Welcome (Back)!

Distributed Leader Election

Porter Shawver

Outline

Housekeeping

- Join the Discord!

Section 1

Admins in No Particular Order

Porter

- Master's Student in B.S.-M.C.S; Math Minor
- SWE internship at Palantir this summer; returning next year
- Taking CS 473 this semester
- Fourth semester as a SIGma Admin
- Triathlon Club captain

Ian

- Junior in Stats & CS
- CA for CS 473
- First semester as SIGma Admin
- Research in network science

Sasha

- Junior double major in CS and Physics
- CA for CS 374 and CS 473
- Third semester as SIGma Admin
- Research in quantum cryptography
- Founder of QueerCoded

Franklin

- Senior in Math & CS
- SWE internship at Meta this summer
- Taking CS 574 this semester
- Third semester as a SIGma Admin
- Competitive programming enthusiast

Section 2

Distributed Computing Basics

What is "Distributed Computing"?

> A collection of entities, each of which is autonomous, programmable, asynchronous, and failure-prone, and which communicate through an unreliable communication medium.
>
> — Indranil Gupta (CS 425)

Examples

- Cloud computing.
  - Within datacenters.
  - Between datacenters.

Examples

- Cloud computing.
  - Within datacenters.
  - Between datacenters.
- Multi-core machines.

Examples

- Cloud computing.
  - Within datacenters.
  - Between datacenters.
- Multi-core machines.
- Public networks.
  - Blockchain (e.g. cryptocurrency).
  - Tor routing.
  - Napster, LimeWire.

Common Challenges

- No synchronized clocks.
- Very large number of hosts.
- Lots of failures. How to detect failures?
- Variable bandwidth, arbitrary network delay.

Common Challenges

- No synchronized clocks.
- Very large number of hosts.
- Lots of failures. How to detect failures?
- Variable bandwidth, arbitrary network delay.
- **Who gets to make decisions?**

Questions?

Section 3

Leader Election v1

Leader Election

- It is often useful to have one specially dedicated node.

Leader Election

- It is often useful to have one specially dedicated node.
  - Many algorithms need to be initiated by one node.

Leader Election

- It is often useful to have one specially dedicated node.
  - Many algorithms need to be initiated by one node.
  - One node to communicate between data centers.

Leader Election

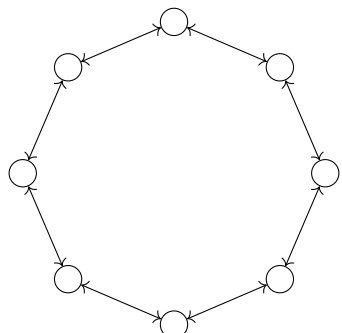
- It is often useful to have one specially dedicated node.
  - Many algorithms need to be initiated by one node.
  - One node to communicate between data centers.
  - Some computation only needs to be done at one machine; why repeat?

Leader Election

- It is often useful to have one specially dedicated node.
  - Many algorithms need to be initiated by one node.
  - One node to communicate between data centers.
  - Some computation only needs to be done at one machine; why repeat?
- Goal: elect a leader upon which all non-faulty processes agree.
  - Multiple nodes can call for an election simultaneously. Result should be the same.

Ring Election

- Nodes are organized in a logical, bi-directional ring (they might not be physically connected in a ring).



- We will assume asynchrony, no node or link failures.

Is this possible?

- True or false? "If processes are identical, no deterministic algorithm can solve leader election in a ring."

Is this possible?

- True or false? "If processes are identical, no deterministic algorithm can solve leader election in a ring."
- True! ...Why?

Is this possible?

- True or false? "If processes are identical, no deterministic algorithm can solve leader election in a ring."
- True! ...Why?
- Proof outline:
  - All nodes have same initial state.
  - If all nodes send the same messages to the left and to the right, all nodes receive the same messages, so no single node can be determined to be *special*.
  - Proof by induction.

