The Flotilla Protocol for Strong Consistency in Distributed Graph Databases

Jim Webber (jim.webber@neo4j.com)
Chief Scientist, Neo4j

Introduction

Neo4j is a graph database. which stores and queries networks of connected data. Historically, Neo4j is transactional replicated database which is performant and safe but at the expense of massive scale. Our users trust the database not to corrupt their data but some require scale with that same level of dependability.

Problem statement

In some contemporary graph databases, scale is achieved by building a graph layer atop an eventually consistent database (e.g. Apache Cassandra). The graph parts of the solution thus "inherit" the scale properties of the underlying database (at least to a degree). However, Ezhilchelvan et al[1] show that such databases will corrupt data under normal operation in the absence of write isolation.

The problem arises because two replica sets need to be kept in sync when an edge spans vertices across two database nodes. While such databases guarantee convergence for a single replica set (or key space or partition), they have no provision for semantically synchronizing two replica sets. This is a sad result because it means we cannot re-use existing commodity database software safely for distributed graphs.

Our solution

We built the current replicated version of Neo4j atop the Raft protocol. Our work since the last HPTS has involved composition of Rafts into a multi-shard commit protocol suitable for distributed graphs (though it can conceivably be used for other things) called "Flotilla."

Flotilla is a chain commit protocol that treats the database as a set of shards where each shard is a Raft group. When building the outward chain, data is committed to the Raft log for each participating chain, but not applied to the database. This is analogous to the prepare in 2PC. On the return chain, assuming no faults or constraint violations, each shard applies the committed entry to its local database.

If a transaction is aborted, the return chain writes an abort record into the Raft log and skips application to the database on each Raft group back to the origin. This increases recovery complexity a little, as two passes of the log may be needed, but it is not hard to reason about.

As an opinionated simplification (versus more sophisticated protocols like Linear Transactions), in the current version of Flotilla, commit ordering is from the lexically smallest to largest shard ID. If two transactions meet on a shard one will be blocked until the other's return chain comes back. This prevents mutually out of order arrivals (as per Ezhilchelvan et al) as each shard in the transaction sees the same ordering. Finer grained concurrency (e.g. page) is plausible.

The future

Flotilla is aggressively simplified for ease of implementation and there are numerous opportunities to improve concurrent cross-shard transactions and to refine its operability. This protocol will underpin our partitioned-graph cluster architecture and airing it to an engaged audience at HPTS would be very helpful.

---

[1] https://doi.org/10.1007/978-3-030-02227-3_1