ongardie / raft.tla

Branch: master ▾ raft.tla / raft.tla Find file Copy path

ongardie Fix AppendEntries to only send one entry at a time 8098acb on 3 Dec 2014

1 contributor

485 lines (433 sloc) 20.1 KB

```
1  ---------------------------------- MODULE raft ----------------------------------
2  \* This is the formal specification for the Raft consensus algorithm.
3  \*
4  \* Copyright 2014 Diego Ongaro.
5  \* This work is licensed under the Creative Commons Attribution-4.0
6  \* International License https://creativecommons.org/licenses/by/4.0/
7
8  EXTENDS Naturals, FiniteSets, Sequences, TLC
9
10 \* The set of server IDs
11 CONSTANTS Server   实际上多个servr，但不用自己指定几个，叫什么名字等
12
13 \* The set of requests that can go into the log
14 CONSTANTS Value
15
16 \* Server states.
17 CONSTANTS Follower, Candidate, Leader
18
19 \* A reserved value.
20 CONSTANTS Nil
21
22 \* Message types:
23 CONSTANTS RequestVoteRequest, RequestVoteResponse,
24           AppendEntriesRequest, AppendEntriesResponse
25
26 ----
27 \* Global variables
28
29 \* A bag of records representing requests and responses sent from one server
30 \* to another. TLAPS doesn't support the Bags module, so this is a function
31 \* mapping Message to Nat.
32 VARIABLE messages  TLAPS是啥？ TLA+ book里面提到了Bags模块，是个multi set
33                   这个变量实际上应该是二维map
34 \* A history variable used in the proof. This would not be present in an
35 \* implementation.
36 \* Keeps track of successful elections, including the initial logs of the
37 \* leader and voters' logs. Set of functions containing various things about
38 \* successful elections (see BecomeLeader).
39 VARIABLE elections
40
41 \* A history variable used in the proof. This would not be present in an
42 \* implementation.
43 \* Keeps track of every log ever in the system (set of logs).
44 VARIABLE allLogs
45
46 ----                              下面是每个server一份的变量。用下标方式使用
47 \* The following variables are all per server (functions with domain Server).
48
49 \* The server's term number.    这些实际上只有currentTerm，votedFor持久化。其他的在restart后丢失了。参见restart()
50 VARIABLE currentTerm
51 \* The server's state (Follower, Candidate, or Leader).
52 VARIABLE state
53 \* The candidate the server voted for in its current term, or
54 \* Nil if it hasn't voted for any.
55 VARIABLE votedFor
```

```
56  serverVars == <<currentTerm, state, votedFor>>
57
58  \* A Sequence of log entries. The index into this sequence is the index of the
59  \* log entry. Unfortunately, the Sequence module defines Head(s) as the entry
60  \* with index 1, so be careful not to use that!
61  VARIABLE log
62  \* The index of the latest entry in the log the state machine may apply.
63  VARIABLE commitIndex 为什么这个可以不持久化？
64  logVars == <<log, commitIndex>>
65
66  \* The following variables are used only on candidates: 实际上在推选自己当leader期间用
67  \* The set of servers from which the candidate has received a RequestVote
68  \* response in its currentTerm.
69  VARIABLE votesResponded
70  \* The set of servers from which the candidate has received a vote in its
71  \* currentTerm.
72  VARIABLE votesGranted
73  \* A history variable used in the proof. This would not be present in an
74  \* implementation.
75  \* Function from each server that voted for this candidate in its currentTerm
76  \* to that voter's log.
77  VARIABLE voterLog
78  candidateVars == <<votesResponded, votesGranted, voterLog>>
79
80  \* The following variables are used only on leaders: 仅仅leader维护的内存变量
81  \* The next entry to send to each follower.
82  VARIABLE nextIndex
83  \* The latest entry that each follower has acknowledged is the same as the
84  \* leader's. This is used to calculate commitIndex on the leader.
85  VARIABLE matchIndex
86  leaderVars == <<nextIndex, matchIndex, elections>>
87
88  \* End of per server variables.
89  ----
90                                                                这个是为了表示方便，比如vars'
91  \* All variables; used for stuttering (asserting state hasn't changed). 那些不变的，必须明确说unchanged
92  vars == <<messages, allLogs, serverVars, candidateVars, leaderVars, logVars>>
93
94  ----
95  \* Helpers
96
97  \* The set of all quorums. This just calculates simple majorities, but the only
98  \* important property is that every quorum overlaps with every other.
