Raft

11/9/18
Raft

System for enforcing strong consistency (linearizability)

Similar to Paxos and Viewstamped Replication, but much simpler

  Clear boundary between leader election and consensus

  Leader log is ground truth; log entries only flow in one direction
Assignment 3 hints

You will implement the *leader election* portion of Raft in assignment 3.
You will implement the *log replication* portion of Raft in assignment 4.

Use `time.Timer` and `select` statements to implement timeout:
- Need to time out on heartbeats → Start election
- Need to time out on waiting for majority of votes

Raft logs are 1-indexed; add a dummy entry in the first slot to enforce this.

When voting for yourself, you can skip the RPC.
Importance of **readability**

A luxury for small projects, but a necessity for large and complex projects

HW4 will build on top of your solution for HW3
HW3 only accounts for about 20% of the work

Some tips:
- Duplicate code is *really* bad; avoid at all costs
- If a function is more than 30 lines, it is too long → split!
- Avoid nested if-else’s; use returns and continues where possible
Raft
Leader election
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentTerm</td>
<td>latest term server has seen</td>
</tr>
<tr>
<td>votedFor</td>
<td>candidate ID that received vote in current term, or -1 if none</td>
</tr>
<tr>
<td>commitIndex</td>
<td>index of highest log entry known to be committed</td>
</tr>
<tr>
<td>lastApplied</td>
<td>index of highest log entry applied to state machine</td>
</tr>
<tr>
<td>nextIndex</td>
<td>for each server, index of the next log entry to send to that server</td>
</tr>
<tr>
<td>matchIndex</td>
<td>for each server, index of highest log entry known to be replicated on the server</td>
</tr>
</tbody>
</table>

Logs are 1-indexed

(Only on leader)
| 0 | currentTerm | 0 |
|   | votedFor    | -1 |

State required for election

- **currentTerm**: Latest term server has seen
- **votedFor**: Candidate ID that received vote in current term, or -1 if none
Leader election

Everyone sets a randomized timer that expires in $[T, 2T]$ (e.g. $T = 150$ms)

When timer expires, increment term and send a RequestVote to everyone

Retry this until either:

- You get majority of votes (including yourself): become leader
- You receive an RPC from a valid leader: become follower again
Timeout
RequestVote
Term: 1
CandidateID: 0
LastLogIndex: -1
LastLogTerm: -1
RequestVoteReply
Term: 1
VoteGranted: true
Node 0:
- currentTerm: 1
- votedFor: 0

Node 1:
- currentTerm: 1
- votedFor: 0

Node 2:
- currentTerm: 1
- votedFor: 0

<empty>
Suppose there are existing log entries...
currentTerm 3
votedFor 1

Timeout

currentTerm 3
votedFor 1

currentTerm 3
votedFor 1

currentTerm 3
votedFor 1

currentTerm 3
votedFor 1

1 1 1 2 3

1 1 1 2 3

1 1 1 2 3

1 1 1 2 3
RequestVote
Term: 4
CandidateID: 0
LastLogIndex: 5
LastLogTerm: 3
1 currentTerm 4
votedFor 0

2 currentTerm 4
votedFor 0

RequestVoteReply
Term: 4
VoteGranted: True
Conditions for granting vote

1. We did not vote for anyone else in this term
2. Candidate term must be $\geq$ ours
3. Candidate log is at least as up-to-date as ours
   a. The log with higher term in the last entry is more up-to-date
   b. If the last entry terms are the same, then the longer log is more up-to-date
Which one is more *up-to-date*?
Which one is more *up-to-date*?
Which one is more *up-to-date*?

1 1 1 2 3

1 1 4 ✅
Why reject logs that are not *up-to-date*?

Leader log is always the ground truth

Once someone is elected leader, followers must throw away conflicting entries

Must NOT throw away committed entries!

*Note: Log doesn’t need to be the MOST up-to-date among all servers*
What if we accept logs that are not as up-to-date as ours?
Suppose entries 4-5 have already been committed.

Then previous leader S0 crashes and S3 times out.

If S3 becomes leader then committed entries 4 and 5 may be overwritten!
Why is it OK to throw away these entries?

If these entries were committed, then it means they must exist on a majority of servers.

In that case S4 can receive votes from the same majority and become a valid leader.
One caveat with entries from old terms…
(later)
Raft

Normal operation
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentTerm</td>
<td>latest term server has seen</td>
</tr>
<tr>
<td>votedFor</td>
<td>candidate ID that received vote in current term, or -1 if none</td>
</tr>
<tr>
<td>commitIndex</td>
<td>index of highest log entry known to be committed</td>
</tr>
<tr>
<td>lastApplied</td>
<td>index of highest log entry applied to state machine</td>
</tr>
<tr>
<td>nextIndex</td>
<td>for each server, index of the next log entry to send to that server</td>
</tr>
<tr>
<td>matchIndex</td>
<td>for each server, index of highest log entry known to be replicated on the server</td>
</tr>
</tbody>
</table>

*(Only on leader)*

Logs are 1-indexed
0 currentTerm 0
votedFor -1
commitIndex 0
lastApplied 0
nextIndex []
matchIndex []

<empty>

1 currentTerm 0
votedFor -1
commitIndex 0
lastApplied 0
nextIndex []
matchIndex []

<empty>

2 currentTerm 0
votedFor -1
commitIndex 0
lastApplied 0
nextIndex []
matchIndex []

<empty>
AppendEntries
Term: 1
LeaderID: 0
PrevLogIndex: 0
PrevLogTerm: -1
LeaderCommit: 0

1
currentTerm 1
votedFor 0
commitIndex 0
lastApplied 0
nextIndex [0, 0, 0]
matchIndex []

2
currentTerm 1
votedFor 0
commitIndex 0
lastApplied 0
nextIndex []
matchIndex []
AppendEntriesReply
Term: 1
Success: True

<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[1, 1, 1]</td>
<td>[0, 0, 0]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>
while commitIndex > lastApplied, apply commands to state machine

Entry 3 is now replicated on a majority, so we can commit it.

<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>[4, 4, 4]</td>
<td>[3, 3, 3]</td>
</tr>
</tbody>
</table>

1 1 1
Once leader has applied an entry to state machine, it is safe to tell the client that the entry is committed.
Raft

After new leader election
Partition!

Timeout

0

- `currentTerm`: 1
- `votedFor`: 0
- `commitIndex`: 3
- `lastApplied`: 3
- `nextIndex`: [4, 4, 4]
- `matchIndex`: [3, 3, 3]

1

- `currentTerm`: 1
- `votedFor`: 0
- `commitIndex`: 0
- `lastApplied`: 0
- `nextIndex`: []
- `matchIndex`: []

2

- `currentTerm`: 1
- `votedFor`: 0
- `commitIndex`: 0
- `lastApplied`: 0
- `nextIndex`: []
- `matchIndex`: []
AppendEntries
Term: 2
LeaderID: 1
PrevLogIndex: 3
PrevLogTerm: 1
LeaderCommit: 0
<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>[4, 4, 4]</td>
<td>[3, 3, 3]</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>[4, 4, 4]</td>
<td>[0, 3, 0]</td>
</tr>
</tbody>
</table>

AppendEntriesReply
Term: 2
Success: True
AppendEntries
Term: 2
LeaderID: 1
PrevLogIndex: 3
PrevLogTerm: 1
LeaderCommit: 3

0
currentTerm 1
votedFor 0
commitIndex 3
lastApplied 3
nextIndex [4, 4, 4]
machine 3, 3, 3

1
currentTerm 2
votedFor 1
commitIndex 3
lastApplied 3
nextIndex [4, 4, 4]
machine [0, 3, 3]

2
currentTerm 2
votedFor 1
commitIndex 0
lastApplied 0
nextIndex []
machine []
currentTerm 1
votedFor 0
commitIndex 3
lastApplied 3
nextIndex [4, 4, 4]
matchIndex [3, 3, 3]

1 currentTerm 2
votedFor 1
commitIndex 3
lastApplied 3
nextIndex [4, 4, 4]
matchIndex [0, 3, 3]

2 currentTerm 2
votedFor 1
commitIndex 3
lastApplied 3
nextIndex []
matchIndex []
Committing entries in the new term...
Let's fix the partition...
<table>
<thead>
<tr>
<th>State</th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>[4, 4, 4]</td>
<td>[3, 3, 3]</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[4, 6, 6]</td>
<td>[0, 5, 5]</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>

**AppendEntries**
- Term: 1
- LeaderID: 0
- PrevLogIndex: 3
- PrevLogTerm: 1
- LeaderCommit: 3

**AppendEntries**
- Term: 1
- LeaderID: 0
- PrevLogIndex: 3
- PrevLogTerm: 1
- LeaderCommit: 3
<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>[4, 4, 4]</td>
<td>[3, 3, 3]</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[4, 6, 6]</td>
<td>[0, 5, 5]</td>
</tr>
</tbody>
</table>

**AppendEntriesReply**

- Term: 2
- **Success: false**

Rejected request because local term is higher (2 > 1)
Old leader is dethroned!
AppendEntries
Term: 2
LeaderID: 1
PrevLogIndex: 3
PrevLogTerm: 1
LeaderCommit: 5

