CS 455: Introduction To Distributed Systems [Election Algorithms]

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- In the ring-based election, would there be a case where two processes have the same identifier?
  - No. You don't want that. In the case of load-balancing etc. you would come up with a deterministic ordering of ids
- Why do Hadoop/YARN send so many pings?
- Can you choose 50% availability and 50% consistency?
  - No

Topics covered in this lecture

- Election Algorithms
  - Bully algorithm [Garcia-Molina]
  - Elections in wireless environments [Vasudevan et al]
- Architectural Styles

The Bully Algorithm

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
  - $\mathbf{T} = 2T_{trans} + T_{process}$
  - If no response arrives within $\mathbf{T}$, 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

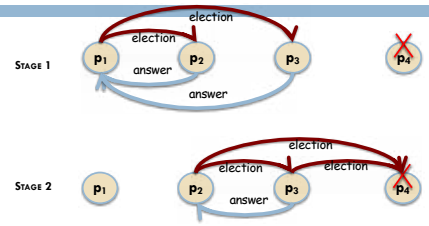
How a process responds to messages that it receives

- 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

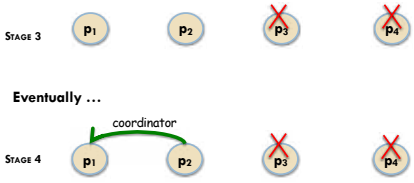
But why is this called the bully algorithm?

- 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 p4



Election of a coordinator after the failure of p4 and then p3



Satisfying properties E1 and E2

- **E1** (safety)
  - Impossible for two processes to decide that they are the coordinator
    - Process with the lower identifier will discover that the other exists and defer to it
- **E2** (liveness)
  - Satisfied because of the assumption of reliable delivery
    - Processes either participate or crash

Safety … not so soon [1/2]

- Not guaranteed to meet safety condition if …
  - Crashed processes are replaced by processes with the same identifier
- Process that replaces a crashed process (coordinator) may decide it has the highest ID
  - Just as another process (which detected the crash) is about to decide that *it* has highest ID
- Two processes may announce themselves as the coordinator **concurrently**

Safety … not so soon [2/2]

- No guarantees on message delivery order
  - Recipients reach different conclusions on which is the coordinator process
- E1 may also be broken if timeout values are inaccurate
  - If the process' failure detector is unreliable

A scenario where safety is violated due to inaccurate failure detection

- p_3 had not failed but was just *running slowly*
- p_2 sends its `coordinator` message, and p_3 does the same
  - p_2 receives this after it has sent its message
  - Sets $elected_2$ to p_3
- p_1 receives p_2's message after p_3's
  - Sets $elected_1$ to p_2

Performance of the algorithm

- Best case
  - 2nd highest identifier notices coordinator failure
    - Elects itself immediately and sends (N-2) `coordinator` messages
    - Turnaround time is 1 message
- Worst case requires $O(N^2)$ messages
  - Process with the lowest ID first detects failure
  - (N-1) processes begin elections … each sending messages to processes with higher identifiers

Elections in Wireless Environments

Elections in wireless environments [Vasudevan's algorithm]

- Solution can handle failing nodes and partitioning networks
- We will look at simplified approach
  - Ad hoc networks … but the nodes are not allowed to move physically

Wireless ad hoc network setting

- Each node can initiate election by sending `election` message to its immediate neighbors
- These are neighbors in its **range**

Forwarding of `election` messages and parent-child relationships

- When node receives an `election` message for first time
  - Designates the sender as **parent**
  - Sends out `election` message to all its neighbors except the parent
- When a node receives an `election` message from a node other than its parent
  - Merely acknowledge receipt of the message

When a node R has designated Q as its parent

- Forward `election` message to immediate neighbors (except Q)
- **Wait** for acknowledgements to come in *before* acknowledging `election` message from Q

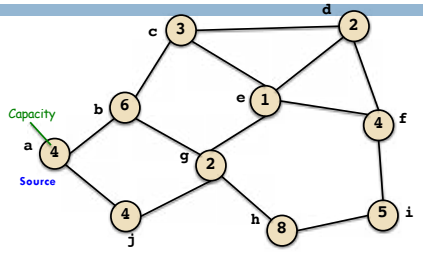
But why wait?

