LU3IN013

Research Project

Formal Modeling of Algorithms for Distributed Systems

First Report

Tra My Nguyen
3702069

2019-2020

1 Introduction

Distributed systems are omnipresent nowadays as they help resolving various computing problems with efficiency and high adaptability. However, along these advantages come a lot of difficulties due to the fact that entities in these systems run concurrently and can only communicate with each other via message-passing. As a result, coordinating the behavior of independent components of the system becomes the major challenge in distributed computing and one that is really hard to analyze. Instead of relying on traditional extensive testing, modern day system design uses formal methods based on mathematical techniques to study the reliability and robustness of a system. One proposal is the Carl Adam Petri's seminal work on Petri nets in the early 1960s.

This project is carried out in responding to the Call for Models by the Model Checking Contest handled annually by LIP6, an event where formal verification tools for concurrent systems are evaluated in order to find the best suited techniques for a class of problem (e.g. state space generation, deadlock detection, reachability analysis, causal analysis). The goal is to create formal models with scaling capabilities from distributed algorithms using previously mentioned Petri nets, which may then be used alongside others in the contest as common benchmarks on which all tools will be compared.

In this first report, I will be giving a general view on the notion of Petri nets and their functioning principle, as well as an initial application of Petri nets in modeling a Client-Server communication system. I will also be briefly presenting two distributed algorithms from which I will attempt to create Petri nets models and the steps I will take to work toward that goal.

2 Petri Net

Place/Transition Petri nets (P/T net) will be the main focus in this project.

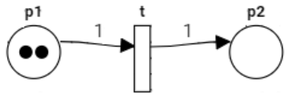


Figure 1: An simple P/T net

The image above shows **a place/transition net (P/T net)**, which, at first glance, resembles a bipartite oriented weighted graph. Its vertices can be separated into a set of rectangle (known as transitions) and a set of circles (places), each circles containing a number of dots (tokens). A state of a P/T net is established by the number of tokens held by each place, formally defined by a marking. Transitions of a P/T net represent actions that can take place and modify its state. The cost of one action is indicated on its input arc (number of tokens taken away from the source place) and its product on the output arc (number of tokens added to the target place).

The models to be produced in the project need to be scalable, which means by increasing a certain type of actors having similar behaviors in a system (parameters), we obtain multiple P/T nets describing the same scenario but of different sizes (number of transitions/places/arcs) or of different initial marking.

2.1 Formal definitions

A P/T net can be hereby formally defined as *a tuple* $\mathcal{N} = (P, T, \mathbf{Pre}, \mathbf{Post})$, *where*
• *P is a finite set of places, representing resources in a system, each place holds a number of tokens, representing the number of occurrences of a resource*
• *T is a finite set of transitions, disjoint from P, representing actions that can occur in the system, and*
• $\mathbf{Pre}$,$\mathbf{Post} \in \mathbb{N}^{|P|\times|T|}$ *are matrices (the backward and forward incidence matrices of $\mathcal{N}$), $\boldsymbol{C} = \boldsymbol{Post} - \boldsymbol{Pre}$ is called the incidence matrix of $\mathcal{N}$.*

Pre	t
p_1	1
p_2	0

Post	t
p_1	0
p_2	1

C	t
p_1	-1
p_2	1

Table 1: The incidence matrices of the P/T net in Figure 1

A marking of a P/T net $\mathcal{N} = (P, T, \mathbf{Pre}, \mathbf{Post})$ is a vector $\mathbf{m} \in \mathbb{N}^{|P|}$. $\mathcal{N}$ *together with a marking* $\mathbf{m}_0$ *(initial marking) is called* **a P/T net system** $\mathcal{S} = (\mathcal{N}, \mathbf{m}_0)$.

It can be said that *a transition $t \in T$ is enabled in a marking* $\mathbf{m}$ *if* $\mathbf{m} \geq \mathbf{Pre}[\bullet, t]$. *In this case* **the successor marking relation** *is defined by* $\mathbf{m} \xrightarrow{t} \boldsymbol{m'} \Leftrightarrow \mathbf{m} \geq \mathbf{Pre}[\bullet, t] \wedge \mathbf{m'} = \mathbf{m} + \mathbf{Post}[\bullet, t] - \mathbf{Pre}[\bullet, t] = \mathbf{m} + \mathbf{C}[\bullet, t]$ *(*$\mathbf{Pre}[\bullet, t]$ *denotes the t-column vector* $\mathbf{Pre}[\bullet, t] = (\mathbf{Pre}[p_1, t], ..., \mathbf{Pre}[p_{|P|}, t])$*) of the* $|P| \times |T|$ *matrix* $\mathbf{Pre}$. *The same holds for* $\mathbf{Post}[\bullet, t]$ *with respect to* $\mathbf{Post}$.

Here, the initial marking is $\mathbf{m}_0 = [2, 0] > \mathbf{Pre}[\bullet, t] = [1, 0]$, the transition t is therefore fireable and firing it will lead to the successor marking $\mathbf{m}_1 = [1, 1]$.