0
currentTerm 2
votedFor -1
commitIndex 3
lastApplied 3
nextIndex []
matchIndex []

1
currentTerm 2
votedFor 1
commitIndex 5
lastApplied 5
nextIndex [4, 6, 6]
matchIndex [0, 5, 5]

2
currentTerm 2
votedFor 1
commitIndex 5
lastApplied 5
nextIndex []
matchIndex []
AppendEntriesReply
Term: 2
Success: true

0
- currentTerm: 2
- votedFor: -1
- commitIndex: 5
- lastApplied: 5
- nextIndex: []
- matchIndex: []

1
- currentTerm: 2
- votedFor: 1
- commitIndex: 5
- lastApplied: 5
- nextIndex: [4, 6, 6]
- matchIndex: [0, 5, 5]

2
- currentTerm: 2
- votedFor: 1
- commitIndex: 5
- lastApplied: 5
- nextIndex: []
- matchIndex: []
Everyone is on the same page again
When log entries don’t match...
prevLogIndex = 5
S1 log[5] = 4
S2 log[5] = 2
Mismatch!
currentTerm: 5
votedFor: 1
commitIndex: 5
lastApplied: 5
nextIndex: []
machIndex: []

AppendEntriesReply
Term: 5
Success: False
prevLogIndex = 4
S1 log[4] = 3
S2 log[4] = 2
Mismatch!
prevLogIndex = 3
S1 log[3] = 1
S2 log[3] = 1

Match!
<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td></td>
<td>1 1 1 3 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
AppendEntriesReply
Term: 5
Success: True
```
Everyone is on the same page again
When log entries don’t match...
When log entries don’t match...

- The leader will find the latest log entry in the follower where the two logs agree
- At the follower:
  - Everything after that entry will be deleted
  - The leader’s log up to that point will be replicated on the follower
Optimization to reduce number of messages?
<table>
<thead>
<tr>
<th>Node</th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[6, 6, 6]</td>
<td>[5, 5, 0]</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>

**AppendEntries**
- Term: 5
- LeaderID: 1
- PrevLogIndex: 5
- PrevLogTerm: 4
- LeaderCommit: 5
Specify index of first log entry in the new term.

```
0  currentTerm 5
   votedFor 1
   commitIndex 5
   lastApplied 5
   nextIndex []
   matchIndex []

1  currentTerm 5
   votedFor 1
   commitIndex 5
   lastApplied 5
   nextIndex [6, 6, 6]
   matchIndex [5, 5, 0]

2  currentTerm 5
   votedFor -1
   commitIndex 3
   lastApplied 3
   nextIndex []
   matchIndex []

AppendEntriesReply
Term: 5
Success: False
RequestedIndex: 4
```
AppendEntries
Term: 5
LeaderID: 1
PrevLogIndex: 3
PrevLogTerm: 1
LeaderCommit: 5

currentTerm 5
votedFor 1
commitIndex 5
lastApplied 5
nextIndex []
matchIndex []

currentTerm 5
votedFor 1
commitIndex 5
lastApplied 5
nextIndex [6, 6, 4]
matchIndex [5, 5, 0]
<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[6, 6, 6]</td>
<td>[5, 5, 5]</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>-1</td>
<td>5</td>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Decrement nextIndex one term at a time
Conditions for committing an entry

1. The entry exists on a majority AND it is written in the current term
2. The entry precedes another entry that is committed
Caveat for committing old entries

Can’t assume an old entry has been committed *even if* it exists on a majority

S1 is the leader

S1.log[2] is only partially replicated...
Caveat for committing old entries

Can’t assume an old entry has been committed even if it exists on a majority

S1 crashes, S5 becomes leader
Caveat for committing old entries

Can’t assume an old entry has been committed *even if* it exists on a majority

S1 crash,
S1 becomes leader

S1.log[2] is now replicated to a majority

S5 crashes,
S1 becomes leader
Caveat for committing old entries

Can’t assume an old entry has been committed *even if* it exists on a majority

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>S5</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

S1 crashes,
S5 becomes leader

S5 replicates S5.log[2] to all other nodes...
Caveat for committing old entries

Can’t assume an old entry has been committed *even if* it exists on a majority

Entry 2 was overwritten even though it was replicated on a majority!

Cannot assume entry 2 was committed
Caveat for committing old entries

Can’t assume an old entry has been committed *even if* it exists on a majority

Entry 2 is committed once entry 3 is committed

Commit old entries *indirectly*

S1 commits entry 3