Blurring the Lines between Blockchains and Database Systems: the Case of Hyperledger Fabric

Ankur Sharma*
Felix Martin Schuhknecht
Divya Agrawal
Jens Dittrich

Big Data Analytics Group
Saarland University
bigdata.uni-saarland.de
This is a distributed database system.
This is a blockchain system.
Can we say that the Blockchain Systems are next-gen Distributed Database Systems?
Can we say that the Blockchain Systems are next-gen Distributed Database Systems?

Not really!
Can we say that the Blockchain Systems are next-gen Distributed Database Systems?

Not really! → Outdated Transaction Processing Model
The order-execute model (Bitcoin, Ethereum, ...)

Client 1

Client 2
The order-execute model (Bitcoin, Ethereum, ...)

Client 1 ➔ Transaction ➔ Ordering Service ➔ Transaction ➔ Client 2
The order-execute model (Bitcoin, Ethereum, ...)

Client 1 -> Transaction

Client 2 -> Transaction

Ordering Service

Ordering Phase

Block
The order-execute model (Bitcoin, Ethereum, ...)

Client 1

Client 2

Transaction

Transaction

Ordering Phase

Execution Phase

Ordering Service

Block

Peer A1
append to ledger

Peer A2
append to ledger

Peer B1
append to ledger

Peer B2
The order-execute model (Bitcoin, Ethereum, ...)

Client 1

Ordering Phase

Client 2

Execution Phase

- Peer A1
- Peer A2
- Peer B1
- Peer B2

Transaction

Append to ledger

Ordering Service

No Scaling

Concurrency
The order-execute model (Bitcoin, Ethereum, ...)

Well-established properties of database systems since decades!

No Scaling

No Concurrency
The order-execute model (Bitcoin, Ethereum, …)
The order-execute model (Bitcoin, Ethereum, …)
The order-execute model (Bitcoin, Ethereum, …)
The Simulate-order-validate-commit model (Fabric)
The Simulate-order-validate-commit model (Fabric)

The diagram illustrates the simulation, ordering, and validation/commit phases of the model. The simulation phase involves clients and peers simulating transactions. The ordering phase involves the ordering service ordering the transactions. The validation/commit phase involves validating and appending transactions to the ledger.
The Simulate-order-validate-commit model (Fabric)

Simulation Phase
- Client 1
  - Peer A1
  - Peer A2
  - Peer B1
  - Peer B2
- Proposal
- Endorsement

Ordering Phase
- Transaction

Validation/Commit Phase
- Peer A1
- Peer A2
- Peer B1
- Peer B2
- Block
- validate
- append to ledger

Parallelization of Transaction Simulation → Scaling!
→ Concurrency!
The Simulate-order-validate-commit model (Fabric)

Optimistic CC

Simulation Phase
Transaction

Ordering Phase
Validation/Commit Phase

Parallelization of Transaction Simulation

Peer A1
Peer A2
Peer B1
Peer B2

Client 1
Client 2

Endorsement
Proposal

simulate
validate
append to ledger

Scaling!
Concurrency!
The Simulate-order-validate-commit model (Fabric)

- Simulation Phase
  - Transaction
  - Peer A1
    - Simulate
    - Endorsement
  - Peer A2
    - Simulate
    - Proposal
  - Peer B1
    - Simulate
    - Proposal
  - Peer B2
    - Simulate
    - Transaction

- Ordering Phase
  - Ordering Service
  - Block
  - Peer A1
    - Validate
    - Append to ledger
  - Peer A2
    - Validate
    - Append to ledger
  - Peer B1
    - Validate
    - Append to ledger
  - Peer B2
    - Validate
    - Append to ledger

- Validation/Commit Phase
  - Peer A1
  - Peer A2
  - Peer B1
  - Peer B2

- Parallelization of Transaction Simulation
  - Scaling!
  - Concurrency!

