Construction of Crux Network Topology for Database Operations

CPSC 490 Spring 2015

Author: Sage Price SY '15
In Collaboration With: Christine Hong TC '15 and Kojiro Murase SY '15
Advised by: Brian Ford
I. Introduction

Crux originated as an attempt to preserve locality in large distributed systems. By preserve locality, we mean to say that when two users who are geographically near interact through the network they are able to interact in time proportional to the distance between the users. In large distributed systems, users which may be geographically proximate nonetheless often suffer large latency in communications due to being forced to pass their transmissions through the entire network. E.g. users located immediately across international borders who are mapped to different regions internally but are in fact adjacent to one another in reality.

Prior implementations of Crux have focused on demonstrating the feasibility of taking a general distributed communication algorithm and constructing a viable topology preserving communication speed. However, to this point no work has been done to demonstrate the practicality of applying the ideas of Crux to database operations. The goal of our larger team was to demonstrate that the Crux algorithm can be successfully applied to implement a distributed ACID database.

II. Contribution to Project

My work with this project was primarily focused on taking timing information about a collection of computers and converting this data into topological information for constructing a Crux network. I worked closely with Christine Hong on data collection and processing to enable efficient network creation. The information produced by Christine and I was then used by Kojiro to produce and experimentally demonstrate the viability of this system as well as replicate requests and maintain database state. In addition to my code responsibilities, I worked alongside Koji and Jose in planning how to proceed with action replication in the network as well as the internal structure of the databases.

My code responsibilities interacted heavily with Christine's which led to frequent collaboration and overlap in our work. We were guided in our efforts by DeDis team member and Yale CS graduate student Jose Faleiro as well as our adviser Bryan Ford. My work extends prior work by Jose and Michael Nowlan, who have previously built Crux implementations of distributed hash tables and publish/subscribe services.

III. Tasks

- Process timing data into usable format for bunch and cluster creation
- Implement bunch creation
- Implement cluster creation
- Determine ring structure
- Produce configuration files for database initialization

IV. Implementation

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. Due to this I was able to develop an application agnostic procedure for constructing a Crux compatible topology when given valid rank and timing information for all users participating in the network based on the description of the Crux algorithm given in the original research paper.

A. Bunches

For a given user in a Crux network, the user's bunch consists of those other users who maintain instances of the distributed system that the given user will join. As we decided on implementing Crux with inclusive rings we determined that it would be most efficient to establish what ring a user would join only at the end of bunch and cluster creation.

Due to this bunch creation was quite simple and implemented the following procedure:

Sort nodes by ping time
For each node X in the network
   for each node Y in the ping list
      if rank(Y) >= the largest seen so far, add it to X’s bunch

Following this process produces a bunch for each node consisting of those nodes that will maintain rings to join, where each node in the bunch is associated with its ping time to the bunch node.

B. Clusters

After determining bunches for each node in the network, we use this information to construct a cluster for each node. A node's cluster is defined to be the set of nodes that include that node in their bunch. So if x is an element of y's bunch, then y is an element of x's cluster. Additionally every node is defined to be a member of its own cluster, though for our use case we determined it would be most
efficient to not maintain this. This decision followed from a couple reasons including MongoDB’s data distribution procedure, but most importantly as a result of how we elected to construct rings.

A ring must be constructed about a node X for every instance in which the range covered by the ring about x includes a node whose distance from X is larger than any previous rings (this follows from the protocol assigning clusters to rings). Consequently if we were to include each node in its own cluster then for consistency we would be required to initiate rings of smallest size for every node in the network. When we forego inclusion of oneself in a cluster, we eliminate the possibility of having rings that contain exclusively the central node itself, as well as reducing the cost on the central node in all rings by not having it act as a component of the distributed algorithm.

However it should be noted that this last part is only reasonable due to our implementation of Crux on MongoDB where data is stored remotely across the network with one centralized access point, specified to be the center of the ring. As an aside, minor time savings could be had by making every node in a ring serve as an access point to the database and performing local look-ups, but as we found that database look-up time swallows ping time in our distributed database we elected to prioritize ease of implementation over the overhead cost of such a change.

**C. Database Design and Request Replication**

While we were able to mimic the topology creation for the Crux network originally outlined by Ford et.al., ACID database transactions would be impossible without some modification to how read/writes are made in the larger network. To facilitate such actions, Koji and I talked with Jose and Professor Ford, eventually determining a couple potential schemes by which to enable ACID 'trading', of which we implemented one.

A standard Crux implementation requires creating multiple smaller networks within the larger system. If this is done for our trading system however, then you create the potential for invalid purchases and sales. Suppose that you have two rings A and B with nodes N and M respectively. Now suppose that there exists some item X for sale in the databases maintained in rings A and B. If N and M move to purchase the item X simultaneously, then both transactions would be successful as there is no check on who can purchase what as we would normally rely on the databases themselves to provide consistent interactions. In an effort to avoid this scenario we decided on the following database structure.

Instead of simply having a market style key-space in each database where people can deposit and withdraw items freely, we moved to a rumor-style exchange system. In this system when someone
wishes to make a purchase or sale of some item, they insert that information into all databases that they participate in. That is, if a user A is looking to sell the item X, then they would insert a 'SELL: X, VENDOR: A' entry into all rings that they have joined. If they were looking to buy, then they would post a similar message such as 'BUY: X, BUYER: A'. However if in posting these messages they find a request already posted that matches their request, then they initiate an exchange with the buyer/vendor associated with the item in question. At that point it is up to the person who posted the original item notice to decide whom to buy/sell the item from/to and only buy/sell it once.

The major alternative to this proposal was to implement exchanges directly in the database. Whenever an individual who is selling finds a matching buy request in a database, they can then elect to place the item in the corresponding database to make a trade. This, however, requires that a seller be constantly searching through all databases they are involved in, an additional overhead cost that does not help us in reducing network latency. For this reason we eventually decided upon taking the former route in our design.

V. Remaining problems and extensions

Even after completing this assignment we are left with a couple questions. The major problem which we were never able to solve was how 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.

One alternative I was playing around with was to have larger rings act as officiators of smaller ring operations. At a high level the network would proceed in the following manner. When a user sees something they wish to trade for in the database, they would remove the item notice from the database and insert a time-stamped notice that they were claiming the item in question. This item notice would then be passed up to the smallest ring containing where the exchange was made. The higher level ring would then check the time-stamps of any conflicting purchases in the smaller rings and decide which one was correct. They would then pass this information to those users who had claimed the items in smaller rings and correctly assign the disputed item to the person in question. Implementing this would be very difficult and increase the overhead to larger nodes substantially, and could lead to issues with returning rings/databases to the state they were in before all such claimed exchanges occurred.
However, it is the only way I was able to come up with of feasibly preserving some element of locality without locking the entire Crux network.

Furthermore our project was focused only on construction of Crux in a static network. We do not currently have any idea of how to apply Crux to a more dynamic network where nodes may enter and leave at any time. Designing such a system would make an excellent future assignment.

VI. Acknowledgments

Kojiro Murase '15 and Christine Hong '15 for their patience, work, and discussion on this project. Bryan Ford and Jose Faleiro for guidance and direction throughout the process.