Is this possible?

- True or false? "If processes are identical, no deterministic algorithm can solve leader election in a ring."
- True! ...Why?
- Proof outline:
  - All nodes have same initial state.
  - If all nodes send the same messages to the left and to the right, all nodes receive the same messages, so no single node can be determined to be *special*.
  - Proof by induction.
- If we assume each node has a unique ID, then the problem is solvable.

Chang-Roberts Algorithm

- Send IDs clockwise, relay smaller IDs only.
- If receiving own ID, output “leader” & send "terminate".
- If receiving “terminate”, relay it & output “not leader”.

Chang-Roberts Pseudocode

```
ChangRoberts():
    Send self.ID to clockwise neighbor
    Upon receiving ID from counter-clockwise neighbor
        If ID < self.ID:
            send ID to clockwise neighbor
        If ID == self.ID:
            send "terminate" to clockwise neighbor
            output "leader" and terminate
    Upon receiving "terminate":
        send "terminate" to clockwise neighbor
        output "not leader" and terminate
```

Aside: What does "output" mean?

Output to whom?

How do we know this is correct?

- *Safety*: something bad will never happen.
  - *No two nodes output "leader".* That would be bad.
- *Liveness*: Something good will happen eventually.
  - In an asynchronous model, "eventually" may be a *very* long time. (Aside: why are we okay with this?)
  - *Every node outputs.*

How do we know this is correct?

- *Safety*: something bad will never happen.
  - *No two nodes output "leader".* That would be bad.
- *Liveness*: Something good will happen eventually.
  - In an asynchronous model, "eventually" may be a *very* long time. (Aside: why are we okay with this?)
  - *Every node outputs.*
- Fairly straightforward to prove Chang-Roberts satisfies both safety and liveness.

How fast is this? Big-O Analysis.

- Big-O analysis is used to analyze the *worst-case* runtime of an algorithm without getting bogged down by relatively insignificant details.

How fast is this? Big-O Analysis.

- Big-O analysis is used to analyze the *worst-case* runtime of an algorithm without getting bogged down by relatively insignificant details.
  - If an algorithm is $O(n)$, runtime grows linearly to the size of the input.

How fast is this? Big-O Analysis.

- Big-O analysis is used to analyze the *worst-case* runtime of an algorithm without getting bogged down by relatively insignificant details.
  - If an algorithm is $O(n)$, runtime grows linearly to the size of the input.
  - For $O(n!)$ algorithms, runtime grows factorially to the size of the input.

How fast is this? Big-O Analysis.

- Big-O analysis is used to analyze the *worst-case* runtime of an algorithm without getting bogged down by relatively insignificant details.
  - If an algorithm is $O(n)$, runtime grows linearly to the size of the input.
  - For $O(n!)$ algorithms, runtime grows factorially to the size of the input.
  - $O(n^2) = O(\frac{n^2}{2}) = O(2000 \cdot n^2 + n \log n + 37) = O(c \cdot n^2)$

How fast is this? Big-O Analysis.

- Big-O analysis is used to analyze the *worst-case* runtime of an algorithm without getting bogged down by relatively insignificant details.
  - If an algorithm is $O(n)$, runtime grows linearly to the size of the input.
  - For $O(n!)$ algorithms, runtime grows factorially to the size of the input.
  - $O(n^2) = O(\frac{n^2}{2}) = O(2000 \cdot n^2 + n \log n + 37) = O(c \cdot n^2)$
- This type of thinking works because computers operate at scale.

Questions?

How fast is Chang-Roberts?

- Time complexity (# messages between start and finish):

How fast is Chang-Roberts?

- Time complexity (# messages between start and finish): $O(n)$
  - The smallest ID travels n hops to get back to the leader, then "terminate" travels n hops.

How fast is Chang-Roberts?

- Time complexity (# messages between start and finish): $O(n)$
  - The smallest ID travels n hops to get back to the leader, then "terminate" travels n hops.
- Message complexity (# messages total):

How fast is Chang-Roberts?

