁(a)

(4) w₁(b)

(5) w₁(b)

(6) w₁(b)

(7) w₁(b)

(8) w₁(b)

(9) w₁(b)

(10) w₁(b)

(11) w₁(b)

(12) w₁(b)

(13) w₁(b)

(14) w₁(b)

(15) w₁(b)

(16) w₁(b)

(17) w₁(b)

(18) w₁(b)

(19) w₁(b)

(20) w₁(b)

(21) w₁(b)

(22) w₁(b)

(23) w₁(b)

(24) w₁(b)

(25) w₁(b)

(26) w₁(b)

(27) w₁(b)

(28) w₁(b)

(29) w₁(b)

(30) w₁(b)

(31) w₁(b)

(32) w₁(b)

(33) w₁(b)

(34) w₁(b)

(35) w₁(b)

(36) w₁(b)

(37) w₁(b)

(38) w₁(b)

(39) w₁(b)

(40) w₁(b)

(41) w₁(b)

(42) w₁(b)

(43) w₁(b)

(44) w₁(b)

(45) w₁(b)

(46) w₁(b)

(47) w₁(b)

(48) w₁(b)

(49) w₁(b)

(50) w₁(b)

(51) w₁(b)

(52) w₁(b)

(53) w₁(b)

(54) w₁(b)

(55) w₁(b)

(56) w₁(b)

(57) w₁(b)

(58) w₁(b)

(59) w₁(b)

(60) w₁(b)

(61) w₁(b)

(62) w₁(b)

(63) w₁(b)

(64) w₁(b)

(65) w₁(b)

(66) w₁(b)

(67) w₁(b)

(68) w₁(b)

(69) w₁(b)

(70) w₁(b)

(71) w₁(b)

(72) w₁(b)

(73) w₁(b)

(74) w₁(b)

(75) w₁(b)

(76) w₁(b)

(77) w₁(b)

(78) w₁(b)

(79) w₁(b)

(80) w₁(b)

(81) w₁(b)

(82) w₁(b)

(83) w₁(b)

(84) w₁(b)

(85) w₁(b)

(86) w₁(b)

(87) w₁(b)

(88) w₁(b)

(89) w₁(b)

(90) w₁(b)

(91) w₁(b)

(92) w₁(b)

(93) w₁(b)

(94) w₁(b)

(95) w₁(b)

(96) w₁(b)

(97) w₁(b)

(98) w₁(b)

(99) w₁(b)

(100) w₁(b)

(101) w₁(b)

(102) w₁(b)

(103) w₁(b)

(104) w₁(b)

(105) w₁(b)

(106) w₁(b)

(107) w₁(b)

(108) w₁(b)

(109) w₁(b)

(110) w₁(b)

(111) w₁(b)

(112) w₁(b)

(113) w₁(b)

(114) w₁(b)

(115) w₁(b)

(116) w₁(b)

(117) w₁(b)

(118) w₁(b)

(119) w₁(b)

(120) w₁(b)

(121) w₁(b)

(122) w₁(b)

(123) w₁(b)

(124) w₁(b)

(125) w₁(b)

(126) w₁(b)

(127) w₁(b)

(128) w₁(b)

(129) w₁(b)

(130) w₁(b)

(131) w₁(b)

(132) w₁(b)

(133) w₁(b)

(134) w₁(b)

(135) w₁(b)

(136) w₁(b)

(137) w₁(b)

(138) w₁(b)

(139) w₁(b)

(140) w₁(b)

(141) w₁(b)

(142) w₁(b)

(143) w₁(b)

(144) w₁(b)

(145) w₁(b)

(146) w₁(b)

(147) w₁(b)

(148) w₁(b)

(149) w₁(b)

(150) w₁(b)

(151) w₁(b)

(152) w₁(b)

(153) w₁(b)

(154) w₁(b)

(155) w₁(b)

(156) w₁(b)

(157) w₁(b)

(158) w₁(b)

(159) w₁(b)

(160) w₁(b)

(161) w₁(b)

(162) w₁(b)

(163) w₁(b)

(164) w₁(b)

(165) w₁(b)

(166) w₁(b)

(167) w₁(b)

(168) w₁(b)

(169) w₁(b)

(170) w₁(b)

(171) w₁(b)

(172) w₁(b)

(173) w₁(b)

(174) w₁(b)

(175) w₁(b)

(176) w₁(b)

(177) w₁(b)

(178) w₁(b)

(179) w₁(b)

(180) w₁(b)