99  Quorum == {i \in SUBSET(Server) : Cardinality(i) * 2 > Cardinality(Server)} 这个定义，只要server个数>=0，还是你能表达原意的
100
101 \* The term of the last entry in a log, or 0 if the log is empty.
102 LastTerm(xlog) == IF Len(xlog) = 0 THEN 0 ELSE xlog[Len(xlog)].term
103
104 \* Helper for Send and Reply. Given a message m and bag of messages, return a
105 \* new bag of messages with one more m in it.
106 WithMessage(m, msgs) ==          这个\in DOMAIN，与\in不是一码事，啥含义呢？ A function f has a domain, written domain f , and it
107     IF m \in DOMAIN msgs THEN    assigns to eac h element x of its domain the value f [ x ]
108         [msgs EXCEPT ![m] = msgs[m] + 1]  也就是说，msgs是个函数，m 属于这个函数的domain.实际上，就是m可以作为下标使用，或者映射的key使用
109     ELSE
110         msgs @@ (m :> 1)
111
112 \* Helper for Discard and Reply. Given a message m and bag of messages, return
113 \* a new bag of messages with one less m in it.
114 WithoutMessage(m, msgs) ==
115     IF m \in DOMAIN msgs THEN
116         [msgs EXCEPT ![m] = msgs[m] - 1]  这样不会减为0么？却没有删除掉！
117     ELSE
118         msgs
119
120 \* Add a message to the bag of messages.
121 Send(m) == messages' = WithMessage(m, messages)
122
123 \* Remove a message from the bag of messages. Used when a server is done
```

```
124 \* processing a message.
125 Discard(m) == messages' = WithoutMessage(m, messages)
126
127 \* Combination of Send and Discard
128 Reply(response, request) ==
129     messages' = WithoutMessage(request, WithMessage(response, messages))
130
131 \* Return the minimum value from a set, or undefined if the set is empty.
132 Min(s) == CHOOSE x \in s : \A y \in s : x <= y
133 \* Return the maximum value from a set, or undefined if the set is empty.
134 Max(s) == CHOOSE x \in s : \A y \in s : x >= y
135
136 ----
137 \* Define initial values for all variables
138
139 InitHistoryVars == /\ elections = {}
140                    /\ allLogs   = {}
141                    /\ voterLog  = [i \in Server |-> [j \in {} |-> <<>>]]
142 InitServerVars == /\ currentTerm = [i \in Server |-> 1]
143                   /\ state       = [i \in Server |-> Follower]
144                   /\ votedFor    = [i \in Server |-> Nil]
145 InitCandidateVars == /\ votesResponded = [i \in Server |-> {}]
146                      /\ votesGranted   = [i \in Server |-> {}]
147 \* The values nextIndex[i][i] and matchIndex[i][i] are never read, since the
148 \* leader does not send itself messages. It's still easier to include these
149 \* in the functions.
150 InitLeaderVars == /\ nextIndex  = [i \in Server |-> [j \in Server |-> 1]]
151                   /\ matchIndex = [i \in Server |-> [j \in Server |-> 0]]
152 InitLogVars == /\ log          = [i \in Server |-> << >>]
153                /\ commitIndex  = [i \in Server |-> 0]
154 Init == /\ messages = [m \in {} |-> 0]    用{}，告诉系统，这是个map?
155         /\ InitHistoryVars
156         /\ InitServerVars
157         /\ InitCandidateVars
158         /\ InitLeaderVars
159         /\ InitLogVars
160
161 ----
162 \* Define state transitions
163
164 \* Server i restarts from stable storage.
165 \* It loses everything but its currentTerm, votedFor, and log.
166 Restart(i) ==
167     /\ state'          = [state EXCEPT ![i] = Follower]
168     /\ votesResponded' = [votesResponded EXCEPT ![i] = {}]
169     /\ votesGranted'   = [votesGranted EXCEPT ![i] = {}]
170     /\ voterLog'       = [voterLog EXCEPT ![i] = [j \in {} |-> <<>>]]
171     /\ nextIndex'      = [nextIndex EXCEPT ![i] = [j \in Server |-> 1]]
172     /\ matchIndex'     = [matchIndex EXCEPT ![i] = [j \in Server |-> 0]]
173     /\ commitIndex'    = [commitIndex EXCEPT ![i] = 0]
174     /\ UNCHANGED <<messages, currentTerm, votedFor, log, elections>>
175
176 \* Server i times out and starts a new election.
