An Asynchronous Control Plane for CRUX

Project Report

Pasindu Nivanthaka Tennage

306888

CS-699

Computer and Communication Sciences- EDIC
DEDIS
Supervisor: Bryan Alexander Ford

Introduction

CRUX is a locality preserving replication system for key value stores that provides availability and strong consistency guarantees of a key value store service in the face of network partitions and other faults [1]. The control plane for CRUX is the algorithm which initiates the CRUX using a set of geographically distributed nodes and handles subsequent node churn and node movements (mobile nodes).

The current control plane of CRUX [2] is implemented using synchronous protocols. Synchronous protocols make the underlying assumption that the set of nodes have synchronized clocks, which does not hold in practice due to clock skew and drift. Hence the current control plane algorithm fails to operate correctly in the absence of synchronized clocks.

To address this problem with synchronized CRUX control plane ,in this project, we propose a novel control plane algorithm for CRUX which is fully asynchronous. Our asynchronous CRUX control plane algorithm provides support for initiating a new CRUX instance, joining new nodes to an existing CRUX system and handling node leavings (can be both planned and unplanned), in an asynchronized system. We use Threshold logical clock (TLC) [3] as the asynchronous logical clock algorithm in order to produce a synchronized time abstraction to the control plane running on top of a fully asynchronous system. We also employ the QSC protocol [3] as the consensus protocol which runs on top of the abstraction provided by TLC.

In the following sections, we first describe the background, assumptions and definitions, properties and the control plane protocol. We then present our simulation and experiment results of the control plane algorithm.

Background and Motivation

CRUX control plane algorithm should support the following set of actions by each participating node in the specified order.

1. Make consensus about the set of nodes participating in the CRUX
2. Calculate ping distances for the set of nodes that each node agreed in step 1
3. Multicast the ping distances to each node in the agreed set of nodes in step 1
4. Generate a ping distance matrix which contains the ping distances observed by each node
5. Make consensus on the ping distance matrix
6. Instantiate a new CRUX instance using the set of nodes and the ping distance matrix

In this project, we do not address the steps 2 and 6, because prior work [1][2] has addressed them. Making the control plane fully asynchronous has its own challenges. First, each of the above six steps should be done in the exact sequential order in each of the machines, thus we need a distributed barrier to impose the ordering and synchronization. Second, having consensus in a totally asynchronous environment is considered an impossible problem when there is at least one failing node. To address these two issues, we use TLC [3] and QSC protocols. TLC provides an abstraction of a synchronized barrier on top of a totally asynchronous message passing system, thus the first problem is addressed. QSC provides a randomized asynchronous consensus algorithm, which guarantees that a majority of nodes will progress making the consensus, given that there are enough consensus rounds. We design and implement the steps 1. 3, 4, 5 above using TLC and QSC.

Assumptions and definitions

Following are the definitions and assumptions we make in our design.

1. There is a set of well defined nodes, called the **initial admission committee**.
2. Each point to point communication is done using reliable unicast (TCP), we assume that no messages are lost
3. **N** is the size of admission committee and **f** is the maximum number of faulty processes in the admission committee
4. Each message is digitally signed by the sender, thus providing integrity protection and tamper resistance. Also each receiving process validates the signature of each message it receives.
5. We only assume **non byzantine behavior** of nodes in this protocol.
6. We denote the currently running CRUX instance as **c**, and each run of the control plane algorithm (also called an **epoch**) results in making a new CRUX round c+1.
7. Initial CRUX instance is already running

Properties

Following are the properties of the control plane algorithm we propose.

1. Consensus of membership list makes sure that majority of the members of the new admission committee agree upon some membership list that is proposed by a previous admission committee member.
2. Multicasting of the ping distances guarantees that at the end of the multicast process, with probability 1- {f/N}^10 each node has multicasted its ping distances to all the other nodes.
3. Consensus of the ping metrix makes sure that all the members of the new admission committee agree upon a ping distance matrix that is proposed by any one admission committee member.

Protocol

In this section we describe the new asynchronous control plane algorithm in detail. First we will look at how a new node can ask the admission committee to join for the c+1th round of CRUX (where c is the current CRUX round running). A new node should first contact the admission committee of CRUX round c, tell them that the node is interested in joining for the c+1th round, prove himself as an eligible node for the c+1th round, and then ask the admission committee to inform him when they are starting the construction of c+1th instance of CRUX.

