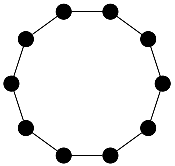
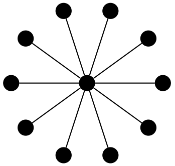
Parallel Programming
Programowanie równoległe

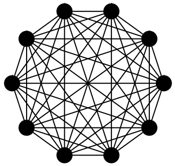
Lecture 4: Distributed algorithms.

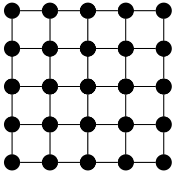
Paweł Rzążewski

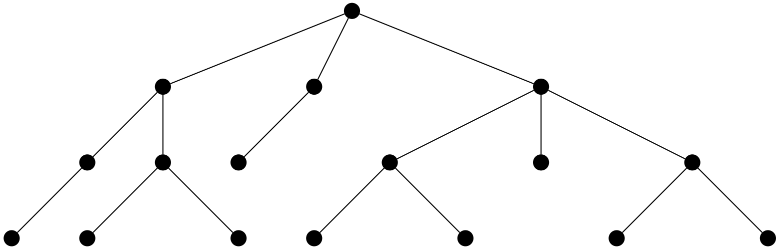
Typical network topologies


ring


star


fully connected


grid


hierarchical

Synchronous network model

There are several models of computation in the case of distributed algorithms. The one we are going to use is **Synchronous Network Model**.

The system consists of a number of computing units (processes) placed in the vertices of some directed graph (network). Each process communicates only with its neighbors.

Also, usually, each process knows its own id. The id's can be any numbers, provided that they are unique (note that they do not have to be consecutive integers!)

Synchronous operations

The computation is divided into rounds. Each round consists of two stages:

1. Perform some local computation. Prepare the messages to be sent to the neighbors.
2. Send all messages to out-neighbors. Receive all messages from in-neighbors. This is performed in one step (there is no particular order of messages received).

Sometimes, for simplicity, we will assume that the process sends a message to every neighbor each turn. The messages with no meaning will be called *null-messages*. When a process receives a null-message, it just ignores it.

Complexity measures

When analyzing distributed algorithms we shall usually be interested in two functions.

Time complexity is the number of rounds needed to perform the computation. Note that in this model we do not care about the time of local computations during each round (which may sometimes be unrealistic).

Complexity measures

When analyzing distributed algorithms we shall usually be interested in two functions.

Time complexity is the number of rounds needed to perform the computation. Note that in this model we do not care about the time of local computations during each round (which may sometimes be unrealistic).

Communication complexity is the number of non-null messages sent during the computation. Sometimes, if more relevant, we will be interested in the total length (in bits) of all messages.

Complexity measures

When analyzing distributed algorithms we shall usually be interested in two functions.

Time complexity is the number of rounds needed to perform the computation. Note that in this model we do not care about the time of local computations during each round (which may sometimes be unrealistic).

Communication complexity is the number of non-null messages sent during the computation. Sometimes, if more relevant, we will be interested in the total length (in bits) of all messages.

In general, we will try to minimize the time complexity. However, high communication complexity can be a real issue. Usually there are several distributed algorithms running in the same network and sharing the same network bandwidth.

Synchronous vs. asynchronous systems

The synchronous model may seem too simple at first glance. However, we can simulate it on an asynchronous one using the algorithms called synchronizers. Basic synchronizers are:

- synchronizer alpha – low time complexity, high communication complexity,
- synchronizer beta – high time complexity, low communication complexity,
- synchronizer gamma – moderate time complexity, moderate communication complexity.

Typical problems in distributed systems

In distributed systems we encounter some problems, which are very easy in the case of local algorithms. Typical ones are:

- choosing a coordinator (selecting a process to perform some special task)
- choosing local coordinators (selecting a group of processes, such that every vertex is either a coordinator, or is adjacent to a coordinator),
- detecting when the computation has finished,
- detecting deadlocks,
- setting global time,
- reaching the agreement (i.e. a common value of some variable).

Selecting the leader in a ring

Leader selection

The processes are organized in a ring (uni- or bi-directional). The number of processes in n. Each process has a local variable *status*, which is initially *unknown*. The problem is to reach a situation when exactly one process has $status = leader$.

Selecting the leader in a ring of identical processes

Theorem

If all processes are identical, the problem cannot be solved in a deterministic way.

Selecting the leader in a ring of identical processes

Theorem

If all processes are identical, the problem cannot be solved in a deterministic way.

Proof.

Suppose we have some algorithm solving the problem of selecting a leader in a ring of identical processes. Consider a unique execution of this algorithm. Observe that since the processes are identical, they start in the same state (values of local variables, state of local memory etc.). Moreover, if the processes are in the same state s, in the next round they will all be in the same state s'.
So whenever a process reaches the state with $status = leader$, all other processes are in the same state. □

Selecting the leader in a ring of identical processes

Theorem

If all processes are identical, the problem cannot be solved in a deterministic way.

Proof.

Suppose we have some algorithm solving the problem of selecting a leader in a ring of identical processes. Consider a unique execution of this algorithm. Observe that since the processes are identical, they start in the same state (values of local variables, state of local memory etc.). Moreover, if the processes are in the same state s, in the next round they will all be in the same state s'.
So whenever a process reaches the state with $status = leader$, all other processes are in the same state. □

Breaking the symmetry is a crucial issue in distributed computation!

Selecting the leader in a directed ring – LCR algorithm

Each process has a unique id. The ids do not correspond to positions of processes in the ring!

Algorithm LCR (Le Lann, Chang, Roberts)

```
Algorithm: LCR
s ← my_id
repeat
    Send(s)
    y ← Receive()
    if y > my_id then
        s ← y
    else if y = my_id then
        status ← leader
    else if y < my_id then
        s ← null
```

Selecting the leader in a ring – LCR algorithm

Theorem

The algorithm described above solves the leader selection problem.

What is the complexity?

Theorem

Time complexity of the described algorithm is n.
Communication complexity is $O(n^2)$, average is $O(n \log n)$.

Selecting the leader in a ring – LCR algorithm

Problem 1

Find an assignment of ids, where the communication complexity of the LCR algorithm is $\Omega(n^2)$.
Find an assignment of ids, where the communication complexity of the LCR algorithm is $O(n)$.

Selecting the leader in a ring – LCR algorithm

Problem 1

Find an assignment of ids, where the communication complexity of the LCR algorithm is $\Omega(n^2)$.
Find an assignment of ids, where the communication complexity of the LCR algorithm is $O(n)$.

Problem 2

How to modify the algorithm to solve the following problem: we want to reach the state when exactly one process will have *status* = *leader* and the other processes will have *status* = *non-leader*?

Selecting the leader in a ring – LCR algorithm

Problem 1

Find an assignment of ids, where the communication complexity of the LCR algorithm is $\Omega(n^2)$.
Find an assignment of ids, where the communication complexity of the LCR algorithm is $O(n)$.

Problem 2

How to modify the algorithm to solve the following problem: we want to reach the state when exactly one process will have *status* = *leader* and the other processes will have *status* = *non-leader*?

Problem 3

The processes in the described solution do not halt. How to modify the algorithm so that each process halts? What if we have additional requirement that each process knows the id of the leader after the computation is completed?

Selecting the leader in a bi-directional ring – HS algorithm

Algorithm HS (Hirschberg, Sinclair)

Each message will consist of a triple $(u, \mathit{flag}, \mathit{hop\_count})$, where u is an id of a process, *flag* can be either *out* or *in*, and *hop_count* is an integer.
Each process has the following local variables:
u, which is some process' id, initialized *my_id*,
send+, containing some message or *null*, initialized with $(\mathit{my\_id}, \mathit{out}, 1)$,
send−, containing some message or *null*, initialized with $(\mathit{my\_id}, \mathit{out}, 1)$,
status, which is initially *unknown*, can be changed to *leader*,
phase, which is an integer, initially 0.

Selecting the leader in a bi-directional ring – HS algorithm

Algorithm: HS

```
repeat
    send $send+$ to the right neighbor
    send $send-$ to the left neighbor
    $send+ \leftarrow null$
    $send- \leftarrow null$
    if if the message from the left neighbor is $(v, out, h)$ then
        if $v > u$ and $h > 1$ then $send+ \leftarrow (v, out, h-1)$;
        if $v > u$ and $h = 1$ then $send- \leftarrow (v, in, 1)$;
        if $v = u$ then $status \leftarrow leader$;
    if if the message from the right neighbor is $(v, out, h)$ then
        if $v > u$ and $h > 1$ then $send- \leftarrow (v, out, h-1)$;
        if $v > u$ and $h = 1$ then $send+ \leftarrow (v, in, 1)$;
        if $v = u$ then $status \leftarrow leader$;
    $\cdots$
```

Selecting the leader in a bi-directional ring – HS algorithm

```
Algorithm: HS
1 repeat
2     if if the message from the left neighbor is $(v, in, 1)$ and $v \neq u$ then
3         $send+ \leftarrow (v, in, 1)$
4     if if the message from the right neighbor is $(v, in, 1)$ and $v \neq u$ then
5         $send- \leftarrow (v, in, 1)$
6     if if the messages from both neighbors are $(u, in, 1)$ then
7         $phase \leftarrow phase + 1$
8         $send+ \leftarrow (u, out, 2^{phase})$
9         $send- \leftarrow (u, out, 2^{phase})$
```

Selecting the leader in a bi-directional ring – HS algorithm

Theorem

The algorithm HS solves the leader selection problem in a bi-directional ring.

Selecting the leader in a bi-directional ring – HS algorithm

Theorem

The algorithm HS solves the leader selection problem in a bi-directional ring.

Theorem

The time complexity of the algorithm HS is $O(\log n)$.
The communication complexity of the algorithm HS is $O(n \log n)$.

Selecting the leader in a bi-directional ring – HS algorithm

Theorem

The algorithm HS solves the leader selection problem in a bi-directional ring.

Theorem

The time complexity of the algorithm HS is $O(\log n)$.
The communication complexity of the algorithm HS is $O(n \log n)$.

Theorem

The communication complexity of any algorithm selecting a leader in a bi-directional ring, using only comparisons of ids, is $\Omega(n \log n)$.

Selecting the leader in a bi-directional ring – HS algorithm

Theorem

The algorithm HS solves the leader selection problem in a bi-directional ring.

Theorem

The time complexity of the algorithm HS is $O(\log n)$.
The communication complexity of the algorithm HS is $O(n \log n)$.

Theorem

The communication complexity of any algorithm selecting a leader in a bi-directional ring, using only comparisons of ids, is $\Omega(n \log n)$.

There are algorithms with communication complexity $O(n)$ – they use some additional knowledge on ids.

Selecting the leader in a directed ring with $O(n)$ messages

- the number n of elements of the ring is known,
- ids are integers
- synchronous model

Idea of an algorithm:

Phase $i = 1, 2, \ldots,$

1. If my id is not i, do not initiate any message.
2. Otherwise, send my id to the neighbor.
3. If a message is received, pass it forward.

Selecting the leader in a directed ring with $O(n)$ messages

- the number n of elements of the ring is known,
- ids are integers
- synchronous model

Idea of an algorithm:

Phase $i = 1, 2, \ldots,$

1. If my id is not i, do not initiate any message.
2. Otherwise, send my id to the neighbor.
3. If a message is received, pass it forward.

Time complexity: $n \cdot m$, where m is min id.
Communication complexity n

Selecting the leader in a general network

What happens if the processes form a arbitrary directed graph (general network)?

Selecting the leader in a general network

What happens if the processes form a arbitrary directed graph (general network)?

The diameter of the graph G is $\max\{\mathrm{dist}(u, v) \colon u, v \in V(G)\}$.

Selecting the leader in a general network

What happens if the processes form a arbitrary directed graph (general network)?

The diameter of the graph G is $\max\{\mathrm{dist}(u, v)\colon u, v \in V(G)\}$.

Suppose that the diameter of the network is known for every process.

Selecting the leader in a general network – Flood Max

Algorithm: Flood Max

```
1 $m \leftarrow my\_id$
2 $d \leftarrow$ diameter of the network
3 for $rounds = 1, 2, \ldots, d$ do
4     receive messages from all neighbors
5     $U \leftarrow$ set of received ids
6     $m \leftarrow \max\{U \cup \{m\}\}$
7 if $m = my\_id$ then $status \leftarrow leader$;
8 else $status \leftarrow nonleader$;
```

Theorem

Time complexity of the Flood Max algorithm is d.
Communication complexity of the Flood Max algorithm is $d \cdot |E|$.

Selecting the leader in a general network – Flood Max

Algorithm: Flood Max

```
1 $m \leftarrow my\_id$
2 $d \leftarrow$ diameter of the network
3 for $rounds = 1, 2, \ldots, d$ do
4     receive messages from all neighbors
5     $U \leftarrow$ set of received ids
6     $m \leftarrow \max\{U \cup \{m\}\}$
7 if $m = my\_id$ then $status \leftarrow leader$;
8 else $status \leftarrow nonleader$;
```

Theorem

Time complexity of the Flood Max algorithm is d.
Communication complexity of the Flood Max algorithm is $d \cdot |E|$.

Any ideas for improvements?

Spreading the gossip

Problem

A process i has some information. Design and analyze an algorithm that will send this information to all processes in the network.

Spreading the gossip

Problem

A process i has some information. Design and analyze an algorithm that will send this information to all processes in the network.

Problem

Design and analyze an algorithm to compute the diameter of the network (for simplicity, suppose we have an undirected graph).

Finding a maximal independent set

We want to select a set X of vertices, such that:

- no two vertices from X are adjacent,
- every vertex is either in X or is adjacent to a vertex from X.

Finding a maximal independent set

We want to select a set X of vertices, such that:

- no two vertices from X are adjacent,
- every vertex is either in X or is adjacent to a vertex from X.

Formally, each process from X should set $\mathit{status} = \mathit{selected}$, while every other process should set $\mathit{status} = \mathit{loser}$.

Finding a maximal independent set

Assume that the number of processes n is known to every process and there are no ids.

Problem

Design an algorithm selecting a maximal independent set in a graph.

Finding a maximal independent set

Assume that the number of processes n is known to every process and there are no ids.

Problem

Design an algorithm selecting a maximal independent set in a graph.

We do not know whether a deterministic polylogarithmic algorithm exists (huge open problem stated in 1986)!

Finding a maximal independent set – LubyMIS

The idea of the algorithm LubyMIS (named after its inventor, Michael Luby [1986]) is based on the following procedure:

Algorithm: LubyMIS – idea

```
1 $X \leftarrow \emptyset$
2 **while** $V \neq \emptyset$ **do**
3     choose some non-empty independent set $X' \subseteq V$
4     $X \leftarrow X \cup X'$
5     remove the vertices from $X'$ and their neighbors from $V$
6 **return** $X$
```

Finding a maximal independent set – LubyMIS

The algorithm is randomized. It works in *stages*, each consisting of three rounds.

1. Every process chooses some value v uniformly at random from the set $\{1, \ldots, B\}$ (B is some big number). Then each process sends v to all its neighbors. X' is the set of processes, whose value of v is bigger than the values of all its neighbors (why is it independent? why is it non-empty?).
2. Vertices from X' (winners) notify the neighbors that they are losers.
3. Losers notify their neighbors. Winners and losers finish. The neighbors of losers remove them from their neighborhoods.

Finding a maximal independent set – LubyMIS

The algorithm is randomized. It works in *stages*, each consisting of three rounds.

1. Every process chooses some value v uniformly at random from the set $\{1, \ldots, B\}$ (B is some big number). Then each process sends v to all its neighbors. X' is the set of processes, whose value of v is bigger than the values of all its neighbors (why is it independent? why is it non-empty?).
2. Vertices from X' (winners) notify the neighbors that they are losers.
3. Losers notify their neighbors. Winners and losers finish. The neighbors of losers remove them from their neighborhoods.

What happens if two vertices choose the same value of v?

Finding a maximal independent set – LubyMIS

The algorithm is randomized. It works in *stages*, each consisting of three rounds.

1. Every process chooses some value v uniformly at random from the set $\{1, \dots, B\}$ (B is some big number). Then each process sends v to all its neighbors. X' is the set of processes, whose value of v is bigger than the values of all its neighbors (why is it independent? why is it non-empty?).
2. Vertices from X' (winners) notify the neighbors that they are losers.
3. Losers notify their neighbors. Winners and losers finish. The neighbors of losers remove them from their neighborhoods.

What happens if two vertices choose the same value of v?

Theorem

With probability one, the algorithm LubyMIS terminates. The expected number of rounds is $O(\log n)$ (setting $B = n^4$ is enough).

Finding a maximal independent set – another approach

Here is a simpler and better algorithm [Afek, Alon, Barad, Hornstein, Barkai, Bar-Joseph, *A Biological Solution to a Fundamental Distributed Computing Problem*, Science, 2011]

The algorithm is inspired by the process of development of fly's nervous system.

$n \leftarrow$ upper bound on the number of nodes
$D \leftarrow$ upper bound on the maximum degree
$M \leftarrow$ carefully chosen constant
B **is a special type of message, is consists of 1 bit**

Finding a maximal independent set

```
Algorithm: MIS
for $i \leftarrow 0$ to $\log D$ do
    for $j \leftarrow 0$ to $M \log n$ do
        (1st exchange)
        $v \leftarrow 0$
        if a success with probability $1/2^{\log D - i}$ then
            broadcast $B$ to all neighbors
            $v \leftarrow 1$
        if received a message $B$ then $v \leftarrow 0$;
        (2nd exchange)
        if $v = 1$ then
            broadcast $B$ to all neighbors
            status $\leftarrow$ selected
        else if received $B$ then
            status $\leftarrow$ loser
```

Finding a maximal independent set

With high probability, all nodes are either selected or losers.

Finding a maximal independent set

With high probability, all nodes are either selected or losers.

Time complexity: $O(\log n \log D)$, worst case: $O(\log^2 n)$.
Communication complexity: $O(n)$, one-bit messages.

Finding a maximal independent set

With high probability, all nodes are either selected or losers.

Time complexity: $O(\log n \log D)$, worst case: $O(\log^2 n)$.
Communication complexity: $O(n)$, one-bit messages.

Communication complexity is optimal: each node has to receive at least one message!