CS60002 Distributed Systems

Assignment 3: Consensus Module using Raft

Date: March 9, 2023

Target Deadline (For sharing the git repo): April 9th, 2023

In the previous assignment, you implemented a distributed queue system that allowed multiple producers and consumers to interact with a set of brokers with the help of broker managers in order to produce and consume messages. However, the current implementation does not guarantee fault tolerance or consistency in the event of broker node failures.

In this assignment, you will extend the system to include a consensus module using the Raft algorithm. This module will allow the system to continue functioning correctly even in the event of broker node failures or network partitions.

Part I: Familiarize with RAFT [20 marks]

The objective of this part is to familiarize yourself with the RAFT consensus. In this part of the assignment, you will familiarize yourself with RAFT usage on a simple use case of an ATM machine network. Note that you need not implement RAFT protocol on your own, you are free to pick any existing raft implementation. We'll refer to this chosen implementation as the RAFT library or RAFT package hereafter.

Implement a toy ATM network that simulates the behavior of real-world ATMs. Each ATM can be simulated by a process that runs on a separate terminal window, accepts user commands and displays appropriate output. Use an existing RAFT library for consistency of data across these processes.

Each ATM should support the following operations on command line prompt:

1. Withdrawal
2. Deposit
3. Balance inquiry
4. Transfer to other account

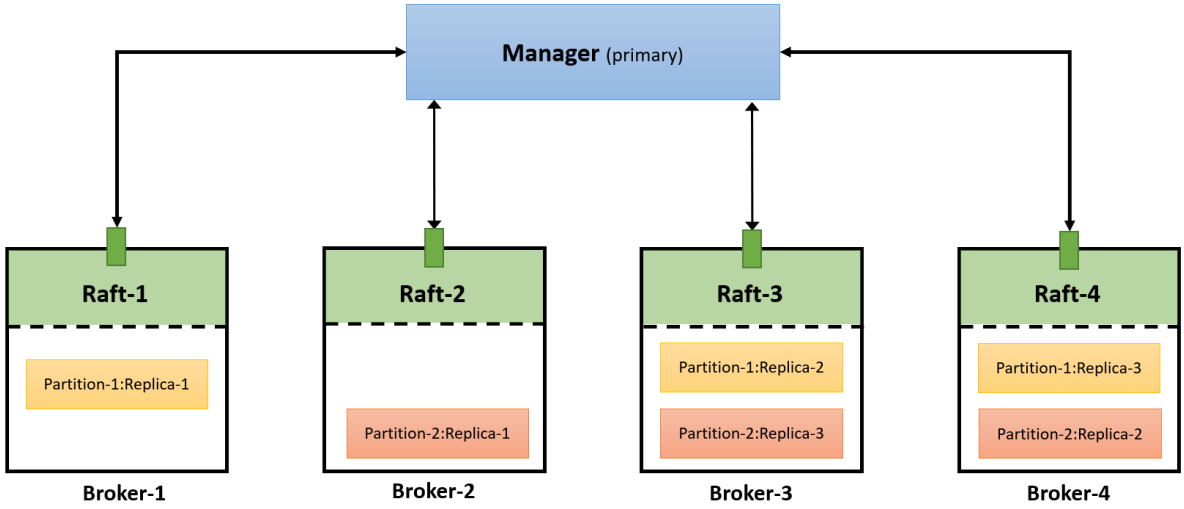
Note: You need **not** implement user authentication. For the purpose of user identification, an ID or a name would suffice.

You can then play around with it and test the following. You implementation should:

- maintain the consistency of user account balances across all ATM instances
- be fault-tolerant and continue to function correctly even if some of the ATMs fail
- be able to handle network partitions (new leader election if needed, etc.)
- sync the raft logs if a node recovers from crash or rejoins network after a partition (try digging into your library to find out how this is done)

Part II: Fault-tolerant Broker [80 marks]

Once you have familiarized yourself with a RAFT implementation, next you'll work on the Broker architecture that you have developed in Assignment 2. In this part of the assignment, you will extend your broker to incorporate a NAT-like design using the RAFT protocol. The goal of this design is to allow for fault-tolerant and consistent replication of partitions across multiple brokers, without the need for additional physical hosts. For simplicity, the replication factor is fixed at 3.



Each broker will have a single RAFT process, bound to one port, that will handle all partition replicas on that broker. For each partition, a separate RAFT instance (object of class Raft) will be maintained within that single process. These objects will maintain a state machine for each partition, including the partition entries themselves. If there are n partition replicas in a broker, then there will be n RAFT objects and a single RAFT process.

Message passing between brokers will be accomplished using TCP sockets. To ensure that messages are correctly routed to their intended RAFT object on the receiving broker, you will need to include the sending raft instance's identifier and encode your message to a byte stream before sending append entry, request vote, etc. and decode it on arrival. Upon receiving a message, the main function of the broker process (in other words the one raft process running on this broker) will pass this append entry/request vote/other message to the appropriate RAFT object.

You will need to handle leader election and ensure log consistency in the system. Additionally, the RAFT log should be persistent, so that it can be recovered on crash-fail of the node.

Please test your implementation thoroughly.

Part III: Come up with a Complete Solution! [Bonus marks to be awarded]

In this part of the assignment, you will come up with a better design for the Broker Manager than what is described so far. The goal is to improve the design, taking into consideration the tradeoffs of each choice made.

You should consider the following questions:

- Is the Broker Manager a single point of failure currently? If yes, how can we avoid that ?
- Would you use Raft for managers as well? If yes, would there be one Raft cluster for all read/write managers or separate clusters for read and write managers?
- Would other read replicas detect the primary manager going down and take its role? Would the secondary have up-to-date information in this case?
- What other design choices can be made to improve the overall system's scalability and reliability while not giving up on the consistency?

After coming up with a better design, try to implement the modified Broker Manager. In addition, you can mention the future scope of extension and how the overall design (the entire distributed queue system, not just the broker manager) can be further improved and scaled. Discuss the shortcomings of the current design and how your modifications address them.

The bonus marks awarded for this part of the assignment would be on a case by case basis and would depend on the actual improvements proposed and implemented by the group.

Deliverables and Grading

The following are some submission and grading guidelines :

1. The source code for Part I which includes the implementation of the ATM and Part II which includes your modified queue system with Raft consensus module.
2. A document describing your design decisions, including how you modified/used the existing Raft library for part II of this assignment and any optimizations you made.
3. All the code and documents are to be submitted via the git repository link as with the previous assignments. The google form for submission of the link would be shared on Piazza in the last week of the deadline.
4. A demo/presentation will be scheduled just after the deadline, more details on this will be announced on Piazza in due time.

Resources

You may find the following resources helpful in completing this assignment:

- The Raft paper by Diego Ongaro and John Ousterhout: https://raft.github.io/raft.pdf
- The Raft website, which includes a list of implementations in various programming languages: https://raft.github.io/
- The Kafka website, which includes information on how Kafka uses replication to achieve fault tolerance: https://kafka.apache.org/documentation/#replication