A new node (who is outside the admission committee) who wishes to join for the c+1th CRUX round will first send a “join CRUX round” message with a name, a public key and ask for a threshold-signature from the admission committee. The new node needs to get a threshold signature from the admission committee, because threshold signature resembles that a majority of the admission committee is in favor of having this new node in the next CRUX round c+1. If the new node manages to get back a threshold-signature from the admission committee, it broadcasts it again to the admission committee (because there is no other way for each member of the admission committee to know whether the new node got threshold signatures). The current committee then acknowledges that it is a possible participant for the next CRUX round c+1, by sending a message “*partially eligible* for round c+1”.

Each node in the admission committee starts a new epoch of the control plane when there is more than zero number of new nodes waiting to join the admission committee or, a predefined time has elapsed since the last epoch of the control plane.

Without loss of generality, let's assume that each node is in TLC step 0 at the beginning. Upon reaching s = 0, members of the admission committee need to agree (consensus) on the set of new nodes who will be part of the c+1th instance of CRUX.

We use a variant of QSC protocol to make consensus about the admission committee of c+1th round of CRUX. Our use of QSC is as follows.

1. At s = 0, each node in the admission committee, generates a random number between (0, 1000*N). We assume that no two nodes obtain the same random number, and ties are broken using a node identifier as the secondary number.

2. Then each node Pi, multicasts a {message containing its view of the members (other nodes in the admission committee, and the partially validated new nodes he knows about), the random number and s = 0 time stamp}, to the admission committee.

3. Then the usual TLC witnessing process happens.

4. Node Pi, upon receiving more than N-f number of acks, multicasts a witnessed message to the admission committee with the same content.

5. Node Pj, after collecting N-f number of witnessed messages to move to s = 1, makes a list of nodes from which it received the N-f number of messages and multicasts it with s = 1.

6. Then the usual TLC witnessing process happens.

7. Each node Pk, at s = 2, knows about a majority of witnessed proposals in S = 0, because of the 2 step broadcast property of TLC [3].

8. Then each node uses the default TLC messages to move to s = 3.

9. By s=3, each node Pi individually chooses node Pj's s = 0 membership list, as the best possible set, by doing the following.

   If (Node Pi used some node Pk's s = 1 message with node list to move to s = 2, and Pj was in that list && Pj's proposal at s = 0, has the highest random number, out of all the (s = 0)'s message's random numbers, which Pi has heard by s = 2) or (out of all nodes in s = 1 node lists messages' node Pi heard at step s = 3, Pj is a member in some message && Pj's proposal at s = 0, has the highest random number, out of all the (s = 0)'s message's random numbers, which Pi has heard by s = 2)

Steps 1-9 above correspond to one round of consensus protocol in QSC.

There are two possible outcomes at the end of a single consensus round. A node has seen the consensus / or not. If the node has seen the consensus, it can continue to the ping distance calculation step, but has to make

sure that the majority of the nodes have reached the consensus. If the node has not seen the consensus, it should participate in another consensus round.

To address both these cases, we propose the following approach. If the consensus is seen with respect to node Pi, it will set a flag in the next unwitnessed message indicating that it has reached the consensus. Irrespective of the fact whether the consensus is made or not, each node will start a new consensus round.

Then, in step s = 5, each node receives a majority of s =3 messages and checks whether a majority of nodes have reached the consensus. If yes, then all the nodes aborts the new consensus round and moves to the ping calculation steps. However if a majority of the nodes has not reached the consensus, the new consensus round is continued, in the same way we explained in the above.

This approach guarantees the majority of nodes will reach the consensus in the same TLC step, thus providing the distributed lock step (barrier) property.

Now, admission committee members who reached the consensus on the membership list have the same view about the nodes who are going to participate in the c+1th CRUX instance. The admission committee then notifies the newly joined nodes that they have been added to the node list and sends the current step. After this, the admission committee retires and the selected members of the new CRUX round are treated as the admission committee.

Let’s say the new size of the admission committee is N1 (size of the membership list agreed upon) and f1 = N1/2+1. Now the new admission committee at CRUX round c+1 calculates the ping distances to each other member in the new admission committee.

Upon completing the distance calculation each node moves to the next TLC step and multicasts its ping distances to the admission committee 10 times. We use 10 rounds of multicasting to make sure that with probability 1- {f/N}^10 ≈ 1 each node succeeds in multicasting its membership list.

Then all the nodes in the admission committee participate in a consensus process identical to what we explained above for the consensus of the membership. Upon reaching consensus on the ping distances, each node moves to the next TLC step, and calculates the levels, bunches, clusters and ARAs.

Experiments and Analysis

In this section we first present our prototype implementation of the Asynchronous control plane algorithm. Then we present our simulation results. We implemented the asynchronous control plane algorithm in the Cothority framework [4] which runs on top of the ONET [5] framework. ONET uses web sockets as the application level abstraction for communication between any two hosts. Both ONET and Cothority framework are written in GoLang [7]. Our prototype implementation is available at [6]. We run our simulations in the ONET.

