Crux: Locality-Preserving Distributed Systems

Application to a Distributed Stock-trading System

Christine Hong
Team Members: Sage Price, Kojiro Murase
Advisor: Bryan Ford

April 29, 2015

Abstract

Crux is a general framework to build locality-preserving distributed systems. Currently, distributed systems achieve scalability by balancing loads across many machines, but for large systems, users who are geographically close can suffer a large delay in communications if transmissions are forced to pass through the entire network. Crux improves latency for localized interactions by transforming any existing scalable distributed algorithm $A$ into a new algorithm $A^*$ with a greatly improved worst-case response time. It’s in essence an algorithm that transforms an algorithm. It guarantees for any two clients interacting via service requests to $A^*$ worst-case response latencies proportional to the network delay between $u$ and $v$. My project was to implement a practical application demonstrating the use of Crux in improving communication time for a stock trade type system.

Background and Motivation

The Crux proposal is a sub-project of Dissent, a research collaboration between Yale University and UT Austin to create a powerful, practical anonymous group communication system offering strong, provable security guarantees with reasonable efficiency.

The goal of Crux is to take any distributed algorithm or system of any kind and systematically transform it into an equivalent distributed system that preserves
locality. Currently, practically all large high-level scalable distributed protocols and services completely fail to provide this locality-preserving feature and as a result, have a great worst-case latency for even nodes that are physically close to each other.

There are not many experiments practically applying the Crux scheme yet. Previously, prior implementations of Crux focused on demonstrating the feasibility of taking a general distributed algorithm and showing the viability of constructing a system of nodes that preserves communication speed. My project’s objective was to actually implement Crux on database operations. In particular, we wanted an application that required stronger consistency guarantees and prototyped evaluation of the Crux scheme to a distributed stock trading type of system. The goal was to show that, if we can automatically create multiple instances of a geographically spaced out large scale decentralized system, through the Crux scheme we can achieve the locality-preserving access property without changing the basic algorithms underlying these existing distributed systems.

**System Design**

The basic idea of our Crux stock trade application was to allow an international bank with trading offices spread out all over the world to execute trades of local-interest commodities faster. If orders needed to be propagated around the world, this trading scenario would require the distributed system in use to provide an ACID-type database for the action of matching a “buy” order with a “sell” order. In particular, our system would need to make sure one “buy” order got matched with exactly one “sell” order, and that one Widget never accidentally got “sold” or “bought” twice due to race conditions. This meant we needed to use the Crux scheme to set up a set of distributed instances of a scale-out NoSql database. We chose to use MongoDB because of the ease of creating multiple instances (separate clusters) of the database onto nodes in the Crux system.
A couple major assumptions were made about the structure of the Crux network for the application. These include: (1) the communication A -> B takes the same time as messages from B -> A, (2) the triangle inequality holds in all messaging so (A->B->C) >= (A -> C), (3) rings are inclusive, and (4) given a landmark node running multiple rings, if there are no new nodes that would join ring i+1 that do not appear in some ring <= i, then we do not instantiate an i+1 ring, and continue onto i+2.

To implement Crux, we determined the larger structure of the network to build local clusters for smaller database instantiations. We worked to collect timing data on an experimental computer cluster known as Planet Lab, which contains a collection of machines connected to the Internet and distributed across the globe. From this timing data we assigned rankings to the nodes in the network, which we then used to create the Crux topology of the nodes.

After gathering information about the relative distance and communication time between all users in a network, we constructed a Crux-reduced topology of the network, at the memory cost of maintaining additional instantiations of the database in smaller scales. Information about network timing was converted into configurations to build a Crux-ified version of the larger distributed database. This included constructing collections of topologically near users and determining from this data which nodes would be in charge of maintaining databases and informing the network of which nodes would join which database. After all configurations had been determined, we automatically booted up and constructed the network in a systematic manner.

Due to time restrictions, our implementation only reduced reading time, not writing time. Writing to all levels of the database participated in always includes the top level whereas reading may stop as soon as the first ring with data of interest is hit.
Project Tasks

Outlined are the steps to collect data and instantiate a Crux-ified version of MongoDB on Planet Lab. These include collecting timing data collection, configuring network setup, and deploying the Mongo databases on remote Planet Lab hosts.