- Optimistic CC → Serialization Conflicts!
The Simulate-order-validate-commit model (Fabric)

Simulation Phase
- Transaction
- Endorsement
- Proposal
- Peer A1
  - simulate
- Peer A2
  - simulate
- Peer B1
  - simulate
- Peer B2
  - simulate

Ordering Phase
- Block
- Peer A1
  - validate
  - append to ledger
- Peer A2
  - validate
  - append to ledger
- Peer B1
  - validate
  - append to ledger
- Peer B2
  - validate
  - append to ledger

Validation/Commit Phase
- Peer A1
- Peer A2
- Peer B1
- Peer B2

Parallelization of Transaction Simulation
→ Scaling!
→ Concurrency!
Serialization Conflicts
Serialization Conflicts

A=5
B=3
C=7
Serialization Conflicts

Simulation Phase

A=5
B=3
C=7

w(A)=10
r(A)=5, w(B)=8
r(A)=5, w(C)=12
Serialization Conflicts

Simulation Phase

A = 5
B = 3
C = 7

w(A) = 10
r(A) = 5, w(B) = 8
r(A) = 5, w(C) = 12
Serialization Conflicts

Simulation Phase

A = 5
B = 3
C = 7

w(A) = 10
r(A) = 5, w(B) = 8
r(A) = 5, w(C) = 12

Ordering Phase

T₁ T₂ T₃

Simulation

Ordering

Phase

Phase
Serialization Conflicts

Simulation Phase

A=5
B=3
C=7

w(A)=10
r(A)=5, w(B)=8
r(A)=5, w(C)=12

Ordering Phase

T1 T2 T3

Validation/Commit Phase
Serialization Conflicts

Simulation Phase

- A = 5
- B = 3
- C = 7

- w(A) = 10
- r(A) = 5, w(B) = 8
- r(A) = 5, w(C) = 12

Ordering Phase

- T1
- T2
- T3

Validation/Commit Phase

- A = 10
- B = 3
- C = 7
Serialization Conflicts

Simulation Phase

A=5
B=3
C=7

w(A)=10
r(A)=5, w(B)=8
r(A)=5, w(C)=12

Ordering Phase

T1 ▶ T2 ▶ T3

 ✓

Validation/Commit Phase

A=10
B=3
C=7

outdated!
outdated!

T1 ▶ T2 ▶ T3

 ✓

outdated!
outdated!
Serialization Conflicts

Simulation Phase

A = 5
B = 3
C = 7

w(A) = 10

r(A) = 5, w(B) = 8

r(A) = 5, w(C) = 12

Ordering Phase

T1

T2

T3

✓

Validation/Commit Phase

A = 10
B = 3
C = 7

Commit Rate: 1/3

✗

✗

outdated!
outdated!

T1

T2

T3

Ordering Phase

Simulation Phase
Serialization Conflicts

Simulation Phase

A=5
B=3
C=7

w(A)=10
r(A)=5, w(B)=8
r(A)=5, w(C)=12

Ordering Phase

T₁ T₂ T₃

Validation/Commit Phase
Serialization Conflicts

Simulation Phase

A=5
B=3
C=7
w(A)=10
r(A)=5, w(B)=8
r(A)=5, w(C)=12

Ordering Phase

T1
T2
T3

Validation/Commit Phase

reordered
Serialization Conflicts

Simulation Phase

A = 5
B = 3
C = 7

A = 5
B = 8
C = 7

Ordering Phase

T2
T3
T1

reordered

Validation/Commit Phase

A = 5
B = 8
C = 7

✓
Serialization Conflicts

Simulation Phase

A=5
B=3
C=7

w(A)=10
r(A)=5, w(B)=8
r(A)=5, w(C)=12

Ordering Phase

T2
T3
T1

Validation/Commit Phase

✓
✓

A=5
B=8
C=7

reordered

A=5
B=8
C=12
Serialization Conflicts

Simulation Phase

A = 5
B = 3
C = 7

w(A) = 10
r(A) = 5, w(B) = 8
r(A) = 5, w(C) = 12

Ordering Phase

T2
T3
T1

Validation/Commit Phase

✓
✓
✓

reordered

A = 5
B = 8
C = 7
✓
A = 5
B = 8
C = 12
✓
A = 10
B = 8
C = 12
✓
Serialization Conflicts

Simulation Phase

A=5
B=3
C=7

w(A)=10
r(A)=5, w(B)=8
r(A)=5, w(C)=12

Ordering Phase

T2
T3
T1
reordered

Validation/Commit Phase

✓
✓
✓

Commit Rate: 3/3
Fabric++: Reordering of Transactions
Fabric++: Reordering of Transactions

