CSE 586 Project Phase 4: RAFT

Ganesh Nayar Harindranathu Rahul Nair

May 6, 2022

1 Introduction

1.1 RAFT

Raft is a consensus algorithm that is designed to be easy to understand. It's equivalent to Paxos in fault-tolerance and performance. The difference is that it's decomposed into relatively independent subproblems, and it cleanly addresses all major pieces needed for practical systems

1.1.1 Consensus and State machine

Consensus is the process by which a collection of computers works together as a coherent group to ensure overall system-reliability in the face of malfunctioning nodes. This is often accomplished by having nodes coordinate with each other. In our project, we utilize distributed replicated log with consensus to ensure that all state machines get the same log.

State machine is a program that takes in inputs and outputs a result, A log is a collection of operations (client requests) that are to be executed on the state machines, Consensus ensures that the logs are replicated across all nodes as long as all the logs are identical, the state machine will produce the same sequence of results. As long as majority of servers are up, the consensus mechanism can ensure log replication

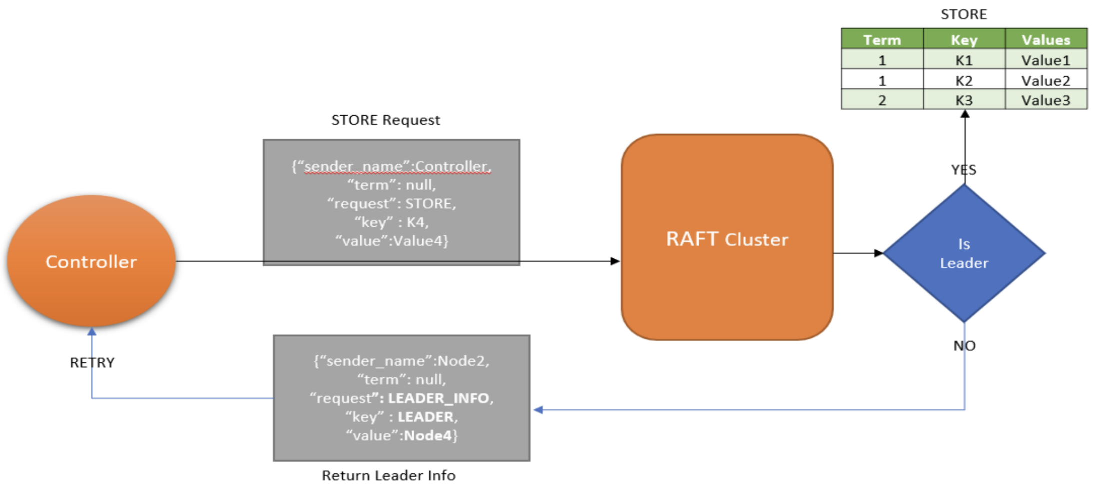
Previously until Phase 3 we have implemented Leader election and AppendEntry RPC, in Phase 4 we will be implementing Safe Log Replication with Leader Election

1.2 Controller Requests

1.2.1 Store

The controller will be used to send out STORE requests to the RAFT cluster. This is similar to the client making a request to the RAFT cluster where the request gets appended to the log's of the leader and eventually gets appended to the follower's logs. You will be using the STORE cmd to make client requests

Figure 1: Store



```
{"sender_name":Controller,
"term": null,
"request": STORE,
"key" : K4,
"value":Value4}
```

```
{"sender_name":Node2,
"term": null,
"request": LEADER_INFO,
"key" : LEADER,
"value":Node4}
```

Term	Key	Values
1	K1	Value1
1	K2	Value2
2	K3	Value3

1.2.2 Retrieve

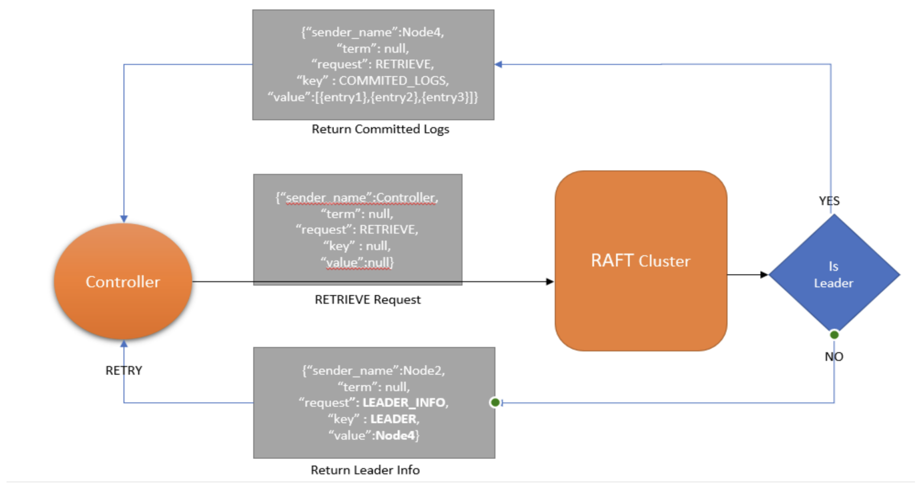
The controller can send a request to any of the nodes to RETRIEVE all the committed entries at that particular node. However, only the Leader will respond with the committed entries[entry1,entry2,entry3], any other node responds with Leader Info as depicted in the diagram below

1.3 Log Replication

In RAFT, the request first gets appended to the log's of each node and once the request has been appended to a majority of the node's logs, the request is said to be committed and will be executed by the leader.

The leader keeps a nextIndex[] for each follower, which is the log entry index that the leader will give to that follower in the upcoming AppendEntry RPC.