We first evaluate the correct execution of the asynchronous control plane. A correct CRUX control plane should be able to start a new epoch of CRUX with a given number of nodes, handle new node joins and handle node leavings. To experiment on the correct execution of CRUX, we perform two experiments; 1. start running the control plane initially with five number nodes and then add two nodes every 15 seconds and count the number of nodes that are included in subsequent epochs of the control plane and 2. start running the control plane initially with 15 number nodes and remove two nodes every 15 seconds and count

the number of nodes that are included in subsequent epochs of the control plane. We set the link latency to 20ms. Figure 1 below depicts the node add/remove events and the number of nodes which are included in each epoch of the control plane.

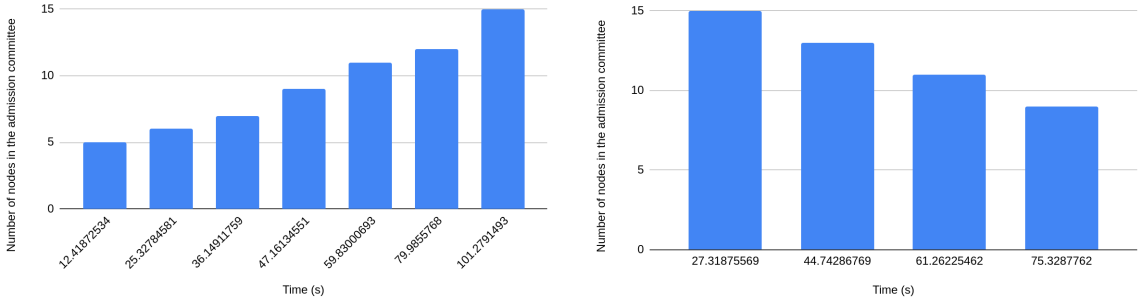

Figure 1: Correct execution of control plane; left: Node joins with two new nodes each 15 seconds, right: Node leavings with two nodes each 15 seconds

As seen in figure 1, the control plane executes correctly; we observe that the control plane dynamically handles the admission committee membership by adding and removing nodes.

A key requirement of an asynchronous CRUX control plane is that it should continue operation despite arbitrary network delays. We experiment on the network resilience of the control plane, by adding high delay components to links that are connected to N number of nodes; first we add high delays to all the links connected to one node, second to all the links connected to two selected nodes and so on (N can be also considered as the number of relatively slow nodes). We increase the N and measure the number of nodes which enter the admission committee. For all values of N, we set the network wide delay to 20ms and per link high delay to 100ms. We also experiment with different numbers of total nodes; 10, 15, 20. Figure 2 depicts the number of nodes which enter the admission committee.

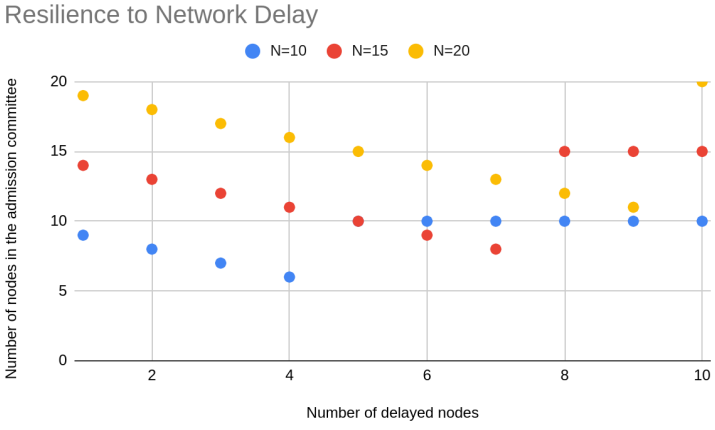

Figure 2: Resilience to network delays

We observe that for each value of total number of nodes (10, 15 and 20), more than a majority (10/2 +1 = 6 when total nodes = 10) of nodes enter the admission committee, hence this observation confirms the property of the control plane algorithm; despite the asynchrony of the network, a majority of nodes make progress. We also note that after N (the number of slow nodes)=5 (when the total number of nodes is 10), all the nodes (=10) enter the admission committee. We explain this behavior as follows.

For the control plane algorithm to make progress, it is required to have responses from at least a majority of the nodes at each step. When a majority of nodes (5 in this example) are slow, the nodes which are not slow

have to wait for at least one slow node; hence the overall progress depends on the progression speed of the nodes with slow links. Hence After N = 5; all the nodes get into the admission committee.

