Progress

Progress represents a follower's progress in the view of the leader. Leader maintains progresses of all followers, and sends `replication message` to the follower based on its progress.

`replication message` is a `msgApp` with log entries.

A progress has two attribute: `match` and `next` . `match` is the index of the highest known matched entry. If leader knows nothing about follower's replication status, `match` is set to zero. `next` is the index of the first entry that will be replicated to the follower. Leader puts entries from `next` to its latest one in next `replication message` .

A progress is in one of the three state: `probe` , `replicate` , `snapshot` .

```
                            +---------------------------------------------------------+
                            |                  send snapshot                          |
                            |                                                         |
                  +---------+----------+                                  +----------v---------
+
              +--->       probe        |                                  |      snapshot
|
              |   |  max inflight = 1  <-----------------------------------+  max inflight = 0
|
              |   +---------+----------+                                  +--------------------
+
              |             |            1. snapshot success
              |             |               (next=snapshot.index + 1)
              |             |            2. snapshot failure
              |             |               (no change)
              |             |            3. receives
msgAppResp(rej=false&&index>lastsnap.index)
              |             |               (match=m.index,next=match+1)
receives msgAppResp(rej=true)
(next=match+1)|             |
              |             |
              |             |
              |             |   receives msgAppResp(rej=false&&index>match)
              |             |   (match=m.index,next=match+1)
              |             |
              |             |
              |             |
              |   +---------v----------+
              |   |     replicate      |
              +---+  max inflight = n  |
                  +--------------------+
```

When the progress of a follower is in `probe` state, leader sends at most one `replication message` per heartbeat interval. The leader sends `replication message` slowly and probing the actual progress of the follower. A `msgHeartbeatResp` or a `msgAppResp` with reject might trigger the sending of the next `replication message` .

When the progress of a follower is in `replicate` state, leader sends `replication message` , then optimistically increases `next` to the latest entry sent. This is an optimized state for fast replicating log entries to the follower.

When the progress of a follower is in `snapshot` state, leader stops sending any `replication message` .

A newly elected leader sets the progresses of all the followers to `probe` state with `match` = 0 and `next` = last index. The leader slowly (at most once per heartbeat) sends `replication message` to the follower and probes its progress.

A progress changes to `replicate` when the follower replies with a non-rejection `msgAppResp`, which implies that it has matched the index sent. At this point, leader starts to stream log entries to the follower fast. The progress will fall back to `probe` when the follower replies a rejection `msgAppResp` or the link layer reports the follower is unreachable. We aggressively reset `next` to `match`+1 since if we receive any `msgAppResp` soon, both `match` and `next` will increase directly to the `index` in `msgAppResp`. (We might end up with sending some duplicate entries when aggressively reset `next` too low. see open question)

A progress changes from `probe` to `snapshot` when the follower falls very far behind and requires a snapshot. After sending `msgSnap`, the leader waits until the success, failure or abortion of the previous snapshot sent. The progress will go back to `probe` after the sending result is applied.

Flow Control

1. limit the max size of message sent per message. Max should be configurable. Lower the cost at probing state as we limit the size per message; lower the penalty when aggressively decreased to a too low `next`

2. limit the # of in flight messages < N when in `replicate` state. N should be configurable. Most implementation will have a sending buffer on top of its actual network transport layer (not blocking raft node). We want to make sure raft does not overflow that buffer, which can cause message dropping and triggering a bunch of unnecessary resending repeatedly.