CS 455: INTRODUCTION TO DISTRIBUTED SYSTEMS [ELECTION ALGORITHMS]

Shrideep Pallickara
Computer Science
Colorado State University

APRIL 13, 2017

Frequently asked questions from the previous class survey

- What if new processes enter the system?
- Why would more than one process call the election algorithm?
- Maekawa’s voting set

Topics covered in this lecture

- Election algorithms
  - Ring based algorithm
  - Failure detectors

ELECTION ALGORITHMS

APRIL 13, 2017

Election algorithms

- Algorithm for choosing a unique process to play a particular role
- When an elected process wants to retire, another election is needed

Calling an election

- When a process calls an election it initiates a particular run of the election algorithm
- A given process does not call more than one election at a time
  - With N processes, there could be N concurrent elections
- At any point a process \( p_i \) is either:
  - A participant: Engaged in the election algorithm
  - Non-participant: Not engaged in the election algorithm
The choice of the elected process must be unique
- Even in cases where several processes call the election simultaneously
- E.g., 2 processes see a coordinator has failed and they both call elections

The elected process is the one with the largest identifier
- The identifier is any value with the provision that the identifiers are unique and totally ordered
- E.g., electing process with the lowest computational load
  - Use $<\text{load, i}>$ as the identifier
  - Process $i$ is used to order identifiers with same load

Managing the identity of the elected process
- Each process $p_i$ ($i=1, 2, ..., N$) has a variable $\text{elected}_i$
  - Contains identifier of the elected process
- When a process first becomes a participant in an election
  - Set this variable to $\perp$ indicating that it is undefined.

Requirements for the election algorithm
- **E1** (safety)
  - Participant process has $\text{elected}_i = \perp$ or $\text{elected}_i = \text{P}$
    - $\text{P}$ is a non-crashed process at the end of run with the largest identifier
- **E2** (liveness)
  - All processes $p_i$ participate and eventually either set $\text{elected}_i \neq \perp$ or crash

Measuring performance of election algorithms
- Network bandwidth utilization
  - How many messages are sent?
- Turnaround time for the algorithm
  - Number of message transmissions between the initiation and termination of a run
Ring-based elections

- Each process $p_i$ has a communication channel to the next process ($p_{(i+1) \mod N}$) in the next ring.
- All messages are sent **clockwise** around the ring.

Conducting elections

- Any process can begin an election.
- Process marks itself as a participant and:
  1. Places its identifier in the election message
  2. Sends it to its clockwise neighbor.
- **On forwarding** an election message:
  - Process marks itself as a participant.

When a process receives an election message it compares identifier with its own

- If the arrived identifier is **greater**:
  - Forward message to its neighbor.
- If the arrived identifier is **smaller**:
  - If the process is not a participant:
    - Substitute with own identifier and forward the message.
  - If the process is already a participant:
    - Do not forward the message.

Selecting the coordinator

- If the received identifier is that of the receiver itself?
  - That process’ identifier must be the greatest.
  - Becomes the **coordinator**.
- Coordinator marks itself as a non-participant.
- Sends an **elected** message to its neighbor.

When a process $p_i$ receives an elected message

- Marks itself as a **non-participant**.
- Sets **elected** to the identifier in the message.
- Unless it is the coordinator, it forwards message to its neighbor.

Ring-based elections

- Participant processes are in green.
- Election was started by process 17.
Satisfying the requirements: Safety

- All identifiers are compared
  - A process must receive its own identifier back before sending an elected message

- For any two processes, the one with the larger identifier will not pass the other’s identifier
  - Impossible that 2 processes will receive their own identifier back

Satisfying the requirements: Liveness

- Message traversals are guaranteed around the ring
  - The basic algorithm assumes no failures

- Duplicate messages arising when 2 processes start election at the same time?
  - Participant and non-participant states extinguish this ASAP and ...
  - Always before winning election result has been announced

Performance analysis

- If only a single process starts an election
  - Worst case is when anti-clockwise neighbor has the highest identifier
  - Will not announce election till its identifier completes another circuit ... N messages
  - Elected message is sent N times
  - Total of 3N-1 messages
  - Turnaround time also is 3N-1 in the worst case
  - Messages are sent sequentially

Failure detection

- In order to properly mask failures ...
  - We generally need to detect them well

- Failure detection is one of the cornerstones of fault tolerance in distributed systems

Failure detectors

- What it all boils down to ...
  - For a group of processes, non-faulty members must be able to decide:
    - Who is still a member and who is not
  - We need to be able to detect when a member has failed
The two mechanisms for detecting failures

- Processes actively send “are you alive?” messages to each other
  - For which they obviously expect an answer
- Passively wait until heartbeats (“I am alive!”) come in from different processes.
  - In practice, often active pinging occurs as well

Huge body of theoretical work on failure detectors

- Timeout mechanisms are used to check whether a process has failed
  - In real settings:
    - Due to unreliable networks, just because a process does not respond to a ping that does not mean it failed
    - So … false positives can occur quite easily
      - A healthy process may be removed from the membership

Disambiguating network failures from node failures

- Multiple nodes participate in failure detection
- When a node notices a timeout on a ping message
  - The node contacts other nodes to see if they can reach the presumed failing node

Communication styles

- Asynchronous communications
  - No timing assumptions
- Synchronous communications have bounds on
  - Maximum message transmission delay
  - Time to execute each step of a process
  - Clock drift rates

Failure assumptions & failure detectors

- Processes use reliable channels to communicate
  - Underlying protocol handles corruptions and retransmissions
- Failures of processes are independent

