Faster PostgreSQL Replication with eRPC - Final Report

Gustavo Angulo Mezerhane
Carnegie Mellon University

Alexander Yu
Carnegie Mellon University

Yue Yin
Carnegie Mellon University

1 Abstract

Database systems usually use replication to improve availability, fault tolerance, and performance of the system. In particular, PostgreSQL, a disk-based database management system optimized for OLTP workloads, uses a master-standby architecture and streaming replication where the master server sends write ahead log records in messages to the standby server via TCP/IP on UNIX sockets. These message speeds have an impact on the replication, as slow speeds can result in replica divergence or slow transnational throughput, depending on if the replication is asynchronous or not. We look at the performance of PostgreSQL's replication messaging protocol, and replace it with eRPC, a high performance RPC library. Using eRPC, we achieve a reduction in message latency between the master and the standby, and 16% faster transactional throughput in the TPC-C benchmark when executing with synchronous replication.

2 Introduction

A database management system (DBMS) is a critical component of virtually every modern day application. Companies generate, store, and analyze business critical data in DBMS's. As such, many users of DBMS's expect high availability, fault tolerance, and performance from their data management solutions.

Many DBMS's use replication to fulfill these expectations. Database replicas serve as copies of the data, with replicas shipping updates between each other to stay up to date. Replicas can thus fulfill queries into the database that users make. The database system itself can then use these replicas for a variety of reasons:

1. **Availability:** Having multiple replicas ensures that the database has high availability that can fulfill client's requests when network is lossy and machines are crashing.
2. **Fault Tolerance:** Multiple replicas ensures that in the event of a crash, data is not lost, and the system can fulfill client's requests by rerouting them to an active replica.
3. **Load Balancing:** The DBMS can cleverly distribute client requests across replicas to balance the work across all available nodes.
4. **Performance:** The DBMS can make use of the replicas to increase throughput through its distribution of user requests. For example, long, read-only analytical queries can be handled by a replica in order to not slow down short, critical write queries executing on a particular node.

Different DBMS's handle replication schemes differently. A scheme commonly used by many DBMS's is a `Master-Slave` scheme. In a Master-Slave scheme, any requests are sent through the master, which can then decide how to handle it, or sent to a central dispatcher which forwards it to the appropriate replica. In a typical Master-Slave scheme, write requests are handled by the master node, whose job is to write the change in its own copy of the data, and forward the request (in some form) to the slave replicas so they can reproduce the write on their copy of the data. Read requests can be handled by either a master or slave replica, providing high availability across replicas.

A critical part of any replication scheme is the sending of information necessary from the master to the slave node so that it can reproduce or handle a request made to the DBMS. This presents a challenge of how best to send data between replicas. We will also take note of the two types of replication as they face different messaging challenges:

- **Asynchronous replication**: Transactions on the master are allowed to commit before the change is reflected on a standby
- **Synchronous replication**: Transactions block until its changes are reflected on a standby

In the case of asynchronous replication, replica divergence, or how different the data between replicas is, is extremely