CADP - Log replication in Raft

Alessandro Monticelli, Jóhannes Helgi Tómasson, Jonas Stahl April 3, 2026

Description of solution

We are shown the entry log of 5 servers, and are tasked with showing the message exchanges as the servers update their logs to match the leader. The initial server logs are listed as follows:

```
Log Index:                    1 2 3 4 5 6 7 8 9
Leader for term 8:          [ 1 1 2 3 4 6 6     ]
(a):                        [ 1 1 2 3 4 6       ]
(b):                        [ 1 1 2 3 5         ]
(c):                        [ 1 1 2 3 4 6 6 7   ]
(d):                        [ 1 1 2 2 2 2       ]
```

When a server becomes leader it initializes a new **nextIndex**[] and **matchIndex**[] set to the next empty index (8) and 0 respectively for each follower. NextIndex will keep track of the index each follower is currently seeking, while matchIndex is set to the last known index confirmed to match the leaders entry. The leader begins sending heartbeats in the form of empty *AppendEntriesRequests* (AERq) which contain information about the previous entries index and term. The followers will send *AppendEntriesResponses* (AERp) based on whether the information provided is in accordance to their logs.

Server (a):

A heartbeat is received with Term = 8 pIndex = 7 and pTerm = 6. The server updates its term to 8 but sees pIndex is greater than the length of its log. The server sends back AERp with *Success: false*. The leader decrements nextIndex for (a) from 8 to 7 and sends the entry for index 7. Server (a) then sees pIndex and pTerm match its own log and appends index entry 7 to its own log and sends AERp with *Success: true*. The leader sets nextIndex[a] = 8 and matchIndex[a] = 7

Server (b):

As with (a), (b) sends back an AERp with *Success: false*, Leader has to decrement nextIndex three times in total. On the second attempt, pIndex = 5, matches for server (b) but pTerm is different $4 \neq 5$. Leader decrements nextIndex and tries again, sending entries for indexes 5-7. (b) replaces its entry in index 5, appends entry indexes 6 and 7 and sends AERp *Success: true*. The leader then sets nextIndex[b] = 8 and matchIndex[b] = 7

Server (c):

Server (c) has an extraneous entry. When (c) receives the heartbeat, it will match the index to its log and sends AERp *Success: true*, but it won't affect the log at index 8. The Raft algorithm will only delete entries when a conflict has been detected. As such the entry in index 8 will only be deleted when the next non-heartbeat AERq is sent with term 8. The leader will set nextIndex[c] = 8 and matchIndex[c] = 7 after receiving the AERp.

Server (d):

As with (a) and (b), the leader will decrement nextIndex to 4 and send all of its entries from indexes 4-7. (d) will match the previous index and term as valid and clear all of its entries after index 3, and append entry indexes 4-7, return AERp *Success: true*. The leader then sets nextIndex[d] = 8 and matchIndex[d] = 7.

conclusion:

By the end, all followers' logs will match the leader's up to index 7, with (c) still carrying its extraneous entry. The leader's nextIndex and matchIndex are set to 8 and 7 respectively for all followers.

```
Final result:
Log Index:                    1 2 3 4 5 6 7 8 9
Leader for term 8:          [ 1 1 2 3 4 6 6     ]
(a):                        [ 1 1 2 3 4 6 6     ]
(b):                        [ 1 1 2 3 4 6 6     ]
(c):                        [ 1 1 2 3 4 6 6 7   ]
(d):                        [ 1 1 2 3 4 6 6     ]
```