Figure 2: Retrieve



```
{"sender_name":Node4,
"term": null,
"request": RETRIEVE,
"key" : COMMITED_LOGS,
"value":[{entry1},{entry2},{entry3}]}
```

```
{"sender_name":Controller,
"term": null,
"request": RETRIEVE,
"key" : null,
"value":null}
```

```
{"sender_name":Node2,
"term": null,
"request": LEADER_INFO,
"key" : LEADER,
"value":Node4}
```

When a leader first comes to power, it initializes all nextIndex[] values in its own log to the index shortly after the last one; this will happen every time a leader is elected.

1.4 Leader Election

We add additional rules to grant positive votes, the voting server denies vote if its own logs are more complete than the candidate

$$(lastterm_v > lastTerm_c) or ((lastTerm_v == lastTerm_c) and (lastIndex_v > lastIndex_c))$$

1.5 Append Reply

When a follower receives an AppendEntry RPC (Note: heartbeats are also AppendEntryRPC), the follower can choose to accept or reject. The follower can:

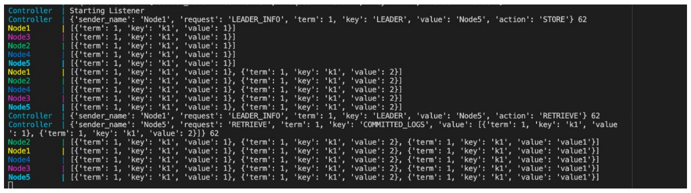
- Reply false if term less than currentTerm
- Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm
- If both the above conditions are not met, the the follower accepts the logs,

appends any new entries to its own logs and sends an Appendreply with success=True

- Also, update the follower node's commitIndex based on the value sent by the leader and the length of the follower's logs

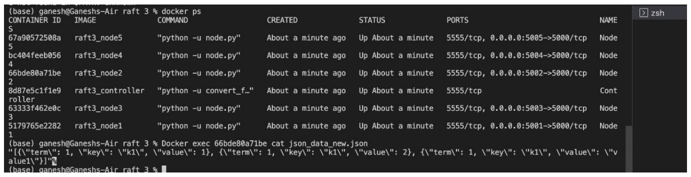
2 Sample Outputs

Figure 3: Store and Retrieve



```
Controller   | Starting Listener
Controller   | {'sender_name': 'Node1', 'request': 'LEADER_INFO', 'term': 1, 'key': 'LEADER', 'value': 'Node5', 'action': 'STORE'} 62
Node1        | [{'term': 1, 'key': 'k1', 'value': 1}]
Node3        | [{'term': 1, 'key': 'k1', 'value': 1}]
Node2        | [{'term': 1, 'key': 'k1', 'value': 1}]
Node4        | [{'term': 1, 'key': 'k1', 'value': 1}]
Node5        | [{'term': 1, 'key': 'k1', 'value': 1}]
Node1        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}]
Node2        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}]
Node4        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}]
Node3        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}]
Node5        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}]
Controller   | {'sender_name': 'Node1', 'request': 'LEADER_INFO', 'term': 1, 'key': 'LEADER', 'value': 'Node5', 'action': 'RETRIEVE'} 62
Controller   | {'sender_name': 'Node5', 'request': 'RETRIEVE', 'term': 1, 'key': 'COMMITTED_LOGS', 'value': [{'term': 1, 'key': 'k1', 'value
': 1}, {'term': 1, 'key': 'k1', 'value': 2}]} 62
Node2        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}, {'term': 1, 'key': 'k1', 'value': 'value1'}]
Node1        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}, {'term': 1, 'key': 'k1', 'value': 'value1'}]
Node4        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}, {'term': 1, 'key': 'k1', 'value': 'value1'}]
Node3        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}, {'term': 1, 'key': 'k1', 'value': 'value1'}]
Node5        | [{'term': 1, 'key': 'k1', 'value': 1}, {'term': 1, 'key': 'k1', 'value': 2}, {'term': 1, 'key': 'k1', 'value': 'value1'}]
```

Figure 4: Data Persistence



```
(base) ganesh@Ganeshs-Air raft 3 % docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED              STATUS              PORTS                             NAME
S
67a90572508a   raft3_node5        "python -u node.py"      About a minute ago   Up About a minute   5555/tcp, 0.0.0.0:5005->5000/tcp   Node
5
bc404feeb056   raft3_node4        "python -u node.py"      About a minute ago   Up About a minute   5555/tcp, 0.0.0.0:5004->5000/tcp   Node
4
66bde80a71be   raft3_node2        "python -u node.py"      About a minute ago   Up About a minute   5555/tcp, 0.0.0.0:5002->5000/tcp   Node
2
8d87e5c1f1e9   raft3_controller   "python -u convert_f…"   About a minute ago   Up About a minute   5555/tcp                           Cont
roller
63333f462e0c   raft3_node3        "python -u node.py"      About a minute ago   Up About a minute   5555/tcp, 0.0.0.0:5003->5000/tcp   Node
3
5179765e2282   raft3_node1        "python -u node.py"      About a minute ago   Up About a minute   5555/tcp, 0.0.0.0:5001->5000/tcp   Node
1
(base) ganesh@Ganeshs-Air raft 3 % Docker exec 66bde80a71be cat json_data_new.json
"[{\"term\": 1, \"key\": \"k1\", \"value\": 1}, {\"term\": 1, \"key\": \"k1\", \"value\": 2}, {\"term\": 1, \"key\": \"k1\", \"value\": \"v
alue1\"}]"%
(base) ganesh@Ganeshs-Air raft 3 %
```

3 References

- UBLearns CSE 586: Phase 4 Handout