177 Timeout(i) == /\ state[i] \in {Follower, Candidate}    Leader，不会发生超时？ 关键是发现也没啥用，已经是leader
178               /\ state' = [state EXCEPT ![i] = Candidate]
179               /\ currentTerm' = [currentTerm EXCEPT ![i] = currentTerm[i] + 1]
180               \* Most implementations would probably just set the local vote
181               \* atomically, but messaging localhost for it is weaker.
182               /\ votedFor' = [votedFor EXCEPT ![i] = Nil]
183               /\ votesResponded' = [votesResponded EXCEPT ![i] = {}]
184               /\ votesGranted'   = [votesGranted EXCEPT ![i] = {}]
185               /\ voterLog'       = [voterLog EXCEPT ![i] = [j \in {} |-> <<>>]]
186               /\ UNCHANGED <<messages, leaderVars, logVars>>
187
188 \* Candidate i sends j a RequestVote request.
189 RequestVote(i, j) ==
190     /\ state[i] = Candidate  Candidate状态，是前提
191     /\ j \notin votesResponded[i]
```

```
192     /\ Send([mtype         |-> RequestVoteRequest,
193              mterm         |-> currentTerm[i],
194              mlastLogTerm  |-> LastTerm(log[i]),
195              mlastLogIndex |-> Len(log[i]),
196              msource       |-> i,
197              mdest         |-> j])
198     /\ UNCHANGED <<serverVars, candidateVars, leaderVars, logVars>>
199
200 \* Leader i sends j an AppendEntries request containing up to 1 entry.
201 \* While implementations may want to send more than 1 at a time, this spec uses
202 \* just 1 because it minimizes atomic regions without loss of generality.
203 AppendEntries(i, j) ==
204     /\ i /= j
205     /\ state[i] = Leader
206     /\ LET prevLogIndex == nextIndex[i][j] - 1
207            prevLogTerm == IF prevLogIndex > 0 THEN
208                               log[i][prevLogIndex].term
209                           ELSE
210                               0
211            \* Send up to 1 entry, constrained by the end of the log.
212            lastEntry == Min({Len(log[i]), nextIndex[i][j]})
213            entries == SubSeq(log[i], nextIndex[i][j], lastEntry)
214        IN Send([mtype          |-> AppendEntriesRequest,
215                 mterm          |-> currentTerm[i],
216                 mprevLogIndex  |-> prevLogIndex,
217                 mprevLogTerm   |-> prevLogTerm,
218                 mentries       |-> entries,
219                 \* mlog is used as a history variable for the proof.
220                 \* It would not exist in a real implementation.
221                 mlog           |-> log[i],
222                 mcommitIndex   |-> Min({commitIndex[i], lastEntry}),
223                 msource        |-> i,
224                 mdest          |-> j])
225     /\ UNCHANGED <<serverVars, candidateVars, leaderVars, logVars>>
226
227 \* Candidate i transitions to leader.
228 BecomeLeader(i) ==
229     /\ state[i] = Candidate
230     /\ votesGranted[i] \in Quorum
231     /\ state'      = [state EXCEPT ![i] = Leader]
232     /\ nextIndex'  = [nextIndex EXCEPT ![i] =
233                          [j \in Server |-> Len(log[i]) + 1]]
234     /\ matchIndex' = [matchIndex EXCEPT ![i] =
235                          [j \in Server |-> 0]]
236     /\ elections'  = elections \cup
237                          {[eterm     |-> currentTerm[i],
238                            eleader   |-> i,
239                            elog      |-> log[i],
240                            evotes    |-> votesGranted[i],
241                            evoterLog |-> voterLog[i]]}
242     /\ UNCHANGED <<messages, currentTerm, votedFor, candidateVars, logVars>>
243
244 \* Leader i receives a client request to add v to the log.