(181) w₁(b)

(182) w₁(b)

(183) w₁(b)

(184) w₁(b)

(185) w₁(b)

(186) w₁(b)

(187) w₁(b)

(188) w₁(b)

(189) w₁(b)
2PL - NoWait

Abort Count: 0

Client Transactions

\[
\begin{align*}
&\text{w}_4(b) & \text{w}_3(b) \\
&\text{r}_4(d) & \text{r}_3(c)
\end{align*}
\]

Worker Thread #1

\[
\begin{align*}
&\text{r}_1(a) \\
&\text{w}_1(b)
\end{align*}
\]

Worker Thread #2

\[
\begin{align*}
&\text{w}_2(b) \\
&\text{r}_2(a)
\end{align*}
\]

Database

\[
\begin{align*}
&\text{a} \\
&\text{b} \\
&\text{c} \\
&\text{d}
\end{align*}
\]
Client Transactions

- \( w_4(b) \)
- \( w_3(b) \)
- \( r_4(d) \)
- \( r_3(c) \)

Worker Thread #1

- \( w_1(b) \)
- \( r_1(a) \)

Worker Thread #2

- \( w_2(b) \)
- \( r_2(a) \)

2PL - NoWait

Abort Count: 0

Database:
- \( a \)
- \( b \)
- \( c \)
- \( d \)
2PL - NoWait

Abort Count: 0

Client Transactions

- w_4(b)
- w_3(b)
- r_4(d)
- r_3(c)

Worker Thread #1

- w_1(b)
- r_1(a)

Worker Thread #2

- w_2(b)
- r_2(a)

Conflict!
Abort transaction (to avoid potential deadlocks)

Client Transactions

- \( w_4(b) \)
- \( w_3(b) \)
- \( r_4(d) \)
- \( r_3(c) \)

Worker Thread #1

- \( w_1(b) \)
- \( r_1(a) \)

Worker Thread #2

- \( w_2(b) \)
- \( r_2(a) \)

Abort Count: 0
Client Transactions

Worker Thread #1

Worker Thread #2

Abort Count: 1

2PL - NoWait
2PL - NoWait

Abort Count: 1

Client Transactions

\[ w_4(b) \]
\[ r_4(d) \]
\[ r_1(a) \]
\[ w_1(b) \]

Worker Thread #1

\[ w_3(b) \]
\[ r_3(c) \]

Worker Thread #2

\[ w_2(b) \]
\[ r_2(a) \]
2PL - NoWait

Abort Count: 1

Client Transactions
- \(w_4(b)\)
- \(r_4(d)\)
- \(r_1(a)\)
- \(w_1(b)\)

Worker Thread #1
- \(w_3(b)\)
- \(r_3(c)\)

Worker Thread #2
- \(w_2(b)\)
- \(r_2(a)\)
2PL - NoWait

Abort Count: 1

Client Transactions

- \textit{w}_4(b)
- \textit{r}_4(d)
- \textit{r}_1(a)
- \textit{w}_1(b)

Worker Thread #1

- \textit{w}_3(b)
- \textit{r}_3(c)

Worker Thread #2

- \textit{w}_2(b)
- \textit{r}_2(a)

Conflict!
Client Transactions

- w4(b)
- r4(d)
- r1(a)
- w1(b)

Worker Thread #1

- w3(b)
- r3(c)

Worker Thread #2

- w2(b)
- r2(a)

2PL - NoWait

Abort Count: 1

Abort transaction (to avoid potential deadlocks)
2PL - NoWait

Abort Count: 2

Client Transactions

- `w_4(b)`
- `w_3(b)`
- `r_1(a)`
- `w_1(b)`
- `r_4(d)`
- `r_3(c)`

Worker Thread #1

Worker Thread #2

Database:

- `a`
- `b`
- `c`
- `d`
2PL - NoWait

Abort Count: 2

Client Transactions

- w₃(b)
- r₃(c)
- r₁(a)
- w₁(b)

Worker Thread #1

- w₄(b)
- r₄(d)

Worker Thread #2

- w₂(b)
- r₂(a)

Database:

- a
- b
- c
- d
2PL - NoWait

Abort Count: 2

Client Transactions

- $w_3(b)$
- $r_3(c)$
- $r_1(a)$
- $w_1(b)$

Worker Thread #1

- $w_4(b)$
- $r_4(d)$

Worker Thread #2

- $w_2(b)$
- $r_2(a)$

Data structure:

- a
- b
- c
- d
2PL - NoWait

Abort Count: 2

Client Transactions

- w3(b)
- r3(c)
- w1(b)
- r1(a)

Worker Thread #1

- w4(b)
- r4(d)

Worker Thread #2

- w2(b)
- r2(a)

Database:

- a
- b
- c
- d
2PL - NoWait

Abort Count: 2

Client Transactions

- $w_3(b)$
- $r_3(c)$
- $r_1(a)$
- $w_1(b)$

Worker Thread #1

- $r_4(d)$
- $w_4(b)$

Worker Thread #2

- $r_2(a)$
- $w_2(b)$

Conflict!
Abort transaction (to avoid potential deadlocks)

2PL - NoWait

Abort Count: 2

Client Transactions

w3(b)  r3(c)  r1(a)  w1(b)

Worker Thread #1

w4(b)  r4(d)

Worker Thread #2

w2(b)  r2(a)

Data

a
b
c
d
2PL - NoWait

Abort Count: 3

Worker Thread #1

Client Transactions
- w_4(b)
- w_3(b)
- r_1(a)
- r_4(d)
- r_3(c)
- w_1(b)

Worker Thread #2

Committed Transactions
- w_2(b)
- r_2(a)
2PL - NoWait

Abort Count: 3

Client Transactions
- \( r_1(a) \)
- \( w_1(b) \)

Worker Thread #1
- \( w_3(b) \)
- \( r_3(c) \)

Worker Thread #2
- \( w_4(b) \)
- \( r_4(d) \)

Committed Transactions
- \( w_2(b) \)
- \( r_2(a) \)
Client Transactions

- $r_1(a)$
- $w_1(b)$

Worker Thread #1
- $w_3(b)$
- $r_3(c)$

Worker Thread #2
- $w_4(b)$
- $r_4(d)$

Committed Transactions
- $w_2(b)$
- $r_2(a)$

Abort Count: 3

2PL - NoWait
2PL - NoWait

Abort Count: 3

Client Transactions

- \( r_1(a) \)
- \( w_1(b) \)

Worker Thread #1

- \( r_3(c) \)
- \( w_3(b) \)

Worker Thread #2

- \( r_4(d) \)
- \( w_4(b) \)

Committed Transactions

- \( w_2(b) \)
- \( r_2(a) \)

Conflict!
2PL - NoWait

Abort Count: 3

Client Transactions

- $r_1(a)$
- $w_1(b)$

Abort transaction (to avoid potential deadlocks)

Worker Thread #1
- $w_3(b)$
- $r_3(c)$

Worker Thread #2
- $w_4(b)$
- $r_4(d)$

Committed Transactions

- $w_2(b)$
- $r_2(a)$

Database:
- $a$
- $b$
- $c$
- $d$
2PL - NoWait

Abort Count: 4

Client Transactions
- w₄(b)
- r₄(d)
- r₁(a)
- w₁(b)

Committed Transactions
- w₂(b)
- r₂(a)

Worker Thread #1
- w₃(b)
- r₃(c)

Worker Thread #2
- w₃(b)
- r₃(c)

Resources
- a
- b
- c
- d
2PL - NoWait

Abort Count: 4

Client Transactions
- $w_4(b)$
- $r_4(d)$

Worker Thread #1
- $w_3(b)$
- $r_3(c)$

Worker Thread #2
- $r_1(a)$
- $w_1(b)$

Committed Transactions
- $w_2(b)$
- $r_2(a)$
2PL - NoWait

Abort Count: 4

Client Transactions
- \( w_4(b) \)
- \( r_4(d) \)

Worker Thread #1
- \( w_3(b) \)
- \( r_3(c) \)

Worker Thread #2
- \( r_1(a) \)
- \( w_1(b) \)

Committed Transactions
- \( w_2(b) \)
- \( r_2(a) \)
2PL - NoWait

Abort Count: 4

Client Transactions

- $w_4(b)$
- $r_4(d)$

Worker Thread #1

- $r_3(c)$
- $w_3(b)$

Worker Thread #2

- $r_1(a)$
- $w_1(b)$

Committed Transactions

- $w_2(b)$
- $r_2(a)$
Client Transactions

- \( w_4(b) \)
- \( r_4(d) \)

Worker Thread #1

- \( w_3(b) \)
- \( r_3(c) \)

Worker Thread #2

- \( r_1(a) \)
- \( w_1(b) \)

Committed Transactions

- \( w_2(b) \)
- \( r_2(a) \)