- Neighbors that already have a parent will immediately respond to R
- If all neighbors have a parent?
  - R is a leaf node and will be able to report back to Q quickly
- Report information such as battery lifetime and other resource capacities
  - Allows Q to **compare** R's capacities to that of *other downstream nodes*
  - Select best eligible node for leadership

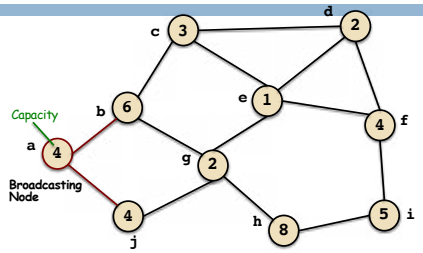
But Q has sent an election message only because its parent P has

- When Q eventually acknowledges `election` message previously sent by P
  - It will pass most eligible node to P as well
- Source will know which node is best to be selected as a leader
  - Broadcast this information to all the other nodes

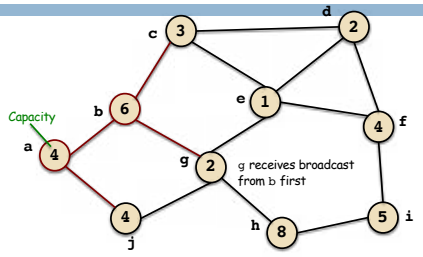
Election algorithm in a wireless network



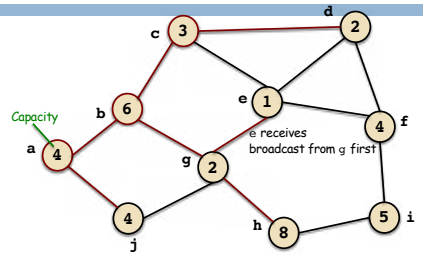
Election algorithm in a wireless network



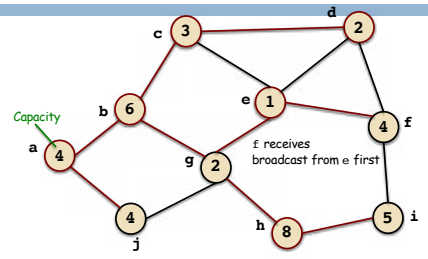
Election algorithm in a wireless network



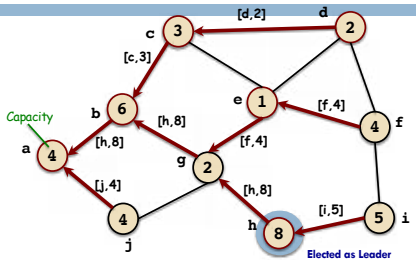
Election algorithm in a wireless network



Election algorithm in a wireless network



Election algorithm in a wireless network



Coping with situations when multiple elections are initiated

- Each source tags its `election` message with a unique identifier
- Nodes participate in elections with the highest identifier
  - Stopping participation in other elections

Architectures & Topology

What we will look at

- Architectural styles for designing systems
  - Layered, objects, data, and event based
- Topologies
  - The role they play in systems design
- Implications:
  - Throughput, scaling, fault tolerance and resiliency, latencies

Architectural Styles

Components are the building blocks of distributed systems

- Modular units
- Well defined-interfaces
- **Replaceable**
- Connectors
  - Mediate communications and coordination between components

Architectural style of distributed systems are formulated in terms of components

- How they are **connected** to each other
- How they **exchange** data
- How they are **configured** into a system

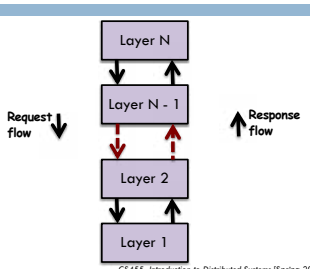
Broad architectural styles

- Layered
- Object-based
- Data-centric
- Event-based

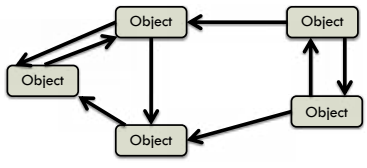
Layered architecture

- Components are organized in a layered fashion
- Component at layer L_i can call components at layer L_{i-1}
- Widely adopted in the networking community

