Janus
Consolidating Concurrency Control and Consensus for Commits under Conflicts

Shuai Mu, Lamont Nelson, Wyatt Lloyd, Jinyang Li

New York University, University of Southern California
State of the Art for Distributed Transactions
Layer Concurrency Control on top of Consensus

Transaction Protocol (e.g., 2PC)

Geo-replicate for fault tolerance

Shard for scalability
Latency Limitation:
Multiple Wide-Area Round Trips from Layering

California  Texas  New York
Throughput Limitation: Conflicts Cause Aborts

- California
- Texas
- New York

```
a++
b++
a* = 2
b* = 2
```
Goals: Fewer Wide-Area Round Trips and Commits Under Conflicts

Best case wide-area RTTs

1

≥ 2

Tapir [SOSP’15]

... 

Spanner [OSDI’12]

... 

Calvin [SIGMOD’12]

Janus

Aborts

Commits

Behavior under conflicts
Establish Order Before Execution to Avoid Aborts

• Designed for transactions with static read & write-sets
• Structure a transaction as a set of stored procedure pieces
• Servers establishes consistent ordering for pieces before execution

Challenge:
Distributed ordering to avoid bottleneck
Establish Order for Transactions and Replication Together to Commit in 1 Wide-area Roundtrip

• Consistent ordering for transaction and replication is the same!
• Layering establishes the same order twice while Janus orders once

Challenge: Fault tolerance for ordering
Overview of the Janus Protocol

Pre-accept
Send pieces to servers
Establish initial order using dependencies
Detect conflicts

Conflicts?
Yes
Accept
Replicate dependencies

No
Commit
Establish final ordering
Execute pieces in order
No Conflicts: Commit in 1 Wide-Area Round Trip

Pre-accept

Commit

1 Local RTT

Execute

Execute

Execute

1 Wide-area RTT
Conflicts: Commit in 2 Wide-Area RTT
Conflicts: Commit in 2 Wide-Area Round Trips

Pre-accept | Accept | Commit

California
A
B

New York
A
B
Conflicts: Commit in 2 Wide-Area Round Trip
Janus Achieves Fewer Wide-Area Round Trips and Commits Under Conflicts

• No conflicts: commit in 1 wide-area round trip
  • Pre-accept sufficient to ensure same order under failures

• Conflicts: commit in 2 wide-area round trips
  • Accept phase replicates dependencies to ensure same order under failures
Janus Paper Includes Many More Details

- Full details of execution
- Quorum sizes
- Behavior under server failure
- Behavior under coordinator (client) failure
- Design extensions to handle dynamic read & write sets
Evaluation

- Throughput under conflicts
- Latency under conflicts
- Overhead when there are no conflicts?
- Baselines
  - 2PL (2PC) layered on top of MultiPaxos
  - TAPIR [SOSP’15]
- Testbed: EC2 (Oregon, Ireland, Seoul)

https://github.com/NYU-NEWS/janus
Janus Commits under Conflicts for High Throughput

TPC-C with 6 shards, 3-way geo-replicated (9 total servers), 1 warehouse per shard.
Janus Commits under Conflicts for Low Latency

TPC-C with 6 shards, 3-way geo-replicated (9 total servers), 1 warehouse per per shard.
Small Throughput Overhead under Few Conflicts

- 13% overhead from tracking dependencies
- Overhead from accept phase + increased dependency tracking
- Overhead from retries after aborts

Microbenchmark with 3 shards, 3-way replicated in a single data center (9 total servers).
## Related Work

<table>
<thead>
<tr>
<th></th>
<th>Isolation Level</th>
<th>1 RTT</th>
<th>Commit under Conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janus [OSDI’16]</td>
<td>Strict-Serial</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Tapir [SOSP’15]</td>
<td>Strict-Serial</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Calvin [SIGMOD’12]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spanner [OSDI’12]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDCC [EuroSys’13]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COPS [SOSP’11]</td>
<td>Causal+</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Eiger [NSDI’13]</td>
<td>Causal+</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

**EPaxos [SOSP’13]**

**Rococo [OSDI’14]**
Conclusion

• Two limitations for layered transaction protocols
  • Multiple wide-area round trips in the best case
  • Conflicts cause aborts

• Janus consolidates concurrency control and consensus
  • Ordering requirements are similar and can be combined!
  • Establishing a single ordering with dependency tracking enables:
    • Committing in 1 wide-area round trip in the best case
    • Committing in 2 wide-area round trips under conflicts

• Evaluation
  • Small throughput overhead when there are no conflicts
  • Low latency and good throughput even with many conflicts