Lecture 21
Distributed Algorithms
Outline

• Models

• Three problems:
  – Leader election in a ring
    • Impossibility (when no unique identifiers)
    • Algorithm (when have unique identifiers)
  – BFS
  – Maximal Independent Set
Distributed algorithms

- Algorithms that are supposed to work in distributed networks, or on multiprocessors.

- Possible tasks:
  - Communication
  - Data management
  - Resource management
  - Synchronization
  - Reaching consensus
Distributed algorithms: Difficulties

- Concurrent activity at many locations
  - sensornets
- Wired or wireless
- Uncertainty of timing, order of events, inputs
- Failure and recovery of processors, communication channels.
Synchronous network model

- Processes at nodes of a network digraph,
  - May have parts of input at each node

- Links connect some process pairs
  - Send messages only along links
  - Digraph: $G = (V,E)$, $n = |V|$  
    - Each node knows its in-neighbors and out-neighbors

- Executes in rounds – in each round:
  - Send and collect messages
  - Local processing
Today’s model:

- Processors know immediate neighbors, might not know global topology*

- Assume common “clock”
  - Ignore synchronization/timing issues

- No processor or link failures

- Unbounded computation at nodes*
  - Focus on communication

* different than our assumptions on parallel processing
Leader election

• Want to distinguish exactly one process as the “leader”.
  – i.e., leader outputs “I am the leader” and no one else outputs anything
  – (slightly more work – others output “I am not the leader”)

• Motivation: Gives a “centralized control”. Leader can take charge of:
  – Communication
  – Coordinating data processing (e.g., in commit protocols)
  – Allocating resources
  – Scheduling tasks
  – Coordinating consensus protocols
  – ...
Simple case: Ring network

- Ring network:
  - Bidirectional links
  - Processes don’t know the numbers;
    - know neighbors by the names “clockwise” and “counterclockwise”.

- Theorem: If all processes are identical, it’s impossible to elect a leader.
Proof of Theorem

• By contradiction. Assume an algorithm that solves the problem.

• State of processor:
  – Program counter
  – Communication history

• All processors start in the same state
• All processes are in identical states after $r$ rounds. By induction:
  – Generate same messages, to corresponding neighbors.
  – Receive same messages.
  – Follow the program code identically.

• Since the algorithm solves the leader election problem, someone eventually gets elected.
• Then everyone gets elected, contradiction.
So we need something more...

- Assume processes have unique identifiers (UIDs)
  - Formally, each process starts with its own UID in local memory
  - Allows to distinguish the processes

- UIDs can appear anywhere in the ring, but each can appear only once.
Idea for algorithm

• Choose max valued UID to be leader!
  – Send value of the biggest UID seen so far clockwise
  – If ever receive your own value from neighbor, then your value made it around the ring, so you must be max valued!
Leader election algorithm
[LeLann] [Chang, Roberts]

- Initialize variable $send_{\text{max}}$ to UID
- Repeat until leader found:
  - Each process sends $send_{\text{max}}$ clockwise
  - Each process receives $incoming_{\text{max}}$ from neighbor, compares with $send_{\text{max}}$.
    - If $incoming_{\text{max}}$ is:
      - Bigger, $send_{\text{max}} \leftarrow incoming_{\text{max}}$
      - Smaller, discard ($send_{\text{max}}$ stays the same)
      - Equal to UID, process declares itself leader!
Comments:

• Works even if:
  – Unidirectional communication (clockwise)
  – Processes don’t know n
Correctness proof

• Prove that exactly one process ever gets elected leader.

• More strongly:
  – Let $i_{\text{max}}$ be the process with the max UID, $u_{\text{max}}$.
  – Prove:
    • $i_{\text{max}}$ outputs “leader” by end of round $n$.
    • No other process ever outputs “leader”.
i_{\text{max}} \text{ outputs } “\text{leader}” \text{ after } n \text{ rounds}

• Prove by induction on number of rounds $0 \leq r \leq n$:

• **Lemma:** After $r$ rounds, $send_{\text{max}}$ at all process within $r$ steps clockwise of $i_{\text{max}}$ contain $u_{\text{max}}$.

• **Key fact:** $i_{\text{max}}$ uses arrival of $u_{\text{max}}$ as signal to set its status to leader.
Uniqueness

• Claim: No one except \(i_{\text{max}}\) ever outputs “leader”.
• Why?
  – For \(j\) other than \(i_{\text{max}}\), \(u_j\) doesn’t get past \(i_{\text{max}}\) when moving around the ring.
  – So \(u_j\) doesn’t reach \(j\)
  – no one except \(i_{\text{max}}\) ever receives its own UID, so no one else ever elects itself.
Complexity bounds

• What to measure?
  – Time = number of rounds until “leader”: $n$
  – Communication = number of single-hop messages: $\leq n^2$

• Can get $O(n \log n)$ message algorithm
General Synchronous Networks
(not just rings)
General synchronous network assumptions

• Digraph $G = (V, E)$:
  – $V$ = set of processes
  – $E$ = set of communication channels
  – $\text{distance}(i,j) = \text{shortest distance from } i \text{ to } j$
  – $\text{diam} = \max \text{ distance}(i,j) \text{ for all } i,j$
  – Assume: Strongly connected (diam is finite), UIDs

• Processes communicate only over digraph edges.

