Distributed Algorithms
Lecture 4

Lecture by Dr. Gadi Taubenfeld
Typeset by Steven Karas

2017-04-26
Last edited 15:25:07 2017-05-03

Disclaimer These lecture notes are based on the lecture for the course Distributed Algorithms, taught by Dr. Gadi Taubenfeld at IDC Herzliyah in the spring semester of 2017. Sections may be based on the lecture slides and accompanying book written by Dr. Gadi Taubenfeld.

1 Leader Election

Leaders are special processes that can perform globally correct actions such as breaking deadlocks, allocating shared resources, collect global data about the network.

Potential Assumptions

- Asynchronous (no assumptions about time/speed) or synchronous?
- Unique identifiers?
- Process topology
- Global "direction"
- Perfect knowledge of number of processes
- Message delivery order
- Are failures possible?
- Starting state

1.1 A trivial algorithm for ring leader election

Model

- Asynchronous network
- Unique identifiers
- Unidirectional ring topology
- Number of processes is unknown
- FIFO message delivery
- No failures (processes, links, messages)
- Starting condition: spontaneous, or upon message delivery.

A trivial algorithm This algorithm was presented by G. LeLann in 1977, and provides leader election in $O(n^2)$ messages.

For processes $i = 1...n$

```
self.others = []
Send your id to your neighbor
Each time a message is received:
  if self.id != message.id:
    self.others.append(message.id)
    forward message
  else:
    if max(self.others) == self.id:
      self.is_leader = True
    else:
      self.is_leader = False
```

A correct algorithm for a unidirectional ring is also correct for a bidirectional ring with a sense of direction. A correct algorithm for a unidirectional ring is not necessarily correct for a bidirectional ring without a sense of direction. A correct algorithm for an unknown number of processes is also correct when the number of processes is known.

In this case, this algorithm is not correct for a bidirectional ring without a sense of direction. For example, take two processes. The theoretical leader's first message is delayed, and the other one pushes across different links its message.

There is no deterministic algorithm for electing a leader when there are no unique identifiers.

1.2 An improved algorithm

Basically, the same as above, but drop messages if message.id is less than self.id. This doesn't affect the worst time complexity, but reduces best case to $O(n)$ and average case to $O(n \log n)$. Even better, it makes the algorithm valid for a bidirectional ring without a sense of direction.

1.3 An even better algorithm

This algorithm was presented by Gary L. Peterson in 1982. Provides leader election in $O(n \log n)$ messages.

Model

- Asynchronous network
- Unique identifiers
- Bidirectional ring without a sense of direction
- Number of processes is unknown
- FIFO message delivery
- No failures (processes, links, messages)
- Starting condition: spontaneous, or upon message delivery.

Bidirectional variant Basically, each process sends its id to its neighbors, and gathers the ids of its neighbors. Each process then makes a decision of whether or not it will continue to the next round of selection, and if not, it turns itself into a dumb relay (just passes messages back and forth).

Unidirectional algorithm To convert from a bidirectional system to a unidirectional system, we simply allow a process to "adopt" the id of its neighbor, and repeat the process as above, but shifting the process identities by one each time (so each process makes the decision as if it were its prior neighbor consider itself and the two previous processes).

As a special case, this does not work for a system with only two processes (or a last round with two processes). As such, we replace an id lower than itself with its own id to send on to the next node.

For processes $i = 1...n$

```
tid = initial value
do:
  send(tid)
  receive(ntid)
  announce elected if ntid == initial value
  send(max(tid, ntid))
  receive(nntid)
  announce elected if nntid == initial value
  if ntid ≥ max(tid, nntid):
    tid = ntid
  else:
    goto relay

relay:
do:
  receive(ntid)
  announce elected if ntid == initial value
  send(ntid)
```

Analysis Each round uses $O(n)$ messages, and reduces the potential leader pool by a factor of 3. Therefore, the total messages used is $O(n \log n)$. A full analysis shows $2n \log n + n$ messages.

Improved Bidirectional Variant By comparing the first neighbor, we short circuit becoming a relay by performing half-turns, alternating left and right comparison. This makes the algorithm slightly simpler.

Improved Unidirectional Variant Simulate the previous by using the same strategy of adopting labels. This improves the number of messages needed to $1.44n \log n$.

2 Next time

We will continue on the election problem, including the lower bound proof of $O(n \log n)$