DS Lecture 7.2
Leader Election

October 24, 2022

Michele Albano
mialb@cs.aau.dk

DEIS
Aalborg University
Denmark

AALBORG UNIVERSITY
DENMARK

Based on slides by Peter G. Jensen, AAU.

What is it?

What is leader election?

Leader election algorithms ensure that one, and **only one** process is elected as the **leader** in the event of **lack of a leader**

What is it?

Examples

- Algorithms with coordinator
  - Mutex?
- Distributed replication
- DNS-servers in case of network partition
- Failover mechanism for crashes
- Parliaments?

What is it?

Big Question:

We detected a crash of the leader, now what?

Agenda

Requirements

Assumptions

Chang-Roberts

Bully Algorithm

Requirements

Leader Election

Let $P = \{p_1, \ldots, p_n\}$ be a set of processes and let $L(p_i) \in P \cup \{\perp\}$ be the leader as seen from a process p_i

1. Safety
   - either $L(p_i) = \perp$ or $L(p_i) = p_j$
     - p_j is the non-crashed process with largest identifier
2. Liveness
   - All process participate, and eventually
     - p_i crashes, or
     - $L(p_i) \neq \perp$

Note

Processes can crash during the election

Assumptions

- Processes stay dead
- Crashes are reliably detected
- Identifiers are unique

Chang Roberts

Idea

- Pass token of largest ID in a ring
- Basic algorithm in election
  - Forward ID to “next” if higher than own,
  - Forward own ID to “next” otherwise.
  - Only one active election
- Two message types:
  - *election* to vote. If p_i receives it and is not "participating", it starts participating
  - *elected* to declare who won. It p_i is participating and receives its own ID, it means it won, becomes "non-participating" and send the *elected* message

Chang-Roberts - Code

```
    def run(self):
        while True:
            nxt = (self.index() + 1) % self.
       number_of_devices()
            if not self._participated:
                self.medium().send(
                    Vote(self.index(), nxt, self.index(),
                           False))
                self._participated = True
            ingoing = self.medium().receive()
            if ingoing is not None:
                if ingoing.vote() == self.index():
                    if not ingoing.decided():
                        self.medium().send(
                            Vote(self.index(), nxt, self.
                                  index(), True))
                    else:
                        self._leader = self.index()
                        return  # this device is the new
                              leader
                elif ingoing.vote() < self.index():
                    continue
                elif ingoing.vote() > self.index():
                    # forward the message
                    self.medium().send(
                        Vote(self.index(), nxt, ingoing.vote(),
                               ingoing.decided()))
                    if ingoing.decided():
                        self._leader = ingoing.vote()
                        return
            self.medium().wait_for_next_round()
```

Properties

Chang Roberts

- Safe
- Live
- $3N - 1$ messages per election

Crashes?

Can be overcome if reliably detected!

Bully Algorithm

Idea

- Bully election requests into silence
- Priority by ID
- Basic algorithm in election
  - Send “shut up” to lower IDs
  - Higher IDs will "answer"
  - Finally, highest alive ID broadcasts itself

NOTE!

Depends on Synchronous Behavior (timeout to detect crashes)

Bully Algorithm - Code

```
class Bully(Device):
    def __init__(...):
        super().__init__(index, number_of_devices, medium)
        self._leader = None
        self._shut_up = False
        self._election = False

    def run(self):
        first_round = True
        while True:
            got_input = False
            if not self._shut_up and not self._election:
                    self.start_election()
            new_election = False
            while True:
                ingoing = self.medium().receive()
                if ingoing is not None:
                    got_input = True
                    if ingoing.vote() < self.index():
                        self.medium().send(
                    Vote(self.index(), ingoing.source, self.index(), self.largest()))
                        new_election = True
                    else:
                        self._shut_up = True
                        if ingoing.decided():
                            self._leader = ingoing.vote()
                            return
                else: break
```

```
            if not self._shut_up and not self._election and new_election:
                self.start_selection()
            if not got_input and not first_round:
                if self._election:
                    if self._shut_up:
                        self._shut_up = False
                        self.start_election()
                    else:
                        for id in self.medium().ids():
                            if id != self.index():
                                self.medium().send(Vote(self.index(), id, self.index(), True))
                        self._leader = self.index()
                        return
            self.medium().wait_for_next_round()
            first_round = False

    def largest(self):
        return self.index() == max(self.medium().ids())

    def start_election(self):
        if not self._election:
            self._election = True
            for id in self.medium().ids():
                if id > self.index():
                    self.medium().send(Vote(self.index(), id, self.index(), self.largest()))
```

Properties
Bully

- Safe & Live, assuming
  - Unique IDs
  - Failure detection is reliable
- Best-case: $N - 2$ messages pr election
- Worst-case: $O(N^2)$ messages (the process with the lowest ID starts the election)
- Election-time: 2 rounds (assuming HW multicast)

Properties

Bully

Beware

Safety is broken if

- too tight deadline,
- process IDs reappear, or
- system is not synchronous.

Requires

- **requires synchronous system** , and
- **ordering of messages**