2PL - NoWait

Abort Count: 4
2PL - NoWait

Abort Count: 4

Client Transactions

- $w_4(b)$
- $r_4(d)$

Worker Thread #1

- $w_3(b)$
- $r_3(c)$

Worker Thread #2

- $r_1(a)$
- $w_1(b)$

Committed Transactions

- $w_2(b)$
- $r_2(a)$

Conflict!
2PL - NoWait

Abort Count: 4

Client Transactions
- \( w_4(b) \)
- \( r_4(d) \)

Worker Thread #1
- \( w_3(b) \)
- \( r_3(c) \)

Worker Thread #2
- \( r_1(a) \)
- \( w_1(b) \)

Abort transaction (to avoid potential deadlocks)

Committed Transactions
- \( w_2(b) \)
- \( r_2(a) \)
2PL - NoWait

Abort Count: 5

Client Transactions

- $w_4(b)$
- $r_4(d)$
- $r_1(a)$
- $w_1(b)$

Worker Thread #1

Committed Transactions

- $w_3(b)$
- $w_2(b)$
- $r_3(c)$
- $r_2(a)$
Worker Thread #1:  
- r_1(a)  
- w_1(b)

Worker Thread #2:  
- w_4(b)  
- r_4(d)

Client Transactions:

- a
- b
- c
- d

Committed Transactions:

- w_3(b)
- w_2(b)
- r_3(c)
- r_2(a)

2PL - NoWait

Abort Count: 5
Eventually transactions commit in some serial order!
Many aborts due to high contention on record b
Non-determinism in CC cause these aborts
Wasted work
Key Insights

• Many aborts due to high contention

• Non-determinism in CC cause these aborts

• Can we do better?

• Is it possible to eliminate non-deterministic concurrency control from transaction execution?
Deterministic Transaction Execution

- H-Store [Kallman et al. ’08]
- Designed and optimized for horizontal scalability, multi-core hardware and in-memory databases
- Stored procedure transaction model
- Static partitioning of database
- Assigns a single core to each partition
- Execute transaction serially without concurrency control within each partition
Client Transactions

- Single-partition transactions

H-Store

Abort Count: 0

Worker Thread #1

P1 is assigned to Worker Thread #1

Worker Thread #2

P2 is assigned to Worker Thread #2

Client Transactions:

- \( w_4(d) \)
- \( w_3(b) \)
- \( w_2(c) \)
- \( r_1(a) \)
- \( r_4(c) \)
- \( r_3(a) \)
- \( r_2(d) \)
- \( w_1(b) \)

P1:

- a
- b

P2:

- c
- d
H-Store

Abort Count: 0

Client Transactions

- \( w_4(d) \)
- \( w_3(b) \)
- \( r_4(c) \)
- \( r_3(a) \)

Worker Thread #1

- \( r_1(a) \)
- \( w_1(b) \)

Worker Thread #2

- \( w_2(c) \)
- \( r_2(d) \)

Committed Transactions

- P1: a, b, c
- P2: d
Client Transactions

Worker Thread #1
- \( w_3(b) \)
- \( r_3(a) \)

Worker Thread #2
- \( w_4(d) \)
- \( r_4(c) \)

H-Store
- Abort Count: 0

Committed Transactions
- \( w_2(c) \)
- \( r_1(a) \)
- \( r_2(d) \)
- \( w_1(b) \)
Committed Transactions

H-Store

Abort Count: 0

Client Transactions

Worker Thread #1

Worker Thread #2

Committed Transactions

w₄(d)  w₃(b)  w₂(c)  r₁(a)

r₄(c)  r₃(a)  r₂(d)  w₁(b)
Client Transactions

H-Store

Abort Count: 0

Worker Thread #1

Worker Thread #2

✓ Deterministic Execution
✓ No aborts because of CC
✓ Minimal coordination among threads

Performs well only when transactions are single-partitioned

Committed Transactions

- \(w_4(d)\)
- \(w_3(b)\)
- \(w_2(c)\)
- \(r_1(a)\)
- \(r_4(c)\)
- \(r_3(a)\)
- \(r_2(d)\)
- \(w_1(b)\)
Effect of Increasing Percentage of Multi-Partition Transactions in the Workload

H-Store is sensitive to the percentage of multi-partition transactions in the workload.
Can We Do Better?