Failure assumptions & failure detectors

- Network partitions are possible between the set of communicating processes
- Over the internet with complex topologies and independent routing choices
  - Connectivity may be asymmetric
    - Communication from \( p \) to \( q \) is possible, but not vice versa
  - Connectivity may be intransitive
    - Communication is possible from \( p \) to \( q \) to \( r \), but \( p \) cannot communicate directly with \( r \)
Failure detectors

- **Service** that processes queries about whether a process failed
- Often implemented as an object **local to each process**
- Runs a failure detection algorithm in conjunction with counterparts at other processes
- Not necessarily accurate
- Only as good as the information available at that process

Unreliable failure detectors

- Produces one of two values when given the identity of a process
  - Suspected or Unsuspected
  - These values are just **hints** and may not accurately reflect if a process has failed
- **Suspected**
  - Detector recently received evidence suggesting process has not failed
- **Unsuspected**
  - Detector has some indication that process probably failed
  - Message not received for more than nominal silence interval

Unreliable failure detectors

- Suspicions may be **misplaced**
  - Process may be functioning, but on the other side of a network partition
  - Process runs slower than expected

Reliable failure detectors

- Answers liveness queries with
  - Unsuspected
  - Failed

Implementing an unreliable failure detector

- Each process $p$, sends a **heartbeat** every $T$ seconds
- Failure detector uses estimate of **maximum message transmission delay** of $D$ seconds
- If failure detector at $q$ does not receive heartbeat from $p$ within $T + D$ seconds of the last one?
  - Detector reports to $q$ that $p$ is Suspected
- If heartbeat is received within $T + D$ then detector at $q$ deems $p$ to be OK.
Choosing values for $T$ and $D$

- If we choose small values for $T$ and $D$
  - Failure detector is likely to suspect non-crashed processes many times
  - Bandwidth will be consumed by heartbeat messages
- If we choose a large $D$?
  - Crashed processes will often be reported as Unsuspected

Practical solution to the problem

- Use timeout values that reflect observed network delay conditions
- If failure detector at $q$ receives heartbeats from $p$ every 20 seconds instead of 10 seconds?
  - Reset timeout for $p$ to 20 seconds
- Failure detector would still be unreliable
  - But probability of accuracy increases

Building reliable failure detectors

- Possible only in synchronous systems
- $D$ in this case is not an estimate, but an absolute bound
  - Absence of heartbeat from $p$ within $T+D$ seconds entitles detector at $q$ to conclude that $p$ has failed

Contrasting reliable and unreliable failure detectors

- Unreliable failure detectors can be
  - Inaccurate
    - Suspects process that has not failed
  - Incomplete
    - May not suspect a process that has failed
- Reliable failure detectors require a system that is synchronous
  - Few practical systems are

So why did we look at failure detectors?

- They help us think about failures in distributed systems
- Any practical system designed to cope with failures, must detect them
  - However imperfectly!
- Unreliable failure detectors with well-defined properties help us to provide practical solutions for coordinating processes

The Bully Algorithm

- April 13, 2017
  - CS455: Introduction to Distributed Systems [Spring 2017]
  - Dept. Of Computer Science, Colorado State University
  - Instructor: SHRIDEEP PALLICKARA
  - L24.37

...
Bully algorithm (Garcia-Molina):
Key features
- Allows processes to crash during an election
- Assumptions:
  - Message delivery between processes is reliable
  - Synchronous system
  - Uses timeouts to detect a failure
  - Each process knows processes that have higher identifiers
  - Can communicate with them

Message types
- Election
  - Sent to announce an election
- Answer
  - Sent in response to an election message
- Coordinator
  - Sent to announce the identity of the elected process

Initiating elections
- A process begins this when it notices that the coordinator has failed
- Several processes may discover this concurrently

Reliable failure detectors are possible because the system is synchronous
- \( T_{trans} \): Maximum transmission delay
- \( T_{process} \): Maximum delay for processing a message
- Upper bound on elapsed time between sending a message to a process & receiving a response
  - \( T = 2T_{trans} + T_{process} \)
  - If no response arrives within \( T \), local failure detector tags intended recipient as having failed

In the case of a failure
- Process that knows it has the highest identifier can elect itself as the coordinator
- Simply send a coordinator message to processes with lower identifiers

When a process with a lower identifier detects coordinator failure it initiates an election
- Send an election message to processes with higher identifiers
- Await answer messages in response
- If no response within time \( T \), process considers itself the coordinator
- If an answer does arrive, wait for additional time \( T' \) for coordinator message to arrive
- If this does not arrive … start another election
How a process responds to messages that it receives

- If a process $p_i$ receives a coordinator message, it sets its variable $elected_i$ to the coordinator ID.
- If a process receives an election message:
  1. Sends back an answer message and ...
  2. Begins another election
     - Unless it has started one already

But why is this called the bully algorithm?

- When a process is started to replace a crashed process it starts an election.
- If this new process has the highest identifier:
  - It decides that it is the new coordinator and announces this.
- The new process becomes the coordinator even though the current coordinator is functioning.

Election of a coordinator after the failure of $p_4$

1. $p_1$ receives an election message and sends an answer message.
2. $p_2$ receives an election message and sends an answer message.
3. $p_3$ receives an election message and sends an answer message.
4. $p_4$ sends an election message.
5. $p_3$ sends an election message.
6. $p_2$ sends an election message.
7. $p_1$ sends an election message.
8. $p_1$ becomes the coordinator.

Election of a coordinator after the failure of $p_4$ and then $p_3$

1. $p_1$ receives an election message and sends an answer message.
2. $p_2$ receives an election message and sends an answer message.
3. $p_3$ sends an election message.
4. $p_4$ sends an election message.
5. $p_3$ sends an election message.
6. $p_2$ sends an election message.
7. $p_1$ sends an election message.
8. $p_1$ becomes the coordinator.

The contents of this slide set are based on the following references: