h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·so GENÈVE
Haute École Spécialisée
de Suisse occidentale

Distributed systems

Professor : Nabil Abdennadher

Introduction to distributed algorithms

September 2020

If you find a typo, no matter how small, please send an email to nabil.abdennadher@hesge.ch

I. Introduction

This document is dedicated to some of the fundamental distributed algorithms. The goal is to study these algorithms from a theoretical point of view: complexity, number of messages generated and stop conditions. This study assumes that:

1. the network is functional,
2. the communications are message passing based.

The network is represented by an undirected graph where each node can be a computer, a switch, a router, a sensor. Inter-node links are wired or wireless.

II. Building a Spanning Tree (ST)

A Spanning Tree (ST) is a tree structure that "covers" a graph. An ST tree covers a graph G if:

- G and ST have the same nodes,
- the links of ST are a subset of the links of G.

To browse the nodes of a given graph, we only need to browse the nodes of its tree. Indeed, it is easier to write tree-based algorithms than graph-based algorithms: the hierarchical structure of a tree (*parent-son* relationship) is easier than the structure of a graph. An ST tree is represented by its root. Each node of the tree knows its parent and its sons. In a distributed architecture, each node has a partial view of the tree: its parent (NULL for the root node) and its children (NULL for the leaf nodes).

A graph G can be represented by several STs having the same root or different roots. Figure 1 shows a graph G with an example of a ST. The red edges represent the links of the ST. The black edges represent the links of graph G.

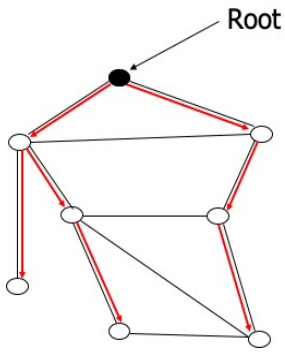

Figure 1 : Example of a graph G and its ST tree

The ST is used, inter alia, to perform broadcast and convergecast operations. Tree-based algorithms are easier and faster to implement than graph-based algorithms.

This section details a distributed algorithm that generates a ST for a given graph G. Each node of the network knows its direct neighbors, in other words the nodes that are directly connected to it. The ST generation starts from a given node of the graph G: n_r. n_r will then be the root of the ST.

Three types of messages are exchanged between the nodes of the network (graph):

1. **M**: This message represents adoption requests. A node n_i sends a message *M* to a neighbour n_j to ask for its adoption.
2. **P**: A node n_i sends a *P* message to a neighbour n_j if n_i accepts the adoption request (*M* message) sent by n_j. The first adoption request is always accepted while those who follow are always refused.
3. **R**: A node n_i sends a *R* message to a neighbour n_j if n_i refuses the adoption request (*M* message) sent by n_j. This means that n_i has already received an adoption request from another node n_k to which it has already sent a *P* message.

II.1 Constructing a width-first search spanning tree for a specified root

When a node n_i receives a first adoption request (*M* message) from n_j, it automatically accepts it and sends a *P* message to n_j. For the *M* messages received later, n_i refuses the adoption request and sends a *R* message. A node n_i can send adoption messages (*M*) only after receiving the first adoption message (*P*). This rule does not apply to the root node. Adoption messages are sent in parallel.

The algorithm for generating the spanning tree (ST) is as follows:

Node n_r (root node)

Sends <M> message to all neighbours

When receiving <M> message from n_i, n_r sends <R> message to n_i

When receiving <P> message from n_i, n_r updates the list of children F_r: $F_r = F_r + n_i$

When receiving <R> message from n_i, n_r updates the list of children: $NF_r = NF_r + n_i$

Node n_i ($i \neq r$)

When receiving a first message <M> from a node n_j

n_i sends a message <P> to n_j

$parent_i = n_j$

n_i sends a message <M> to all its neighbours except n_j (I)

When receiving <M> messages from a node n_j later

n_i sends <R> message to n_j

When receiving <P> message from n_j, n_i updates the list of children: $F_i = F_i + n_j$

When receiving <R> message from n_j, n_i updates the list of children: $NF_i = NF_i + n_j$

Alg. 1: Algorithm for building a width-first search spanning tree

F_i (*resp.* NF_i) is the set of children (*resp.* non-child) nodes of n_i. The nodes belonging to the two sets F_i and NF_i are necessarily neighbours of n_i.

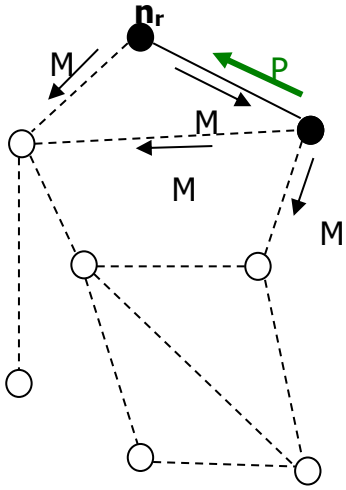

Figure 2 : Sending a *P* message to the first node that sends a *M* message

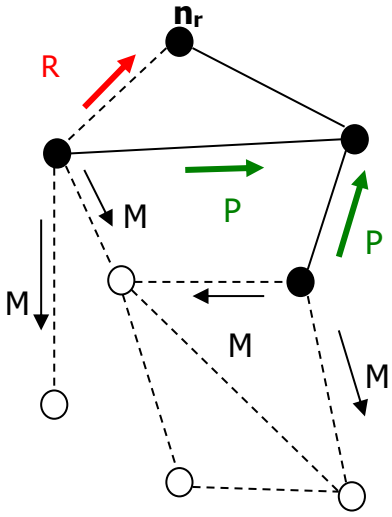

Figure 3 : *P* and *R* messages are returned in response to an adoption request *M*.

When receiving a first *M* message from n_j, n_i (non-root node) sends a *P* message to n_j (its parent) and *M* messages to all its neighbours except its parent n_j (Figure 2). It will therefore receive return messages of type *P* or *R* from all its neighbours other than its parent n_j (Figure 3). The number of received messages of type *P* and *R* is therefore equal to $|v_i| - 1$. v_i is the set of neighbours of n_i. $|v_i|$ is the cardinality of v_i.

The algorithm stops when:

$|v_i| - 1 = |NF_i| + |F_i|$.

For the root node (n_r),the algorithm stops when :

$|v_r| = |NF_r| + |F_r|$.

The stop condition is not explained in the Alg.1 algorithm. This condition is explained in the slides of the course.

The simultaneous sending of several messages of type *M* to all the neighbours (line I in the algorithm) makes it possible to construct "wide" trees. The algorithm favours the adoption of several sons by the same parent. This algorithm is called: width-first search spanning tree.

II.2 Constructing a depth-first search spanning tree for a specified root

The "depth-first search spanning tree for a specified root" is a modified version of the width-first search spanning tree algorithm. The "depth-first search spanning tree for a specified root" builds trees in depth first by sending the adoption messages (M) sequentially. A node sends a new adoption request only upon receipt of the response (P and R). A detailed version of this algorithm is in the slides (see course website).

II.3 Constructing a Depth-First Search Spanning Tree without a specific root

Unlike to two other versions, the root is not known in advance. It is determined on the fly, during the execution of the algorithm. Candidates nodes (nodes that want to be root) start executing the algorithm (N_0 set). Only one ST will be completely built. Its root is the candidate node that has the largest identifier. In this release, adoption requests (M) include the candidate node identifier that initiated the execution. On receiving an adoption request (M), a node n_i compares the identifier of the root of the tree to which it belongs ($leader_i$) to the identifier of the request it has just received (y). Three cases are possible:

1. If y < $leader_i$: the tree that *y* would like to build must be blocked. In this case, node n does not propagate adoption requests. The construction of the tree whose root is *y* will never be finished.
2. y = $leader_i$: node *n* receives another adoption request from *y* (the adoption request is not the first one). n_i then sends a reject message.
3. y > $leader_i$: The adoption request (y) received by the node is greater than the current local variable $leader_i$. The node must then change from the old to the new tree and start sending adoption requests on behalf of the new tree (whose root is *y*)

A detailed version of this algorithm is in the *slides* (see course website).

III. Broadcast algorithms

A broadcast operation transmits the same data D to all nodes of a the network. A broadcast algorithm therefore transmits D, available at one or more nodes, to all the nodes of the network.

III.1 Broadcast by using a Spanning Tree

This algorithm assumes that Spanning Tree (ST) is already built. Each node n_i holds the following variables:

$parent_i$: parent of n_i,

F_i: set of sons of n_i,

The ST is represented by the root n_r.

An intermediate node (nodes which are not root or leaves) receives data D from his parent and transmits it to his sons. The root node sends the data D to all its children. A leaf node only receives the data from its parent.

Each node of the ST only transmits the data D. The number of transmitted messages is therefore equal to n - 1. *n* being the number of nodes in the graph G (or ST). Indeed, the number of links in ST is equal to n - 1.

III.2 Broadcast by flooding

N_0 is the set of nodes which hold the data D before the execution of the broadcast algorithm. The broadcast algorithm is as follows: a node which does not belong to N_0 waits for the reception of the data D. Once the data D has been received, n_i sends D to all its neighbours including the sending node n_j.