245 ClientRequest(i, v) ==
246     /\ state[i] = Leader     如果发的不是leader，直接无视了？不告诉它leader是谁？
247     /\ LET entry == [term  |-> currentTerm[i], 里面有term信息
248                      value |-> v]
249            newLog == Append(log[i], entry)
250        IN  log' = [log EXCEPT ![i] = newLog]
251     /\ UNCHANGED <<messages, serverVars, candidateVars,
252                    leaderVars, commitIndex>>
253
254 \* Leader i advances its commitIndex.
255 \* This is done as a separate step from handling AppendEntries responses,
256 \* in part to minimize atomic regions, and in part so that leaders of
257 \* single-server clusters are able to mark entries committed.
258 AdvanceCommitIndex(i) ==
259     /\ state[i] = Leader
```

log'，只有server i的log变了，其他server的log不变

IN的含义：(let def in expr，只对这个表达式有效)

leader自身，不含其他

```
260     /\ LET \* The set of servers that agree up through index.
261            Agree(index) == {i} \cup {k \in Server :
262                                      matchIndex[i][k] >= index}
263            \* The maximum indexes for which a quorum agrees
264            agreeIndexes == {index \in 1..Len(log[i]) :
265                                Agree(index) \in Quorum}
266            \* New value for commitIndex'[i]
267            newCommitIndex ==
268               IF /\ agreeIndexes /= {}
269                  /\ log[i][Max(agreeIndexes)].term = currentTerm[i]
270               THEN
271                   Max(agreeIndexes)
272               ELSE
273                   commitIndex[i]   条件不符合，则根本没变
274        IN commitIndex' = [commitIndex EXCEPT ![i] = newCommitIndex]
275     /\ UNCHANGED <<messages, serverVars, candidateVars, leaderVars, log>>
276
277 ----
278 \* Message handlers
279 \* i = recipient, j = sender, m = message
280
281 \* Server i receives a RequestVote request from server j with
282 \* m.mterm <= currentTerm[i].
283 HandleRequestVoteRequest(i, j, m) ==                    TLA+ book里面有"Let expression"
284     LET logOk == \/ m.mlastLogTerm > LastTerm(log[i])
285                  \/ /\ m.mlastLogTerm = LastTerm(log[i])
286                     /\ m.mlastLogIndex >= Len(log[i])
287         grant == /\ m.mterm = currentTerm[i]
288                  /\ logOk
289                  /\ votedFor[i] \in {Nil, j}   这个感觉与论文中的反了？==>实际上，UpdateTerm已经修改了server的term，就比mterm大。
290     IN /\ m.mterm <= currentTerm[i]            但是这个感觉比较奇怪，这些/\，是不是一个原子操作？
291        /\ \/ grant  /\ votedFor' = [votedFor EXCEPT ![i] = j]  grant为true，执行后面这个。替代了if..else
292           \/ ~grant /\ UNCHANGED votedFor
293        /\ Reply([mtype        |-> RequestVoteResponse,
294                  mterm        |-> currentTerm[i],
295                  mvoteGranted |-> grant,
296                  \* mlog is used just for the `elections' history variable for
297                  \* the proof. It would not exist in a real implementation.
298                  mlog         |-> log[i],
299                  msource      |-> i,
300                  mdest        |-> j],
301                  m)
302        /\ UNCHANGED <<state, currentTerm, candidateVars, leaderVars, logVars>>
303
304 \* Server i receives a RequestVote response from server j with
305 \* m.mterm = currentTerm[i].
306 HandleRequestVoteResponse(i, j, m) ==
307     \* This tallies votes even when the current state is not Candidate, but
308     \* they won't be looked at, so it doesn't matter.
309     /\ m.mterm = currentTerm[i]
310     /\ votesResponded' = [votesResponded EXCEPT ![i] =
311                               votesResponded[i] \cup {j}]
312     /\ \/ /\ m.mvoteGranted
313           /\ votesGranted' = [votesGranted EXCEPT ![i] =
314                                   votesGranted[i] \cup {j}]
315           /\ voterLog' = [voterLog EXCEPT ![i] =
316                               voterLog[i] @@ (j :> m.mlog)]
317        \/ /\ ~m.mvoteGranted
318           /\ UNCHANGED <<votesGranted, voterLog>>
319     /\ Discard(m)
320     /\ UNCHANGED <<serverVars, votedFor, leaderVars, logVars>>
321
322 \* Server i receives an AppendEntries request from server j with
323 \* m.mterm <= currentTerm[i]. This just handles m.entries of length 0 or 1, but
324 \* implementations could safely accept more by treating them the same as
325 \* multiple independent requests of 1 entry.