1. build conflict graph:
Fabric++: Reordering of Transactions

1. build conflict graph:

\[ T_i \text{ writes a key, which is read by } T_j \]
Fabric++: Reordering of Transactions

1. build conflict graph:

$T_i$ writes a key, which is read by $T_j$
Fabric++: Reordering of Transactions

2. compute strongly connected subgraphs:

$T_i$ writes a key, which is read by $T_j$
Fabric++: Reordering of Transactions

3. compute cycle-free conflicts graph:

\[ T_i \text{ writes a key, which is read by } T_j \]
3. compute cycle-free conflicts graph:

\[ T_i \text{ writes a key, which is read by } T_j \]

4. compute schedule: \( T_5 \Rightarrow T_1 \Rightarrow T_3 \Rightarrow T_4 \)

\( T_0 \text{ and } T_2 \text{ aborted} \)
Fabric: Lock-based Concurrency Control
Fabric: Lock-based Concurrency Control
Fabric: Lock-based Concurrency Control

Simulation

Validation/Commit

Current State

X=(70,T4)

Y=(80,T3)
Fabric: Lock-based Concurrency Control

Simulation

Validation/Commit

Current State

\[ r(X) = (70, T_4) \]
\[ r(Y) = (80, T_3) \]

\[ X = (70, T_4) \]
\[ Y = (80, T_3) \]
Fabric: Lock-based Concurrency Control

Simulation

Validation/Commit

Current State

X = (70, T4)
Y = (80, T3)

r(X) = (70, T4)
r(Y) = (80, T3)

w(X) = (50, T5)
w(Y) = (100, T5)

✓

X = (50, T5)
Y = (100, T5)

✓

X = (50, T5)
Y = (100, T5)
Fabric: Lock-based Concurrency Control

Simulation

Validation/Commit

Current State

X = (70, T4)
Y = (80, T3)

X = (70, T4)
Y = (80, T3)

W(X) = (50, T5)
W(Y) = (100, T5)

W(X) = (50, T5)
W(Y) = (100, T5)

R(X) = (70, T4)

R(Y) = (80, T3)

✓
✓

T3 → T4 → T5 → T6

T3
T4
T5
T6

X = (50, T5)
Y = (80, T3)

X = (50, T5)
Y = (100, T5)

✓
✓

X = (50, T5)
Y = (70, T4)
Fabric: Lock-based Concurrency Control

Simulation

r(X) = (70, T4)
r(Y) = (80, T3)

w(X) = (50, T5)
w(Y) = (100, T5)

r(X) = (70, T4)
r(Y) = (80, T3)

Validation/Commit

T3  T4  T5  T6

w(X) = (50, T5)
w(Y) = (100, T5)

r(X) = (70, T4)
r(Y) = (80, T3)

Current State

X = (70, T4)
Y = (80, T3)

X = (50, T5)
Y = (80, T3)

X = (50, T5)
Y = (100, T5)
Fabric++: Multi-version Concurrency Control

Simulation

Validation/Commit

Current State

X=(70, T4)
Y=(80, T3)
Fabric++: Multi-version Concurrency Control

Simulation

Validation/Commit

Current State

version at start
T4

X=(70,T4)
Y=(80,T3)
Fabric++: Multi-version Concurrency Control

version at start
T4

\[ r(X) = (70, T4) \]

\[ \parallel \]

\[ T4 \]

Simulation

Validation/Commit

Current State

\[ X = (70, T4) \]

\[ Y = (80, T3) \]
Simulation

Validation/Commit

Current State

Fabric++: Multi-version Concurrency Control

version at start
T4

r(X) = (70, T4)

T4

w(X) = (50, T5)
w(Y) = (100, T5)

✓

✓

X = (70, T4) Y = (80, T3)

X = (50, T5) Y = (80, T3)

X = (50, T5) Y = (100, T5)
Fabric++: Multi-version Concurrency Control

- **Simulation**
  - Version at start: $T_4$
  - $r(X) = (70, T_4)$
  - $r(Y) = (100, T_5)$