- Time complexity (# messages between start and finish): $O(n)$
  - The smallest ID travels n hops to get back to the leader, then "terminate" travels n hops.
- Message complexity (# messages total): $O(n^2)$
  - If every node starts an election at the same time, and IDs are ascending, each of the n IDs travels n hops.

Can we do better?

Any ideas?

Hirschberg-Sinclair Algorithm

- Send IDs in phases with doubling distances.
  - In phase p, winners of phase $p-1$ send their ID to both neighbors with a counter set to 2^p.
  - Upon receiving an ID, decrement the counter and pass it along.
  - Upon receiving an ID with counter $= 0$, send an endorsement back.
  - Upon receiving an endorsement for self, win the round, proceed to next phase.
  - Upon receiving own ID, output “leader” & send "terminate".

Hirschberg-Sinclair Correctness

- Similarly straightforward.

Hirschberg-Sinclair Correctness

- Similarly straightforward.
- Saftey.

Hirschberg-Sinclair Correctness

- Similarly straightforward.
- Saftey. Exactly one node can—and will—win every phase.

Hirschberg-Sinclair Correctness

- Similarly straightforward.
- Saftey. Exactly one node can—and will—win every phase.
- Liveness.

Hirschberg-Sinclair Correctness

- Similarly straightforward.
- Saftey. Exactly one node can—and will—win every phase.
- Liveness. Every process outputs on "terminate".

Hirschberg-Sinclair Efficiency

- What got better?

Hirschberg-Sinclair Efficiency

- What got better?
- Time complexity: $2 \cdot (1 + 2 + 4 + 8 + \cdots + n) = O(n)$.

Hirschberg-Sinclair Efficiency

- What got better?
- Time complexity: $2 \cdot (1 + 2 + 4 + 8 + \cdots + n) = O(n)$.
- Message complexity:

Hirschberg-Sinclair Efficiency

- What got better?
- Time complexity: $2 \cdot (1 + 2 + 4 + 8 + \cdots + n) = O(n)$.
- Message complexity:
  - In phase p, at most $\frac{n}{2^p}$ nodes send their IDs.

Hirschberg-Sinclair Efficiency

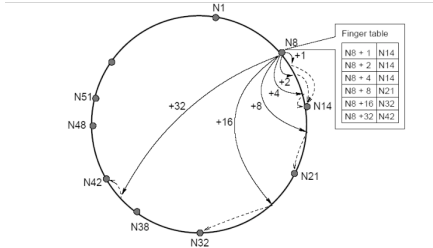
- What got better?
- Time complexity: $2 \cdot (1 + 2 + 4 + 8 + \cdots + n) = O(n)$.
- Message complexity:
  - In phase p, at most $\frac{n}{2^p}$ nodes send their IDs.
  - In each phase, each node causes at most $4 \cdot 2^{p+1}$ messages to be sent.

Hirschberg-Sinclair Efficiency

- What got better?
- Time complexity: $2 \cdot (1 + 2 + 4 + 8 + \cdots + n) = O(n)$.
- Message complexity:
  - In phase p, at most $\frac{n}{2^p}$ nodes send their IDs.
  - In each phase, each node causes at most $4 \cdot 2^{p+1}$ messages to be sent.
  - This implies $8n$ messages per phase.
  - At most $\log n$ phases.
  - $O(n \log n)$ ($MUCH$ better than $O(n^2)$.)

Why do we care about using a ring?

- Flexible; easy to add nodes in the middle of the ring with *consistent hashing*.
- We can get tree-like efficiency with *finger tables*:



- Why not just use a tree?

Bibliography I

- CS 539 "Distributed Algorithms" lecture slides by Ling Ren (Illinois professor).
- My CS 425 "Distributed Systems" notes, based on lecture slides by Indranil Gupta (Illinois professor).