326 HandleAppendEntriesRequest(i, j, m) ==
327     LET logOk == \/ m.mprevLogIndex = 0
```

```
                \/ /\ m.mprevLogIndex > 0
                   /\ m.mprevLogIndex <= Len(log[i])
                   /\ m.mprevLogTerm = log[i][m.mprevLogIndex].term
    IN /\ m.mterm <= currentTerm[i]
       /\ \/ /\ \* reject request
             \/ m.mterm < currentTerm[i]
             \/ /\ m.mterm = currentTerm[i]
                /\ state[i] = Follower
                /\ \lnot logOk
          /\ Reply([mtype           |-> AppendEntriesResponse,
                    mterm           |-> currentTerm[i],
                    msuccess        |-> FALSE,
                    mmatchIndex     |-> 0,
                    msource         |-> i,
                    mdest           |-> j],
                    m)
          /\ UNCHANGED <<serverVars, logVars>>
       \/ \* return to follower state
          /\ m.mterm = currentTerm[i]
          /\ state[i] = Candidate
          /\ state' = [state EXCEPT ![i] = Follower]
          /\ UNCHANGED <<currentTerm, votedFor, logVars, messages>>
       \/ \* accept request
          /\ m.mterm = currentTerm[i]
          /\ state[i] = Follower
          /\ logOk
          /\ LET index == m.mprevLogIndex + 1
             IN \/ \* already done with request
                   /\ \/ m.mentries = << >>
                      \/ /\ Len(log[i]) >= index
                         /\ log[i][index].term = m.mentries[1].term
                      \* This could make our commitIndex decrease (for
                      \* example if we process an old, duplicated request),
                      \* but that doesn't really affect anything.
                   /\ commitIndex' = [commitIndex EXCEPT ![i] =    根据什么决定修改commitIndex? 而且还是捎带？
                                        m.mcommitIndex]
                   /\ Reply([mtype           |-> AppendEntriesResponse,
                             mterm           |-> currentTerm[i],
                             msuccess        |-> TRUE,
                             mmatchIndex     |-> m.mprevLogIndex +
                                                Len(m.mentries),
                             msource         |-> i,
                             mdest           |-> j],
                             m)
                   /\ UNCHANGED <<serverVars, logVars>>
                \/ \* conflict: remove 1 entry
                   /\ m.mentries /= << >>
                   /\ Len(log[i]) >= index
                   /\ log[i][index].term /= m.mentries[1].term
                   /\ LET new == [index2 \in 1..(Len(log[i]) - 1) |->
                                    log[i][index2]]
                      IN log' = [log EXCEPT ![i] = new]
                   /\ UNCHANGED <<serverVars, commitIndex, messages>>
                \/ \* no conflict: append entry
                   /\ m.mentries /= << >>
                   /\ Len(log[i]) = m.mprevLogIndex
                   /\ log' = [log EXCEPT ![i] =
                                Append(log[i], m.mentries[1])]
                   /\ UNCHANGED <<serverVars, commitIndex, messages>>
       /\ UNCHANGED <<candidateVars, leaderVars>>

\* Server i receives an AppendEntries response from server j with
\* m.mterm = currentTerm[i].
HandleAppendEntriesResponse(i, j, m) ==
    /\ m.mterm = currentTerm[i]
    /\ \/ /\ m.msuccess \* successful
          /\ nextIndex'  = [nextIndex  EXCEPT ![i][j] = m.mmatchIndex + 1]
          /\ matchIndex' = [matchIndex EXCEPT ![i][j] = m.mmatchIndex]
```

```
396         \/ /\ \lnot m.msuccess \* not successful
397            /\ nextIndex' = [nextIndex EXCEPT ![i][j] =
398                                Max({nextIndex[i][j] - 1, 1})]
399            /\ UNCHANGED <<matchIndex>>
400      /\ Discard(m)
401      /\ UNCHANGED <<serverVars, candidateVars, logVars, elections>>
402
403  \* Any RPC with a newer term causes the recipient to advance its term first.
