HYPERLEDGER

EB 20/21	Enterprise Blockchain Technologies	Number:	1
Module I - Introduction		Issue Date:	
Background: Distributed Systems		Due Date:	

Instructors Guide

This document provides a proposal for a solution for LAB#01, which concerns an introduction to distributed systems and the RAFT consensus algorithm. Most of the solutions can be found at [1–4].

This class aims to illustrate what consensus is and provide intuition on how does it work. For that, we present the RAFT algorithm. However, this class does not present RAFT in a thorough manner. For details, check the paper [1].

1 RAFT

Exercise 1:

To consolidate your knowledge, watch the guided visualization of the RAFT algorithm [3]. Elaborate on how the RAFT algorithm can increase the system's availability from a client perspective.

RAFT is a crash fault-tolerant consensus algorithm that it tolerates when some servers are in a cluster crash. When a client requests the system, the leader disseminates the request, making every follower execute it. If a fault occurs in one of the followers, leading to an inconsistent state (error). Assuming that such error leads to a failure and the system crashes, the whole system's resiliency will not probably fail, as that fault is tolerated by other nodes. This way, the system is more likely to be available.

Exercise 2:

When does a new term start? A new term starts whenever a new leader is elected or a tie at the leader election process.

Exercise 3:

How do candidates act if two of them attempt to become leaders at precisely the same time?

When votes are split in a way, there is no majority of any candidate, the outcome of a leader election is "(c) a period of time goes by with no winner" [1]. In this case, each candidate will time out and start a new election. A new election includes incrementing the term number and to initiate a new round of voting. There is at most one winner per term, and candidates eventually win, as the new timeouts are chosen randomly.

Exercise 4:

If a follower receives a message that has a different term than the one recorded, does it accept the message?

If a message received has a greater term than the term recorded by a node, that node updates its term and becomes a follower. If a message received has a lower term than the recorded.

Exercise 5:

Suppose that a leader receives a command from a client and notifies all followers. Upon sending the second transaction, one of the followers crashes. How does RAFT guarantee that the follower is updated?

After the leader receives a command, the leader propagates that new command information to the followers (uncommitted). Only then he executes the function and returns the result to the client. The followers are then notified, and they execute the command in their state machine. The transaction is considered committed at this point. The follower is updated by following the Log matching property.

Exercise 6: Consider the following RAFT cluster, composed of a leader and four followers. The leader has sent eight messages, corresponding to eight commands. Those eight commands translate into eight log entries. The current term is term three. What are the commands, represented by each log index, that are safe to be executed by each node?

The log entries that are considered to be committed are log entries up to log index 7, because they are replicated across most nodes (3 out of 5). Two servers are out of sync.

Exercise 7:

Consider the following RAFT cluster, composed of a leader and four followers. There are inconsistencies in the logs across nodes. Assume that node S4 crashed on term 1. Node S5 became the leader on term 2, but was only successful in replicating logs 4 and 5, before crashing. Node S3 is the leader in term 3.

RAFT assumes that the current leader is correct (holds the most up-to-date log of commands). S5 will discard its version of logs 6-9, and receive the updated log from S3. S1 and S2 eventually receive logs 8 and 9. As most nodes have the log complete up to log entry 9, the commands present in the log entries can be executed.

Exercise 8: During the leader election process, a mechanism that ensures consistency across logs is the rejection of candidates that have outdated logs. What are the checks the follower does against the candidate's proposal? On which occasion does it reject voting on the candidate?

The node checks if the candidate's last log entry is in the same term as the candidate's last log entry. If the node's last log entry is in a superior term, it does not vote on the candidate. If the terms are the same, and the node's log index is higher (more complete log), it does not vote on the candidate. This process certifies the elected leader has all the committed entries.

Exercise 9: Consider the following RAFT cluster, composed of seven nodes. Given that the leader of term eight has ten log entries and interacts with six followers, how did the system get into this state?

From RAFT's paper: When the leader at the top comes to power, it is possible that any of the scenarios (a–f) could occur in follower logs. Each box represents one log entry; the number in the box is its term. A follower may be missing entries (a–b), may have extra uncommitted entries (c–d), or both (e–f). For example, scenario (f) could occur if that server was the leader for term 2, added several entries to its log, then crashed before committing any of them; it restarted quickly, became leader for term 3, and added a few more entries to its log; before any of the entries in either term 2 or term 3 were committed, the server crashed again and remained down for several terms.

Exercise 10: How does the system re-organize itself if a node leaves the system?

configuration changes must use a two-phase approach: the first phase disables the old consortium configuration, disabling client requests' processing; the second phase enables the new configuration. "In RAFT, the cluster first switches to a transitional configuration we call joint consensus; once the joint consensus has been committed, the system then transitions to the new configuration". For more details, see Section 6 of [1].

References

[1] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *USENIX Annual Technical Conference*, 2014.

[2] J. Ousterhout and D. Ongaro, "Designing for Understandability: The Raft Consensus Algorithm - YouTube," 2016. [Online]. Available: https://www.youtube.com/watch?v=vYp4LYbnnW8

[3] B. Johnson, "Raft - Understandable Distributed Consensus," 2013. [Online]. Available: http://thesecretlivesofdata.com/raft/

[4] RAFT, "Raft Consensus Algorithm," 2016. [Online]. Available: https://raft.github.io/