Q-Store: Distributed, Multi-Partition Transactions via Queue-Oriented Execution and Communication

Thamir M. Qadah*
School of Electrical and Computer Engineering

Suyash Gupta
Department of Computer Science

Mohammad Sadoghi
Department of Computer Science

* Also With Umm Al-Qura University, Makkah, Saudi Arabia
Cloud Computing Trends

The rise of cloud computing

Large core counts

Large main-memory

<table>
<thead>
<tr>
<th></th>
<th>RAM</th>
<th>vCPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>24 TB</td>
<td>448</td>
</tr>
<tr>
<td>MSA</td>
<td>12 TB</td>
<td>416</td>
</tr>
<tr>
<td>GC</td>
<td>12 TB</td>
<td>416</td>
</tr>
</tbody>
</table>

Distributed Commit Protocols

- Two-phase Commit (2PC)
  - Very good general solution and widely used
  - Adds **overhead per transaction**
  - Can we avoid using it?
Distributed Deterministic Transaction Processing

• Provides strict serializability

• **Avoids** non-deterministic transaction **aborts** due to concurrency control

• Removes the **coordination** for transaction-commit from the **critical path**

• Key limitations: requires **knowledge of full read/write sets of transactions prior to execution**
Calvin Overview
Thomson et al. SIGMOD’12

Client Transactions

Single-threaded Sequencing
Calvin Overview
Thomson et al. SIGMOD’12

Client Transactions

Single-threaded Sequencing

Batches of Sequenced Transactions

Single-threaded Scheduling
Calvin Overview
Thomson et al. SIGMOD’12

Client Transactions

Single-threaded Sequencing

Batches of Sequenced Transactions

Single-threaded Scheduling

Multi-threaded thread-to-transaction
Conservative 5-phase execution

DB
Is it possible to avoid the two single-threaded pre-execution steps and improve parallelism during execution?
Key Ideas in Q-Store

✓ Combine sequencing and scheduling into a single step
✓ Unified queue-oriented processing paradigm
✓ Global execution priority invariant
✓ Support speculative and conservative executions of queues
✓ Support multiple isolation levels
Key Ideas in Q-Store

- Combine sequencing and scheduling into a single step
- Unified queue-oriented processing paradigm
- Global execution priority invariant
- Support speculative and conservative executions of queues
- Support multiple isolation levels
Calvin Vs. Q-Store

Client Transactions

Client Transactions
Calvin Vs. Q-Store

Client Transactions

Single-threaded Sequencing

Single-threaded Scheduling

Batches of Sequenced Transactions

Client Transactions

Multi-threaded Planning

Batches of queues of transaction fragments
Calvin Vs. Q-Store

Client Transactions

Single-threaded Sequencing

Single-threaded Scheduling

Multi-threaded thread-to-transaction
Conservative 5-phase execution

Batches of Sequenced Transactions

Multi-threaded Planning

Batches of queues of transaction fragments

Multi-threaded thread-to-queue
Speculative or Conservative

DB

DB
Processing Transactions in Q-Store

Planning Phase
1. Breakdown transactions into fragments
2. Create prioritized execution-queues of transaction fragments
3. Enforce a strict serial order of conflicting fragments within an execution-queue

Execution Phase
Process queues while maintaining the **global execution priority invariant**:

*Operations belonging to higher priority execution-queues must always be executed before executing any conflicting lower priority operations.*
Unified Queue-Oriented Transaction Processing

Clients

Network Buffers

Transaction Queues

Communication Threads

Worker Threads

Remote EQs

Remotely Planned EQs

Locally Planned EQs

In-Memory Storage

Index

Data

Batch Meta-data
Unified Queue-Oriented Transaction Processing

Clients

Network Buffers

Transaction Queues

Communication Threads

Remote EQs

Remotely Planned EQs

Locally Planned EQs

In-Memory Storage

Index

Data

Batch Meta-data
Unified Queue-Oriented Transaction Processing

- Clients
- Network Buffers
- Transaction Queues
- Communication Threads
- Worker Threads
- Remote EQs
- Remotely Planned EQs
- Locally Planned EQs
- In-Memory Storage
- Index
- Data
- Batch Meta-data
Unified Queue-Oriented Transaction Processing

Clients

Network Buffers

Transaction Queues

Communication Threads

Worker Threads

Remote EQs

Remote EQs

Locally Planned EQs

Remotely Planned EQs

Index

Data

In-Memory Storage

Batch Meta-data
Q-Store

Planning thread 1

High-priority Queues

Planning thread 2

Low-priority Queues

I

a

b

II

c

d

III

e

f
Planning thread 1

Planning thread 2

Q-Store
Q-Store

Planning thread 1

ACK (q_{1c})

Planning thread 2

ACK (q_{2f})
Evaluation Environment

32 (16 clients + 16 servers) AWS EC2 c5.2xlarge instances with:
- CPU: 8 vCPUs
- RAM: 16GB

Workload
- YCSB: 1 table, RMW and Read-only operations, Uniform and Zipfian distribution
- TPC-C: 9 tables, Payment and NewOrder

Software
- Operating System: Ubuntu LTS 16.04.3
- Compiler: GCC with -O2 compiler optimizations
Effect of Varying Batch Size

- 8 read and 8 RMW operations per transaction
- 50% multi-partition transactions
- Uniform distribution

Q-Store eliminates the bottleneck of single-threaded sequencing scheduling and scales well while increasing the batch size
Effect of Varying Batch Size

- 8 read and 8 RMW operations per transaction
- 50% multi-partition transactions
- Uniform distribution

Q-Store eliminates the bottleneck of single-threaded sequencing scheduling and scales well while increasing the batch size
Q-Store’s performance is comparable to non-deterministic protocols with 0% MPT
Q-Store’s performance is comparable to non-deterministic protocols with 0% MPT.
Effect of Multi-Partition Transactions

Calvin is sensitive to multi-partition transactions while Q-Store is not
Effect of Multi-Partition Transactions

The diagram illustrates the effect of multi-partition transactions on different systems. It shows the transactions per second (y-axis) for various systems as a function of the percentage of multi-partition transactions (x-axis). The systems compared are:

- CALVIN
- MVCC
- Q-Store
- MaaT
- NO_WAIT
- TIMESTAMP

Each system is represented by a different line and marker style on the graph.
Effect of Multi-Partition Transactions

Best performance with multi-partition transactional workload
Conclusions and Future Work

• We can **improve the performance and efficiency** of deterministic transaction processing by using **queue-oriented transaction processing principles**

• Q-Store improves system throughput over Calvin by **up to 22x**

• Q-Store improves system throughput over non-deterministic protocols by **up to two orders of magnitude**

• Future work include studying and developing **queue-oriented protocols for byzantine fault-tolerance in database systems**