The algorithm is as follows:

```
atteint_i : Boolean, initialized to False.
count_i : Number of received messages, initialized to 0.
If n_i belongs to N_0
      atteint_i = True
      Send D to all direct neighbors (v_i)
Endif
```

```
While (count_i < | v_i |) do
      On receiving of D
            If (atteint_i =False)
                  atteint_i := True
                  Send D to all direct neighbours (v_i)

            Endif
Endwhile
```

The Boolean $atteint_i$ is set to true if:

- the node n_i belongs to N_0, or
- when receiving D for the first time.

Data D is thus broadcasted by "waves" (Figure 4).

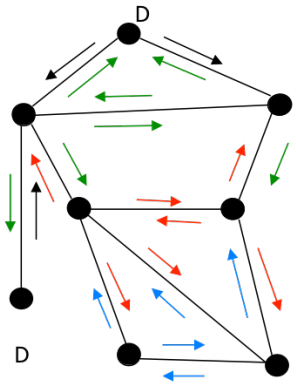

Figure 4 : Broadcast by waves (flooding). Each node creates its own wave.

Each link of the graph transmits data D twice, once in each direction. The number of messages transmitted is therefore equal to *2*m*. *m* being the number of links in the graph G.

III.3 Broadcast by flooding with acknowledgment

This algorithm is a variant of the broadcast algorithm described in section III.2. In this case, the N_0 set is a singleton ($N_0 = \{n_r\}$). The node n_r which initially has the D data must receive a receipt acknowledgment specifying that D has been transmitted correctly. The algorithm is as follows:

```
atteint_i := False
count_i = 0
If n_i belongs to N_0
      atteint_i := true
      Send D to all direct neighbours (v_i)
Endif
On receiving D from n_j
      count_i = count_i + 1;
      If atteint_i =False
            atteint_i := True ;
            parent_i := n_j
            Send D to all direct neighbours (v_i) except n_j
      Endif
```

```
        If count_i = |v_i| and parent_i ≠ NULL
              Send D to parent_i
        Endif
    Whileend
```

Upon receiving the first message (data D) from the node n_j, a node n_i initialises its variable $parent_i$ to n_j. The counter $count_i$, initialized to 0, is incremented by 1 at each reception of the data D. The algorithm stops when $count_i$ is equal to the number of neighbours, the data D is then transmitted to the parent of the node n_i (this message is considered as an receipt acknowledgment).

The number of messages sent by a node is equal to the number of its neighbours. The total number of messages is therefore equal to *2*m.* where *m* is the number of edges in the graph. The complexity of this algorithm is the same as the "Broadcast by Wave (flooding)" algorithm. However, its execution is slower since each node is waiting to receive all messages from its neighbours before sending the last message to the node that sent it first the D data.

At the end of its execution, the “broadcast algorithm with acknowledgment” builds a spanning tree. The parent of a given node is the one which sends it the first message. If each node knows its parent, then it is possible to build the ST.

IV. Convergecast Algorithm

A convergecast algorithm gathers data from one or more nodes of the graph. Each node has a local data D_i. At the end of the execution of the convergecast algorithm, all the D_i data are stored at the n_r node. It is assumed that an ST whose root is n_r, is already known. Each node of the tree knows its sons and its father. Let’s remind here that the root node has no father and the child nodes have no children. The leaf nodes start be sending their data to their parent. Upon receiving all data from their sons, an intermediate node (a node that is neither parent nor root) sends all the received data to his parent. Finally, the root node hangs until it receives all data from its sons nodes. The algorithm is as follows:

```
Each leaf node
      Send its local D_i to his patent
Each non leaf node n_i (i ≠ r)
      On receiving all messages from all sons
            send all received messages to its parent
Root node n_r
      Receives messages from all son nodes
```

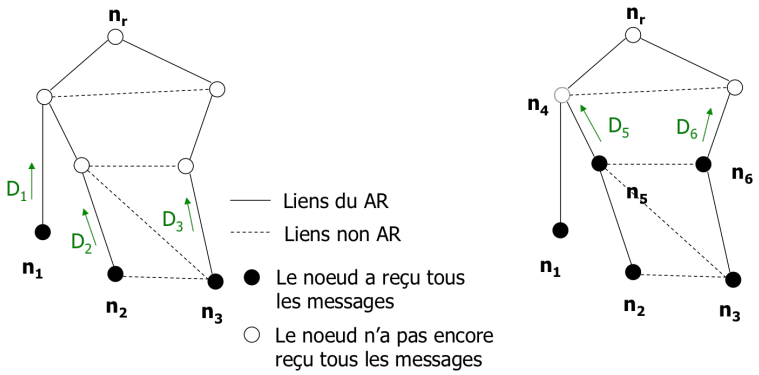

Figure 5 : Step 1 : n_1, n_2, n_3 send D_1, D_2, D_3 to their parents

Step 2 : n_5, n_6 send D_5, D_6 to their parents.