Our motivations are

• Efficiently exploits **multi-core and large main-memory systems**
• Provide **serializable** multi-statement transactions for key-value stores
• Scales well under **high-contention** workloads

Desired Properties

• Concurrent execution over shared data
• Not limited to partitionable workloads
• Without any concurrency controls
Is it possible to have concurrent execution over shared data without having any concurrency controls?
Introducing: QueCC
Queue-Oriented, Control-Free, Concurrency Architecture

A two parallel & independent phases of priority-driven planning & execution

Phase 1: Deterministic priority-based planning of transaction operations in parallel

- Plans take the form of Prioritized Execution Queues
- Execution Queues inherits predetermined priority of its planner
- Results in a deterministic plan of execution

Phase 2: Priority driven execution of plans in parallel

- Satisfies the Execution Priority Invariance

“For each record (or a queue), operations that belong to higher priority queues (created by a higher priority planner) must always be executed before executing any lower priority operations.”
QueCC Architecture

Priority-based Parallel Planning Phase
QueCC Architecture

Priority-based Parallel Planning Phase

Batching Client Transactions

Planning Threads (Pre-determined Priority)

Main Memory DB Storage

Index

High Priority Queues
Low Priority Queues
QueCC Architecture

Priority-based Parallel Planning Phase

Batching Client Transactions → Planning Threads (Pre-determined Priority) → Execution Queues

Main Memory DB Storage → Index

Queues

High Priority Queues

Low Priority Queues
Queue-oriented Parallel Execution Phase

QueCC Architecture

Batching Client Transactions

Planning Threads (Pre-determined Priority)

High Priority Queues

Low Priority Queues

Execution Queues

Execution Threads

Main Memory

DB Storage

Index
Client Transactions

Planning Thread #1

Planning Thread #2

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions

QueCC

Abort Count: 0
**Priority Groups**

- **Low-priority Queues**
  - Transaction w3(b)
  - Transaction r3(c)

- **High-priority Queues**
  - Transaction r1(a)
  - Transaction w1(b)

**Client Transactions**

- Transaction w4(b)
- Transaction r4(d)
- Transaction w2(b)
- Transaction r2(a)

**Planning**

- **Thread #1**
  - Transaction r1(a)
  - Transaction w1(b)

- **Thread #2**
  - Transaction w3(b)
  - Transaction r3(c)

**Committed Transactions**

- a
- b
- c
- d

**Abort Count:** 0
Client Transactions

Planning Thread #2

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions
Client Transactions

Planning Thread #2

Planning Thread #1

Prioritized Execution Queues

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions

w_4(b)  r_3(c)  r_4(d)

w_3(b)  r_3(c)

w_2(b)  r_2(a)

w_1(b)  r_1(a)

r_2(a)  w_3(b)  r_3(c)  r_4(d)

r_1(a)  w_2(b)  r_2(a)

r_1(a)

Abort Count: 0

QueCC

61
Client Transactions

Execution Thread #1

Execution Thread #2

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions

QueCC

Abort Count: 0
Client Transactions

Execution Priority Invariance

Execution Thread #1
- r₁(a)
- r₂(a)

Execution Thread #2
- w₁(b)
- w₂(b)

Low-priority Queues
- w₄(b)
- w₃(b)
- r₃(c)
- r₄(d)

High-priority Queues

Priority Groups

Committed Transactions
- a
- b
- c
- d

Abort Count: 0

QueCC

QueCC
Client Transactions

Execution Priority Invariance

Execution Thread #1

Execution Thread #2

Abort Count: 0

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions

w₄(b)

w₃(b)

r₃(c)

r₄(d)

w₂(b)

r₁(a)

r₂(a)

w₁(b)

a

b

c

d
Client Transactions

Execution Priority Invariance

Execution Thread #1

Execution Thread #2

Abort Count: 0

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions

QueCC

Committed Transactions:

- w2(b)
- r1(a)
- r2(a)
- w1(b)

Low-priority Queues:

- r4(d)

High-priority Queues:

- w4(b)
- w3(b)

Execution Priority Invariance
Client Transactions

Execution Thread #1

Execution Thread #2

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions

QueCC

Abort Count: 0
Client Transactions

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions

QueCC
Abort Count: 0
QueCC
Abort Count: 0

Execution Thread #1

Client Transactions

Execution Thread #2

Priority Groups

Low-priority Queues

High-priority Queues

Committed Transactions

w₄(b)  w₃(b)  w₂(b)  r₁(a)
   r₄(d)  r₃(c)  r₂(a)  w₁(b)
QueCC
Abort Count: 0

Execution Thread #2

Execution Thread #1

Priority Groups

Low-priority Queues

High-priority Queues

✓ Deterministic Execution
✓ No aborts because of CC
✓ Minimal coordination among threads
✓ Not sensitive to multi-partition transactions
✓ Exploits Intra-transaction parallelism

Committed Transactions

w₄(b) w₃(b) w₂(b) r₁(a)
r₄(d) r₃(c) r₂(a) w₁(b)
ResilientDB Blockchain Fabric

Application Layer / Testbed (YCSB, SYCSB, TPC-C Benchmarks)

Enable/Disable Secure Transactions

Concurrent Control Protocols
- 2PL, QueCC, 2VCC, DORA, MVCC, Timestamp, H-Store, NoWait, Silo, Foedus, MOCC, TicToc, Cicada

Consensus Protocols
- GeoBFT, PoE, RCC, Delayed Replication, ByShard, RingBFT, Zyzzyva, Bitcoin-NG, PoW, PBFT, RBFT

Transaction Manager

Execution Threads

commit Protocols:
- Q-Store, 2PC, 3PC, Calvin, EasyCommit

Execution Threads

Crypto Toolkit

Block Creator (Distributed Ledger)

Storage Layer: Lineage-based Storage Architecture

Fault-tolerant Distributed Transactions on Blockchain., S. Gupta, J. Hellings, M. Sadoghi

https://github.com/resilientdb/
https://resilientdb.com/
Evaluation Environment

Hardware
- Microsoft Azure instance with 32 core
- CPU: Intel Xeon E5-2698B v3
  - 32KB L1 data and instruction caches
  - 256KB L2 cache
  - 40MB L3 cache
- RAM: 448GB

Workload
- YCSB: 1 table, 10 operations, 50% RMW, Zipfian distribution
- TPCC: 9 tables, Payment and NewOrder, 1 Warehouse

Software
- Operating System: Ubuntu LTS 16.04.3
- Compiler: GCC with -O3 compiler optimizations
Effect of Varying Contention

- 5 write and 5 read operation per transaction
- 32 worker threads

Workload contention resiliency
Cache locality under high-contention
Effect of Varying Worker Threads

• 5 write and 5 read operation per transaction
• Zipfian theta = 0.99

Avoiding thread coordination & eliminating all execution-induced aborts
Effect of Increasing Percentage of Multi-Partition Transactions in the Workload

![Graph showing the effect of increasing percentage of multi-partition transactions on throughput. The graph depicts a decreasing trend in throughput as the percentage of multi-partition transactions increases.]
Effect of Increasing Percentage of Multi-Partition Transactions in the Workload

QueCC is not sensitive to multi-partitioning

Two orders of Magnitude

4.3x at 1%
TPC-C Results

1 Warehouse (highly contended workload)
50% Payment + 50% NewOrder transaction mix

QueCC can achieve up to 3x better performance on high-contention TPC-C workloads
QueCC Conclusions

- Efficient, parallel and deterministic in-memory transaction processing
- Eliminates almost all aborts by resolving transaction conflicts \textit{a priori}
- Works extremely well under high-contention workloads
What’s Next: Q-Store

QueCC

Multi-core
Single-node

Execution
Queues

Q-Store

Partitioned
on Distributed
Cluster
What’s Next: Q-Store

Batching Client Transactions

Plan Local and Remote Execution Queues
What’s Next: Q-Store

Batching Client Transactions

Plan Local and Remote Execution Queues

Deliver Remote Execution Queues

Q-Store: Distributed, Multi-partition Transactions via Queue-oriented Execution and Communication, T. Qadah, S. Gupta, M. Sadoghi, EDBT 2020
What’s Next: Q-Store

Batching Client Transactions

Plan Local and Remote Execution Queues

Deliver Remote Execution Queues

Execute Queues

Q-Store: Distributed, Multi-partition Transactions via Queue-oriented Execution and Communication, T. Qadah, S. Gupta, M. Sadoghi, EDBT 2020
What’s Next: Q-Store

Q-Store: Distributed, Multi-partition Transactions via Queue-oriented Execution and Communication., T. Qadah, S. Gupta, M. Sadoghi, EDBT 2020
What’s Next: QBFT

QueCC
Multi-core
Single-node
Execution Queues

Q-Store
Partitioned on Distributed Cluster

QBFT
Partitioned & Replicated
What’s Next: QBFT

✓ Queue-oriented Byzantine Fault-Tolerance

✓ Resilient planning followed by resilient execution