404  UpdateTerm(i, j, m) ==
405      /\ m.mterm > currentTerm[i]
406      /\ currentTerm'    = [currentTerm EXCEPT ![i] = m.mterm]
407      /\ state'          = [state       EXCEPT ![i] = Follower]
408      /\ votedFor'       = [votedFor    EXCEPT ![i] = Nil]
409         \* messages is unchanged so m can be processed further.
410      /\ UNCHANGED <<messages, candidateVars, leaderVars, logVars>>
411
412  \* Responses with stale terms are ignored.
413  DropStaleResponse(i, j, m) ==
414      /\ m.mterm < currentTerm[i]
415      /\ Discard(m)
416      /\ UNCHANGED <<serverVars, candidateVars, leaderVars, logVars>>
417
418  \* Receive a message.
419  Receive(m) ==
420      LET i == m.mdest
421          j == m.msource
422      IN \* Any RPC with a newer term causes the recipient to advance
423         \* its term first. Responses with stale terms are ignored.
424         \/ UpdateTerm(i, j, m)          执行的原子单位是什么？ UpdateTerm会不会执行完了，后面的操作都不执行，中间插入了别的操作？
425         \/ /\ m.mtype = RequestVoteRequest
426            /\ HandleRequestVoteRequest(i, j, m)
427         \/ /\ m.mtype = RequestVoteResponse
428            /\ \/ DropStaleResponse(i, j, m)
429               \/ HandleRequestVoteResponse(i, j, m)
430         \/ /\ m.mtype = AppendEntriesRequest
431            /\ HandleAppendEntriesRequest(i, j, m)
432         \/ /\ m.mtype = AppendEntriesResponse
433            /\ \/ DropStaleResponse(i, j, m)
434               \/ HandleAppendEntriesResponse(i, j, m)
435
436  \* End of message handlers.
437  ----
438  \* Network state transitions
439
440  \* The network duplicates a message
441  DuplicateMessage(m) ==
442      /\ Send(m)
443      /\ UNCHANGED <<serverVars, candidateVars, leaderVars, logVars>>
444
445  \* The network drops a message
446  DropMessage(m) ==
447      /\ Discard(m)
448      /\ UNCHANGED <<serverVars, candidateVars, leaderVars, logVars>>
449
450  ----
451  \* Defines how the variables may transition.
452  Next == /\ \/ \E i \in Server : Restart(i)
453             \/ \E i \in Server : Timeout(i)
454             \/ \E i,j \in Server : RequestVote(i, j)
455             \/ \E i \in Server : BecomeLeader(i)
456             \/ \E i \in Server, v \in Value : ClientRequest(i, v)
457             \/ \E i \in Server : AdvanceCommitIndex(i)
458             \/ \E i,j \in Server : AppendEntries(i, j)
459             \/ \E m \in DOMAIN messages : Receive(m)
460             \/ \E m \in DOMAIN messages : DuplicateMessage(m)
461             \/ \E m \in DOMAIN messages : DropMessage(m)
462             \* History variable that tracks every log ever:
463          /\ allLogs' = allLogs \cup {log[i] : i \in Server}
```

```
464
465 \* The specification must start with the initial state and transition according
466 \* to Next.                          //TODO: 这里没有没有让系统保证什么条件一定成立，liveness或者theorem,这样运行的意义？
467 Spec == Init /\ [][Next]_vars        到底在保证什么？例如paxos例子的voting module中，有很多。
468                                       起码需要保证，各个server，同位置的log，是相同的。
469 ============================================================================
470
471 \* Changelog:
472 \*
473 \* 2014-12-02:
474 \* - Fix AppendEntries to only send one entry at a time, as originally
475 \*   intended. Since SubSeq is inclusive, the upper bound of the range should
476 \*   have been nextIndex, not nextIndex + 1. Thanks to Igor Kovalenko for
477 \*   reporting the issue.
478 \* - Change matchIndex' to matchIndex (without the apostrophe) in
479 \*   AdvanceCommitIndex. This apostrophe was not intentional and perhaps
480 \*   confusing, though it makes no practical difference (matchIndex' equals
481 \*   matchIndex). Thanks to Hugues Evrard for reporting the issue.
482 \*
483 \* 2014-07-06:
484 \* - Version from PhD dissertation
```