• Don’t know the entire network (not even size!), just local neighborhood.
Breadth-first search

• **Given:** Distinguished source node \( i_0 \).

• **Find:** Breadth-first spanning tree, rooted at source node \( i_0 \).
  – Spanning: Includes every node.
  – Breadth-first: Node at distance \( d \) from \( i_0 \) appears at depth \( d \) in tree.
  – **Form of output:** Each node (except \( i_0 \)) sets a parent variable to indicate its parent in the tree.
Breadth-first search
Breadth-first search
Breadth-first search algorithm

- **Mark** nodes as they get incorporated into the tree.
  - Initially, only $i_0$ is marked.

- **Round 1**: $i_0$ sends **search** message to out-nbrs.

- **At every round**: An unmarked node that receives a search message:
  - Marks itself.
  - Designates one process from which it received search as its parent.
  - Sends search to out-nbrs at the next round.
Breadth-first search

Round 1 (start)
Breadth-first search

Round 1 (msgs)
Breadth-first search

Round 1 (trans)
Breadth-first search

Round 2 (start)
Breadth-first search

Round 2 (msgs)
Breadth-first search

Round 2 (trans)
Breadth-first search

Round 3 (start)
Breadth-first search

Round 3 (msgs)
Breadth-first search

Round 3 (trans)
Breadth-first search

Round 4 (start)
Breadth-first search

Round 4 (msgs)
Breadth-first search

Round 4 (trans)
Breadth-first search

Round 5 (start)
Breadth-first search

Round 5 (msgs)
Breadth-first search

Round 5 (trans)
Breadth-first search algorithm

- **Mark** nodes as they get incorporated into the tree.
- Initially, only $i_0$ is marked.
- **Round 1**: $i_0$ sends search message to out-nbrs.
- **At every round**: An unmarked node that receives a search message:
  - Marks itself.
  - Designates one process from which it received search as its parent.
  - Sends search to out-nbrs at the next round.
- Yields a BFS tree because all the branches are created synchronously.

**Complexity**: Time = diam + 1; Messages = $|E|$
Applications of BFS

- Message broadcast:
  - establish a BFS tree, with child pointers, then use it for broadcasting.
  - Can reuse the tree for many broadcasts
  - Each takes time only $O(\text{diameter})$, messages $O(n)$.
Applications of BFS

- Global computation:
  - Sum, max, or any kind of data aggregation:
  - Complexity: Time $O(\text{diameter})$; Messages $O(n)$

- Leader election (without knowing diameter):
  - Everyone starts BFS, determines max UID.
  - Complexity: Time $O(\text{diam})$; Messages $O(n \cdot |E|)$

- Compute diameter:
  - All do BFS.
  - find height of each BFS tree.
  - find max of all heights.
Maximal Independent Set
Maximal independent set
Maximal Independent Set

- Subset $I$ of vertices $V$ of undirected graph $G = (V,E)$ is \textbf{independent} if no two neighbors are in $I$.

- Independent set $I$ is \textbf{maximal} if no strict superset of $I$ is independent.
  - But might not be maximum!
Maximal independent set
Maximal Independent Set

- **Assume:** (Today only) max degree $d$ known to all processors

- **Output:**
  - Compute an MIS $I$ of the network graph.
  - Each process in $I$ should output winner, others output loser.
Notes

• Slight modification of algorithm (and bigger modification of analysis) gives $O(\log n)$ round algorithm on general graphs

• Needs no UID’s

• Can be made “asynchronous”
Application 1

• Maximal Matching –
  – Matching where no edge can be added without violating “matching” property
    • Might not be maximum matching!
  – Find via MIS:
    • From $G=(V,E)$ construct $G' = (V',E')$
      – let $V' \leftarrow E$ and $E'$ be pairs of nodes in $V'$ whose respective edges in $G$ are adjacent
      – MIS in $G'$ gives a maximal matching in $G$
Application 2

• Vertex coloring
  – Can color nodes of a graph with at most \( (\text{max\_degree} + 1) \) colors
Application 2’

Wireless network transmission:

– Nodes broadcast messages, all neighbors in “receive mode” receive

– Let nodes in the MIS transmit messages simultaneously, others receive.
  
  • Independence guarantees that all transmitted messages are received by all neighbors (since neighbors don’t transmit at the same time).
    
    – Neglecting collisions here---assume receiver receives everything that reaches it.
  
  • Maximality ensures that everyone either transmits or receives something.
Another application:

• Fruit Fly nervous system cell differentiation
  – Sensory Organ Precursor cells do not connect to each other – form MIS!