- **Validation/Commit**
  - $w(X) = (50, T_5)$
  - $w(Y) = (100, T_5)$
  - $X = (70, T_4)$, $Y = (80, T_3)$
  - $X = (50, T_5)$, $Y = (80, T_3)$
  - $X = (50, T_5)$, $Y = (100, T_5)$

- **Current State**
  - $X = (50, T_5)$
  - $Y = (100, T_5)$
Fabric++: Multi-version Concurrency Control

**Simulation**
- **version at start**: T4
- **r(X)** = (70, T4)
- **r(Y)** = (100, T5)
- **w(X)** = (50, T5)
- **w(Y)** = (100, T5)
- **X** = (70, T4)
- **Y** = (80, T3)
- **T4** abort

**Validation/Commit**
- ✔
- ✔

**Current State**
- **X** = (50, T5)
- **Y** = (80, T3)
- **X** = (50, T5)
- **Y** = (100, T5)
Fabric++: Multi-version Concurrency Control

version at start
T4

r(X)=
(70,T4)
‖
T4

r(Y)=
(100,T5)
≤
T4
abort

w(X)=
(50,T5)
✓

w(Y)=
(100,T5)
✓

✓

X=(70,T4)
Y=(80,T3)

X=(50,T5)
Y=(80,T3)

X=(50,T5)
Y=(100,T5)
Early Abort of Transactions

Simulation Phase

Transaction

Client 1

Endorsement

Proposal

Endorsement

Transaction

Client 2

Proposal

Proposal

Proposal

Simulation/Commit Phase

Ordering Service

Peer A1

simulate

validate

append to ledger

Peer A2

simulate

validate

append to ledger

Peer B1

simulate

validate

append to ledger

Peer B2

simulate

validate

append to ledger

Validation/Commit Phase
Early Abort of Transactions

Simulation Phase
- Client 1
  - Proposal
  - Endorsement
  - Transaction
- Client 2
  - Proposal
  - Endorsement
  - Transaction

Ordering Phase
- Peer A1
  - simulate
- Peer A2
  - simulate
- Peer B1
  - simulate
- Peer B2
  - simulate

Validation/Commit Phase
- Peer A1
  - validate
  - append to ledger
- Peer A2
  - validate
  - append to ledger
- Peer B1
  - validate
  - append to ledger
- Peer B2
  - validate
  - append to ledger

Transaction abort
Early Abort of Transactions

Simulation Phase

Transaction

Endorsement

Proposal

Peer A1

Peer A2

Peer B1

Peer B2

Ordering Phase

Validation/Commit Phase

Transaction

Endorsement

Proposal

Peer A1

Peer A2

Peer B1

Peer B2

Transaction

transaction abort

Endorsement

Proposal

Endorsement

Transaction
Early Abort of Transactions

Simulation Phase
- Client 1
  - Proposal
  - Endorsement
  - Transaction
  - Peer A1
    - simulate
  - Peer A2
    - simulate
  - Peer B1
    - simulate
  - Peer B2
    - simulate
- Client 2
  - Proposal
  - Endorsement
  - Transaction

Ordering Phase
- Peer A1
  - validate
- Peer A2
  - validate
- Peer B1
  - validate
- Peer B2
  - validate

Validation/Commit Phase
- Peer A1
  - append to ledger
- Peer A2
  - append to ledger
- Peer B1
  - append to ledger
- Peer B2
  - append to ledger

Transaction Simulation
- Client 1
  - Peer A1
    - simulate
  - Peer A2
    - simulate
  - Peer B1
    - simulate
  - Peer B2
    - simulate

Validation/Commit Phase
- Peer A1
  - append to ledger
- Peer A2
  - append to ledger
- Peer B1
  - append to ledger
- Peer B2
  - append to ledger

Ordering Service with Reordering
- Early Abort of Transactions
  - Transaction abort

transaction abort
Early Abort of Transactions

Simulation Phase

- Client 1
  - Transaction
  - Endorsement
  - Proposal
  - Peer A1
    - simulate
    - Transaction
    - Peer A2
      - simulate
      - Transaction
    - Peer B1
      - simulate
      - Transaction
    - Peer B2
      - simulate
      - Transaction
  - Proposal
  - Endorsement

- Client 2
  - Transaction
  - Endorsement
  - Proposal
  - Peer A1
    - Transaction
    - Proposal
    - Endorsement
  - Peer A2
    - Transaction
    - Proposal
    - Endorsement
  - Peer B1
    - Transaction
    - Proposal
    - Endorsement
  - Peer B2
    - Transaction
    - Proposal
    - Endorsement

Ordering Phase

- Transaction
- Peer A1
  - validate
  - append to ledger
- Peer A2
  - validate
  - append to ledger
- Peer B1
  - validate
  - append to ledger
- Peer B2
  - validate
  - append to ledger

Validation/Commit Phase

- Transaction
- Peer A1
  - validate
  - append to ledger
- Peer A2
  - validate
  - append to ledger
- Peer B1
  - validate
  - append to ledger
- Peer B2
  - validate
  - append to ledger

Ordering Service with Reordering

Transaction
Validation/Commit Phase

transaction
abort
transaction
abort
transaction
abort

Early abort of Transactions already during simulation?
Experimental Evaluation: Setup

- 2 x Quad-Core Intel Xeon
- 48GB RAM
- Gigabit Ethernet
- Ordering Service
- Client
- Peers
Experimental Evaluation: Workloads
Experimental Evaluation: Workloads

Smallbank: asset transfer scenario
- 6 transactions: 5 update transactions + 1 read-only transaction:

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of users (two accounts per user)</td>
<td>100.000</td>
</tr>
<tr>
<td>Probability for picking a modifying transaction (Pw)</td>
<td>95%, 50%, 5%</td>
</tr>
<tr>
<td>s-value of Zipf distribution for account picking</td>
<td>0.0 - 2.0</td>
</tr>
</tbody>
</table>
Experimental Evaluation: Workloads

Smallbank: asset transfer scenario
- 6 transactions: 5 update transactions + 1 read-only transaction:

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of users (two accounts per user)</td>
<td>100,000</td>
</tr>
<tr>
<td>Probability for picking a modifying transaction (Pw)</td>
<td>95%, 50%, 5%</td>
</tr>
<tr>
<td>s-value of Zipf distribution for account picking</td>
<td>0.0 - 2.0</td>
</tr>
</tbody>
</table>

Custom:
- 1 highly-configurable transaction

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of accounts balances (N)</td>
<td>10,000</td>
</tr>
<tr>
<td>Number of hot account balances (HSS)</td>
<td>1%, 2%, 4%</td>
</tr>
<tr>
<td>Number of read &amp; written balances per transaction (RW)</td>
<td>4, 8</td>
</tr>
<tr>
<td>Probability for picking a hot account for reading (HR)</td>
<td>10%, 20%, 40%</td>
</tr>
<tr>
<td>Probability for picking a hot account for writing (HW)</td>
<td>5%, 10%</td>
</tr>
</tbody>
</table>
Successful Transactions (Smallbank)

Smallbank balanced workload (Pw = 50%)
Successful Transactions (Custom Workload)

18 different configurations of workload
Optimization Breakdown

Custom Workload:
BS=1024, RW=8, HR=40%, HW=10%, HSS=1%
Conclusion

Fabric
Conclusion

Fabric++*

* Open Source. Available at tiny.cc/fabricpp
Conclusion

Fabric++*

Up to 12x Improvement in Successful Transaction’s Throughput

* Open Source. Available at tiny.cc/fabricpp
Conclusion

Fabric++*

Up to 12x Improvement in Successful Transaction’s Throughput

Up to 50% Less Latency

* Open Source. Available at tiny.cc/fabricpp
Summary

The order-execute model (Bitcoin, Ethereum, ...)

Well-established properties of database systems since decades!

Serialization Conflicts

Fabric++: Multi-version Concurrency Control

Successful Transactions (Custom Workload)

18 different configurations of workload
Backup Slides
Successful Transactions (Smallbank)

(a) $P_w = 5\%$ (read-heavy)  
(b) $P_w = 50\%$ (balanced)  
(c) $P_w = 95\%$ (write-heavy)
Scaling of Fabric++: Custom Workload

(a) **Varying the number of channels** from 1 to 8. Per channel, we use 2 clients to fire the transaction proposals.

(b) **Varying the number of clients per channel** from 1 to 8. All clients fire their transaction proposals in a single channel.