Requests go down the hierarchy; results flow upward



Object-based: Objects are components, connected via (remote) procedure calls



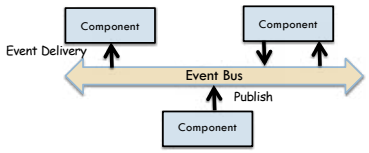
Data centered architectures

- Processes communicate through a **shared repository**
  - Shared distributed file system
  - Shared Web-based data services

Event-based architectures

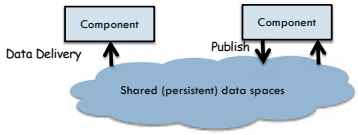
- Communication is via events
- Processes are **loosely-coupled**
  - Don't need to be aware of each other
  - Only specify what you need
- **Middleware** decides what goes where
  - Event routed to processes that are interested in them

Event-based architectures



Shared data spaces: Data-centric plus Event-based

- Processes are **time-decoupled**
  - No need to be active simultaneously
  - Consumers may be offline



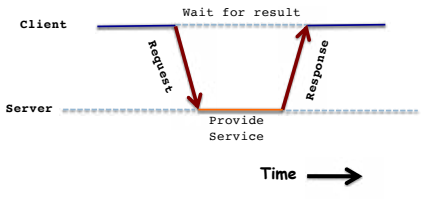
System Architectures

Client Server architecture

- Server **implements** a service
- Client **requests** the service
  - Send request
  - Await server response

} Request-reply semantics

Interaction between a client and a server



Communications between the client and server

- Could be based on a connectionless, unreliable protocol
- But that means dealing with occasional transmission failures
  - Difficult!

Why dealing with occasional failures is difficult

- Is resending messages enough?
- Client **cannot detect** whether
  - Original message was lost OR
  - The transmission of the reply failed
    - If request is resent, operation will be performed twice

Idempotent operations are those that can be repeated many times

- How much do I have in my checking account?
  - Idempotent
- Transfer $10,000 from my bank account
  - Not idempotent

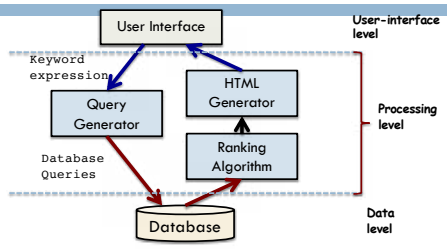
Solution is to use reliable connection-oriented protocols

- Most Internet application protocols are based on TCP/IP
  - Client requests service after setting up connection
  - Server uses **same** connection to send a response
- Issues
  - Setting up and tearing down connection is costly
    - Even more so for small requests and responses

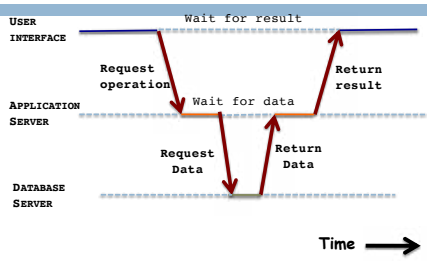
Demarcation of client-server roles is an issue

- Server for a distributed database
  - Forwards requests to file servers that manage the database table
  - The server itself acts as a client
- Suggested layers include
  - **User**-interface level
  - **Processing** level
  - **Data** level

An example of a 3-tier application



Timing diagram in such a setting



Client-server and variants

- **Vertical** distribution
- Tiers correspond to logical organization of applications
- Logically different components reside on different machines

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

- *Distributed Systems: Concepts and Design.* George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011 [Chapter 15]
- *Distributed Systems: Principles and Paradigms. Andrew S. Tanenbaum and Maarten Van Steen. 2nd Edition. Prentice Hall. ISBN: 0132392275/978-0132392273* [Chapter 2, 6]