RIOT: Replicated Independently Ordered Transactions

At Neo4j we have longstanding Raft implementation which we love and which has proven to be very reliable. However, the single-leader approach and total ordering of operations constitute a bottleneck for writes, nor does the protocol work natively across shards. It is time for a new transaction protocol.

To address the related concerns of write scalability and sharded graphs, we developed RIOT: a generalized consensus protocol that eliminates the need for centralized leadership and log replication. RIOT allows any server to coordinate work at any time. It achieves consensus through decentralized coordination over a directed acyclic graph (known as a TxDAG) of entries representing transactions. RIOT allows commutative operations to execute concurrently while strictly preserving order where conflicts would otherwise occur.

While RIOT is a general-purpose protocol, it is rooted in the specific challenges of distributed graph databases, and specifically the need to guarantee reciprocal consistency—ensuring that relationships spanning different shards remain consistent on both sides. While prior work on leaderless protocols has shown it is possible to uphold strong consistency, those protocols tend to rely on analysis of read and write sets prior to execution to determine dependencies. This strategy fails for graph workloads where data access patterns are dynamic, and keys are discovered only as a query traverses a graph. Instead, RIOT tracks dependencies the transaction level, rather than tracking individual keys, which significantly reduces the computational overhead (though makes it slightly more prone to spurious aborts).

Part of what makes RIOT novel is its departure from the linear log model in favour of a TxDAG, where each server maintains a graph of transaction dependencies via happened-before relationships. To maintain consistency without a leader, servers exchange metadata known as the "leading edge"—the set of committed transactions that have no committed children (though there may be children in the prepared state). When a coordinator proposes a transaction, participants check this leading edge against their local history. Crucially, participants respond not only with a vote but also with their leading edge to ensure mutual compatibility in a world where servers can advance without permission from their peers. If the bilateral check fails, then the vote is treated as an abort for safety, while if the histories are compatible, the transaction proceeds. This allows the system to reach atomic agreement on entry ordering without a centralized coordinator.

We have integrated RIOT into Neo4j and compared it against the production Raft implementation. Even in a single shard, for an apples-to-apples comparison, RIOT delivers up to 2.5 times higher throughput and 2.3 times lower tail latency for common workloads. Notably, the removal of leader elections means that the system is more resilient to failure; whereas leader-based systems may pause service during re-election, RIOT avoids these latency spikes and maintains continuous availability.

We believe it would make an interesting contribution to the conference and spark lively debate. Having formally proven RIOT with a team from the University of Sydney, we are well prepared for scrutiny and any heckling!