Bandwidth usage is an important aspect in the control plane algorithm. Theoretically the number of messages per CRUX control plane algorithm can be derived as N+3N*T = N*(1+3T), where N is the number of nodes and T is the number of TLC time steps it takes to run the control plane algorithm. T depends on the number of rounds it takes for the node consensus and the ping metrix consensus to finish. On average we observed that it takes roughly 30 time steps (when averaged) for a single round of control plane algorithm. Hence the number of messages is 91*N which is a linear function of N. If we define the average message size to be m, the total traffic for a single round of CRUX control plane is 91*N*m; a linear function of N. We simulate the bandwidth usage of the control plane w.r.t the number of nodes by measuring the bandwidth for different numbers of nodes. Figure 3 below depicts this behavior.

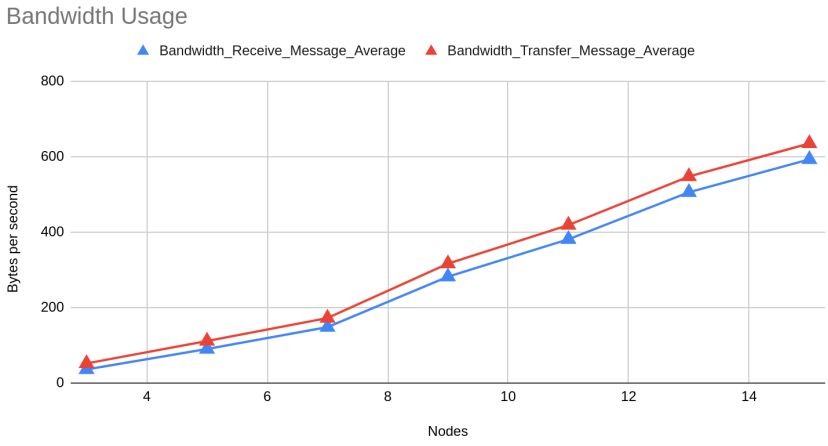

Figure 3: Bandwidth Usage

We can observe that the bandwidth usage increases near linearly when the number of nodes are increased. This validates our theoretical results. However it should also be noted that linear increase in bandwidth is not an optimum characteristic of distributed systems, and we will explore on how to reduce the bandwidth usage in our future work. Moreover, the bandwidth usage of each node is roughly equal, because the TLC algorithm treats each node equally, and each node receives and sends an equal number of messages.

Summary and Future Work

In this project, we aimed at building a novel asynchronous control plane algorithm for CRUX. We used TLC protocol to provide the abstraction of synchronized lock steps on top of a totally asynchronous system. We implemented our prototype in GoLang. We also simulated the correctness, resilience to network delays, and bandwidth characteristics of the control plane.

The current implementation uses point to point web socket connections between each two nodes; in a N node system there are N^2 + N numbers of web socket connections. While this approach scales for small < 20 number of nodes, it does not scale to higher > 20 number of node systems. Hence, scalable broadcast methods will be explored as the future work.

We use the QSC protocol for the consensus. Evaluating the performance (bandwidth, CPU time) of QSC against the classical consensus protocols such as RAFT, PAXOS is a plausible future work. This will give insight into which consensus algorithm performs better in an asynchronous control plane algorithm.

References

[1] C. Basescu, M. Nowlan, K. Nikitin, J. Faleiro and B. Ford, "Crux: Locality-Preserving Distributed Services", arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1405.0637. [Accessed: 19- Jun- 2020].

[2] Pannatier. A, “A Control Plane in Time and Space for Locality-Preserving Blockchains”, Master Thesis, École Polytechnique Fédérale de Lausanne, 2020

[3] Ford. B, Jovanovic. P, Syta. E, “Que Sera Consensus: Simple Asynchronous Agreement with Private Coins and Threshold Logical Clocks”, arXiv.org, 2020. [Online]. Available: https://arxiv.org/pdf/2003.02291.pdf. [Accessed: 19- Jun- 2020].

[4] "dedis/cothority", GitHub, 2018. [Online]. Available: https://github.com/dedis/cothority. [Accessed: 19- Jun- 2020].

[5] "dedis/onet", GitHub, 2018. [Online]. Available: https://github.com/dedis/onet. [Accessed: 19- Jun- 2020].

[6] "student_20_tlcCtrlPlane/onet", GitHub, 2018. [Online]. Available: https://github.com/dedis/student_20_tlcCtrlPlane/ . [Accessed: 19- Jun- 2020].

[7] "The Go Programming Language", Golang.org, 2017. [Online]. Available: https://golang.org/. [Accessed: 19- Jun- 2020].