Lecture 17: Leader Election
Why Election?

• Example 1: Your Bank account details are replicated at a few servers, but one of these servers is responsible for receiving all reads and writes, i.e., it is the leader among the replicas
  • What if there are two leaders per customer?
  • What if servers disagree about who the leader is?
  • What if the leader crashes?

   Each of the above scenarios leads to Inconsistency
More motivating examples

• Example 2: (A few lectures ago) In the sequencer-based algorithm for total ordering of multicasts, the “sequencer” = leader

• Example 3: Group of NTP servers: who is the root server?

• Other systems that need leader election: Apache Zookeeper, Google’s Chubby

• Leader is useful for coordination among distributed servers
Leader Election Problem

• In a group of processes, elect a Leader to undertake special tasks
  • And let everyone know in the group about this Leader

• What happens when a leader fails (crashes)
  • Some process detects this (using a Failure Detector!)
  • Then what?

• Focus of this lecture: Election algorithm. Its goal:
  1. Elect one leader only among the non-faulty processes
  2. All non-faulty processes agree on who is the leader
System Model

- $N$ processes.
- Each process has a unique id.
- Messages are eventually delivered.
- Failures may occur during the election protocol.
Calling for an Election

• Any process can call for an election.
• A process can call for at most one election at a time.
• Multiple processes are allowed to call an election simultaneously.
  • All of them together must yield only a single leader
• The result of an election should not depend on which process calls for it.
Election Problem, Formally

- A run of the election algorithm must always guarantee at the end:
  - **Safety:** For all non-faulty processes $p$: ($p'$'s elected = (q: a particular non-faulty process with the best attribute value) or Null)
  - **Liveness:** For all election runs: (election run terminates) & for all non-faulty processes $p$: $p'$'s elected is not Null

- At the end of the election protocol, the non-faulty process with the **best (highest)** election attribute value is elected.
  - Common attribute: leader has highest id
  - Other attribute examples: leader has highest IP address, or fastest cpu, or most disk space, or most number of files, etc.
First Classical Algorithm: Ring Election

- \( N \) processes are organized in a logical ring
  - Similar to ring in Chord p2p system
  - \( i \)-th process \( p_i \) has a communication channel to \( p_{(i+1) \mod N} \)
  - All messages are sent clockwise around the ring.
The Ring
The Ring Election Protocol

• Any process $p_i$ that discovers the old coordinator has failed initiates an “Election” message that contains $p_i$’s own id:attr. This is the initiator of the election.

• When a process $p_i$ receives an “Election” message, it compares the attr in the message with its own attr.
  • If the arrived attr is greater, $p_i$ forwards the message.
  • If the arrived attr is smaller and $p_i$ has not forwarded an election message earlier, it overwrites the message with its own id:attr, and forwards it.
  • If the arrived id:attr matches that of $p_i$, then $p_i$’s attr must be the greatest (why?), and it becomes the new coordinator. This process then sends an “Elected” message to its neighbor with its id, announcing the election result.
The Ring Election Protocol (2)

- When a process $p_i$ receives an “Elected” message, it
  - sets its variable $elected_i \leftarrow$ id of the message.
  - forwards the message unless it is the new coordinator.
Ring Election: Example

Goal: Elect highest id process as leader

Initiates the election
Election: 3
Goal: Elect highest id process as leader

Initiates the election

Election: 32

N80
N32
N5
N6
N3
N12
Goal: Elect highest id process as leader

Election: 32

Initiates the election
Initiates the election

Goal: Elect highest id process as leader
Goal: Elect highest id process as leader
Initiates the election

Election: 80

N80
N32
N3
N12
N6
N5

Goal: Elect highest id process as leader
Goal: Elect highest id process as leader
Goal: Elect highest id process as leader

Initiates the election

Elected: 80

N12 → N3 → N6 → N32 → N80 → N5
Goal: Elect highest id process as leader
Goal: Elect highest id process as leader

Initiates the election

elected = 80

N12

N6

N80

N3

N32

N5

Elected: 80

elected = 80
Initiates the election

Goal: Elect highest id process as leader
Analysis

• Let’s assume no failures occur during the election protocol itself, and there are $N$ processes.

• How many messages?

• Worst case occurs when the initiator is the ring successor of the would-be leader.
Worst-case

Goal: Elect highest id process as leader
Worst-case Analysis

- \((N-1)\) messages for Election message to get from Initiator (N6) to would-be coordinator (N80)
- \(N\) messages for Election message to circulate around ring without message being changed
- \(N\) messages for Elected message to circulate around the ring
- Message complexity: \((3N-1)\) messages
- Completion time: \((3N-1)\) message transmission times
- Thus, if there are no failures, election terminates (liveness) and everyone knows about highest-attribute process as leader (safety)
Best Case?

- Initiator is the would-be leader, i.e., N80 is the initiator
- Message complexity: $2N$ messages
- Completion time: $2N$ message transmission times
Multiple Initiators?

- Include initiator’s id with all messages
- Each process remembers in cache the initiator of each Election/Elected message it receives
- (All the time) Each process suppresses Election/Elected messages of any lower-id initiators
- Updates cache if receives higher-id initiator’s Election/Elected message
- Result is that only the highest-id initiator’s election run completes
- What about failures?
Effect of Failures

Initiates the election

Elected: 80

elected = 80

Election: 80 will circulate around the ring forever
Liveness violated
Fixing for failures

- One option: have predecessor (or successor) of would-be leader N80 detect failure and start a new election run
  - May re-initiate election if
    - Receives an Election message but times out waiting for an Elected message
    - Or after receiving the Elected:80 message
  - But what if predecessor also fails?
  - And its predecessor also fails? (and so on)
Fixing for failures (2)

• Second option: use the failure detector
• Any process, after receiving Election:80 message, can detect failure of N80 via its own local failure detector
  • If so, start a new run of leader election
• But failure detectors may not be both complete and accurate
  • Incompleteness in FD => N80’s failure might be missed => Violation of Safety
  • Inaccuracy in FD => N80 mistakenly detected as failed
    • => new election runs initiated forever
    • => Violation of Liveness
Why is Election so Hard?

- Because it is related to the consensus problem!
- If we could solve election, then we could solve consensus!
  - Elect a process, use its id’s last bit as the consensus decision
- But since consensus is impossible in asynchronous systems, so is election!
- (later in lecture) Consensus-like protocols used in industry for leader election
Another Classical Algorithm: Bully Algorithm

- All processes know other process’ ids
- When a process finds the coordinator has failed (via the failure detector):
  - if it knows its id is the highest
    - it elects itself as coordinator, then sends a *Coordinator* message to all processes with lower identifiers. Election is completed.
  - else
    - it initiates an election by sending an *Election* message
    - (contd.)
Bully Algorithm (2)

- else it initiates an election by sending an *Election* message
  - Sends it to only processes that have a higher id than itself.
  - if receives no answer within timeout, calls itself leader and sends *Coordinator* message to all lower id processes. Election completed.
  - if an answer received however, then there is some non-faulty higher process => so, wait for coordinator message. If none received after another timeout, start a new election run.

- A process that receives an *Election* message replies with *OK* message, and starts its own leader election protocol (unless it has already done so)
Bully Algorithm: Example

Detects failure of N80
Detects failure of N80
Coordinator: N32

Times out waiting for N80's response

Election is completed
Failures during Election Run

- N80
- N32
- N12
- N3
- N5
- N6

Waiting… Waiting…
Waiting…

Times out, starts new election run

Election

OK
Times out, starts another new election run
Failures and Timeouts

• If failures stop, eventually will elect a leader
• How do you set the timeouts?
• Based on Worst-case time to complete election
  • 5 message transmission times if there are no failures during the run:
    1. Election from lowest id server in group
    2. Answer to lowest id server from 2\textsuperscript{nd} highest id process
    3. Election from 2nd highest id server to highest id
    4. Timeout for answers @ 2nd highest id server
    5. Coordinator from 2\textsuperscript{nd} highest id server
Analysis

- **Worst-case** completion time: 5 message transmission times
  - When the process with the lowest id in the system detects the failure.
    - \((N-1)\) processes altogether begin elections, each sending messages to processes with higher ids.
    - \(i\)-th highest id process sends \((i-1)\) election messages
  - Number of Election messages
    \[ = N-1 + N-2 + \ldots + 1 = (N-1)\times N/2 = O(N^2) \]

- **Best-case**
  - Second-highest id detects leader failure
  - Sends \((N-2)\) Coordinator messages
  - Completion time: 1 message transmission time
Impossibility?

- Since timeouts built into protocol, in asynchronous system model:
  - Protocol may never terminate => Liveness not guaranteed
- But satisfies liveness in synchronous system model where
  - Worst-case one-way latency can be calculated = worst-case processing time + worst-case message latency
Can use Consensus to solve Election

• One approach
  • Each process proposes a value
  • Everyone in group reaches consensus on some process $P_i$’s value
  • That lucky $P_i$ is the new leader!
Election in Industry

- Several systems in industry use Paxos-like approaches for election
  - Paxos is a consensus protocol (safe, but eventually live): earlier in this course
- Google’s Chubby system
- Apache Zookeeper
Election in Google Chubby

• A system for locking
• Essential part of Google’s stack
  • Many of Google’s internal systems rely on Chubby
  • BigTable, Megastore, etc.
• Group of replicas
  • Need to have a master server elected at all times

Reference: http://research.google.com/archive/chubby.html
• Group of replicas
  • Need to have a master (i.e., leader)

• Election protocol
  • Potential leader tries to get votes from other servers
  • Each server votes for at most one leader
  • Server with *majority* of votes becomes new leader, informs everyone
Why safe?

- Essentially, each potential leader tries to reach a quorum
- Since any two quorums intersect, and each server votes at most once, cannot have two leaders elected simultaneously

Why live?

- Only eventually live! Failures may keep happening so that no leader is ever elected
- In practice: elections take a few seconds. Worst-case noticed by Google: 30 s
After election finishes, other servers promise not to run election again for “a while”

- “While” = time duration called “Master lease”
- Set to a few seconds

Master lease can be renewed by the master as long as it continues to win a majority each time

Lease technique ensures automatic re-election on master failure
Election in Zookeeper

- Centralized service for maintaining configuration information
- Uses a variant of Paxos called Zab (Zookeeper Atomic Broadcast)
- Needs to keep a leader elected at all times

http://zookeeper.apache.org/
Election: Summary

• Leader election an important component of many cloud computing systems
• Classical leader election protocols
  • Ring-based
  • Bully
• But failure-prone
  • Paxos-like protocols used by Google Chubby, Apache Zookeeper