While our use case for Crux requires modification to the internal database representation, the overall network construction mimics the original description of how to develop a locality-preserving network from the Crux paper[2]. After assigning a ranking system and timing information for all nodes in the network, we constructed a Crux-ified topology of the nodes.

1. Initial Network Creation (Setup)
Initially collect hostname information for all nodes on a Planet Lab slice with prompt for username and password. After collecting all nodes, all dead and unreachable nodes are removed. Assign landmark values to all nodes to create a ranking system. The rank information will be used to construct bunches later after collecting timing information.

2. Timing Data Collection
Move all necessary files for timing data collection to remote Planet Lab system. Ping all pairs of working Planet Lab nodes in the network to collect timing data. After all ping data is collected remotely, bring the data to local text file.

3. Ring Construction
Use ping times and ranks calculated during setup to determine a bunch for each node in the network. A node’s bunch consists of all other nodes that it will interact with. Use bunches to create clusters for each node. A cluster consists of all nodes that will interact with the host node’s rings. Create a ring list for every cluster. Create all necessary configuration files for each node.

4. Trading System
Run basic Crux-ified MongoDB trading system on each Planet Lab node. The trading program will create N instances of MongoDB and N data folders, where N is the
number of rings centered on a node. The system will wait for input before running commands specified in a text file.

**Personal Contribution to Team**

My work with this project was primarily focused on automating setup of all the nodes in the Crux configuration on Planet Lab, timing ping data collection of the nodes on Planet Lab, ring construction, and creating shards related to the MongoDB database.

I worked closely with Sage Price on designing setup and outlining and implementing the Crux algorithm. We also worked closely together on data collection and processing to enable efficient network creation. In terms of ring construction, I worked more on creating the bunches and clusters while Sage worked more on creating the configuration files and automated installation of MongoDB onto the Planet Lab nodes. Our work was written solely in Python and Bash scripts. The information produced by Sage and me was then used by Kojiro to experimentally demonstrate the viability of the Crux system as well as replicate requests and maintain the state of the database. I also worked with Kojiro on how to connect the network Sage and I created with the database Kojiro set up.

**Implementation**

The application of Crux to a distributed stock-trading system was implemented using Python and Bash scripts for the automated setup and deployment to Planet Lab and MongoDB was the NoSQL database for storing information in the system. We chose MongoDB after attempting to use Cassandra because Cassandra did not allow for easy creation of multiple instances of the database.

**Conclusion and Future Work**
We were able to construct a basic and working application of a distributed stock trading system that used the Crux scheme and MongoDB database for a network of nodes on the virtual Planet Lab. The setup of the application was successfully automated and dependable. However, there are a lot of additional features we would like to add that we were unable due to time constraints.

First of all, we would like to be able to produce ACID database behavior in a Crux network using simple read/write operations. While implementing the databases' internal structure we had many discussions about how to do this, but any solution we found required implementing some sort of semaphore or global lock on transactions at top-level databases. As these top-level databases would have the same processing time as a naive database comprised of the entire network, such an operation would defeat the purpose of creating the Crux network.

Additionally, we would like to test on a more reliable network system than Planet Lab, which hosted the testing of our Crux application. We encountered a large amount of trouble with Planet Lab due to frequent packet drops and memory caps on some nodes. On a more stable system, we would like to gather more dependable data on timing results related to latency within the system.

Lastly, we would like to create a user-friendly GUI that allows for real-time input for our application. Currently, all commands are done via command terminal.

**Acknowledgements**

Thank you to Bryan Ford for advising me and welcoming me to his Dissent team, Jose Faleiro for guiding me on how to implement Crux properly, and my teammates Sage Price and Kojiro Murase for working on the practical application with me. Understanding Crux properly and how to integrate it with Planet Lab and MongoDB was a time-consuming and (sometimes frustrating) process, but I had a great time
learning the algorithm and a lot of new technologies to implement it in my application.

References