Parallel Paxos

Harshal Sheth
Advised by Robert Soule

Summary

Many applications use distributed consensus algorithms to ensure that data remains intact and consistent, and that the system remains fault-tolerant across many nodes. However, they are also almost always the bottleneck in such distributed systems. One such algorithm, which is widely in use today, is Paxos.

In database systems, Paxos serves to impose an agreed-upon ordering across incoming database commands. Previous work has optimized around the bottleneck of Paxos by separating the control plane from the data plane; that is, it separates the agreement process on this ordering from the execution of those ordered commands.

My work continues this effort, specifically on the behavior of executing single commands which span across multiple shards of a database. We approach this problem with two goals: correctness and efficiency. While we have designed a system that achieves these goals in theory, the implementation is not yet far enough along to validate this belief.

Without the previous work, which separated the agreement and execution, this would essentially serialize the commands, rather than allowing the opportunity for parallel execution. The efficiency of our proposed algorithm depends largely on the ordering and types of commands. In particular, we experimented with a setup in which the state of a database is partitioned across four shards, and incoming commands involve any number of these shards. The main restriction was that an executing command must have exclusive access to all of the shards which it involves at the time of its execution.

After we designed the synchronization algorithm which satisfies those requirements efficiently, we also began implementation work. We were building off of a previously developed, research-quality codebase that handled some of the scaffolding already. Much of the work this semester was simply getting familiar with the codebase and learning about the associated environment and hardware. We have also implemented the aforementioned changes, as well as the client changes required for testing purposes, but have bugs which prevented the production of many direct results.

Background Reading

Papers
Existing Work

This project was a continuation of work that my advisor and some of his Ph.D. students started. As such, some of the legwork was already completed.

In particular, the Paxos protocol has already been implemented in P4, and runs on a switch developed by Barefoot Networks. Barefoot provides the compiler and toolchain necessary executing these programs. The P4 programs are available in https://github.com/usi-systems/p4xos-public.

A client library has also been written, which enables the use of Paxos in C/C++ programs. This library is available at https://github.com/usi-systems/libpaxos. This library internally uses a library called the Data Plane Development Kit (DPDK) for its internal networking functionality. The usage of the DPDK library enables high-performance and low-latency packet handling, since it uses a polling architecture rather than an event-driven architecture.

Additionally, some work was already done to use this switch-based Paxos implementation and library in a real system. It runs four instances of RocksDB, a single-threaded key-value store developed by Facebook. These four instances represent a distributed database, which shards its state across these four RocksDB instances. The code is responsible for handling incoming requests and directing them to the correct instances. Additionally, there is a client program called xclient which is used to generate load.

Initial Experiment

I initially ran an experiment which tested the throughput and latency as the number of shards increased from one to four. After some data wrangling, it produced the following results:
Notably, the throughput increased sublinearly with the number of replicas. Upon further analysis, we realized that the requests we were sending were multi-shard commands, meaning that all of the RocksDB instances would need to be available before the commands could be executed. As such, this
experiment did not really test the scaling with number of shards, but instead tested the scaling of only the Paxos agreement in the control plane.

Based on these results, we decided on two main tasks: (1) modifying the xclient to send commands with varied shard configurations and (2) updating the synchronization mechanism in the replicas to handle these multi-shard commands.

**xclient Modifications**

Modifying xclient was relatively straightforward. At its core, we needed a mechanism to configure the sequence of commands. I added a configuration option to xclient allows us to specify a configuration file, where this file contains a sequence of commands. More specifically, it contains a list of bitmasks representing the set of shards associated with a given command.

**Synchronization Mechanism**

We also designed a synchronization mechanism for coordinating between these shards. If we have N shards, an efficient algorithm that requires \(O(2^N)\) locks is fairly straightforward: simply create a lock for every possible combination of included shards, and acquire the appropriate lock as commands are delivered to the system.

As suggested by Fernando Pedone, we developed an algorithm that only required \(O(N^2)\) locks. While this didn’t make a difference for our tests, where we had 4 shards, we wanted something that might scale better. The algorithm is described below.

**Algorithm**

For each thread, we have a vector \(L\) of semaphores of size \(N\). We have a command \(C\) arriving which involves shards \(S_c\). Finally, we decide that shard \(T\) will be the shard which executes command \(C\).

For shard \(T\):

for each shard \(X\) in \(S_c\) where \(X\) is not \(T\):
    wait on semaphore \(L_{T}[X]\)
    execute \(C\)
for each shard \(X\) in \(S_c\) where \(X\) is not \(T\):
    signal on semaphore \(L_{T}[T]\)

For shard \(X\) in \(S_c\) that is not \(T\):
    signal semaphore \(L_{T}[X]\)
    wait on semaphore \(L_{T}[T]\)
Current Progress

I have currently implemented the xclient modifications and synchronization changes in the replicas, and am in the process of debugging some issues. Once those issues are resolved, I should be able to re-run the performance tests with a variety of command shard configurations.