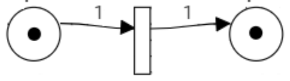


Figure 2: New P/T net after firing transition t

3 An example: Client-Server System

Let's look at the example of an P/T net simulating a communication channel between 2 clients and 2 servers. Each client can send a request to the channel, which will address the request to one of the servers and transfer the reply back to the same client once the chosen server has responded.

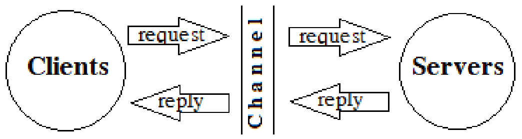


Figure 3: Client-Server communication system

Places:
Channel
• `sys`, waiting for request
• $\texttt{sys\_req}_i$, having request from Client$_i$
• $\texttt{sys\_rep}_i$, having reply for request of Client$_i$
Client$_i$
• $\texttt{c}_i$, wanting to send request
• $\texttt{ca}_i$, waiting for reply
Server$_i$
• $\texttt{sv}_i$, available
• $\texttt{sv}_i\texttt{\_c}_j$, treating request from Client$_j$

Transitions:
Channel and Client$_i$
• $\texttt{send}_i$, Client$_i$ sending request to channel
• $\texttt{receive}_i$, Client$_i$ receiving reply from channel
Server$_i$
• $\texttt{req}_i\texttt{\_c}_j$, receiving request of Client$_j$ from channel
• $\texttt{rep}_i\texttt{\_c}_j$, sending reply to request of Client$_j$ to channel

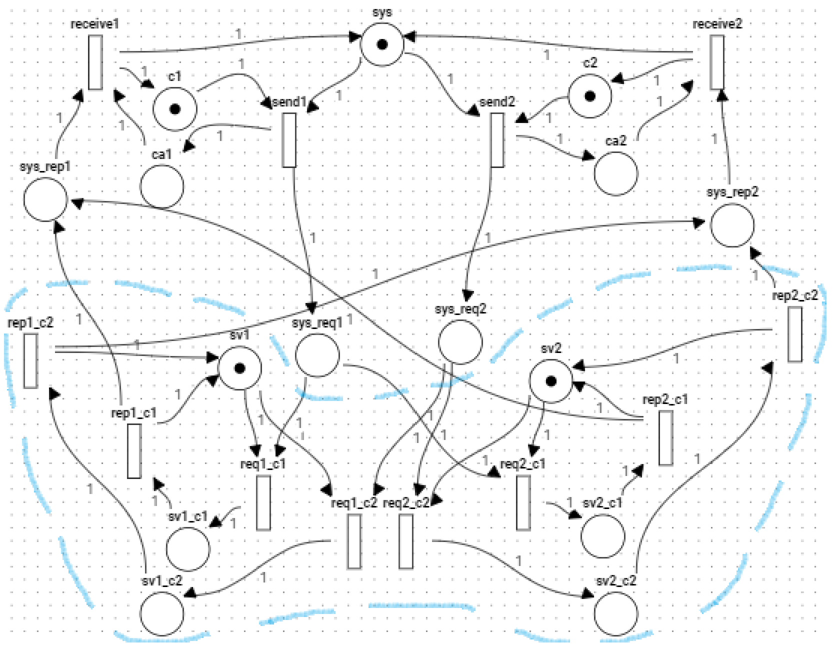


Figure 4: P/T net of 2 Clients and 2 Servers system

It can be seen that for a given server, there is only one token in the initial marking and there are as many sets of {sv$_i$_c$_j$,req$_i$_c$_j$,rep$_i$_c$_j$} as there are clients, for the reason being that a server can take a request from any client and while processing that request, it is no longer available. The channel also processes one request at a time and a client can only send one request at a time. Since the set of places and transitions of all Server$_i$ is transition-bordered (only transitions in the set are linked to the rest of the model), it can be abstracted into one common transition like in the following P/T net.

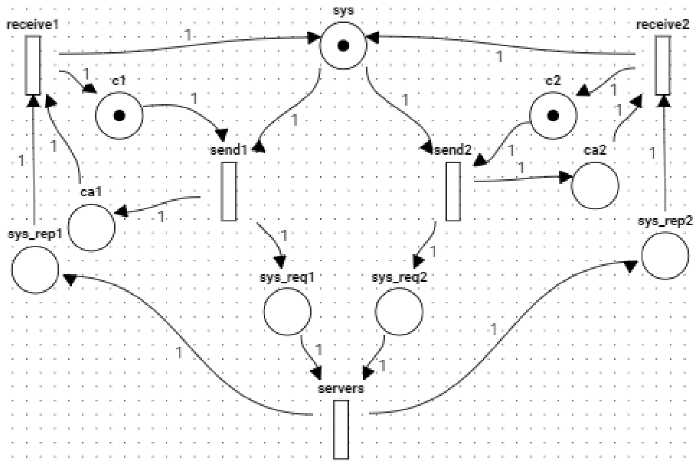


Figure 5: Abstracted P/T net of 2 Clients and 2 Servers system

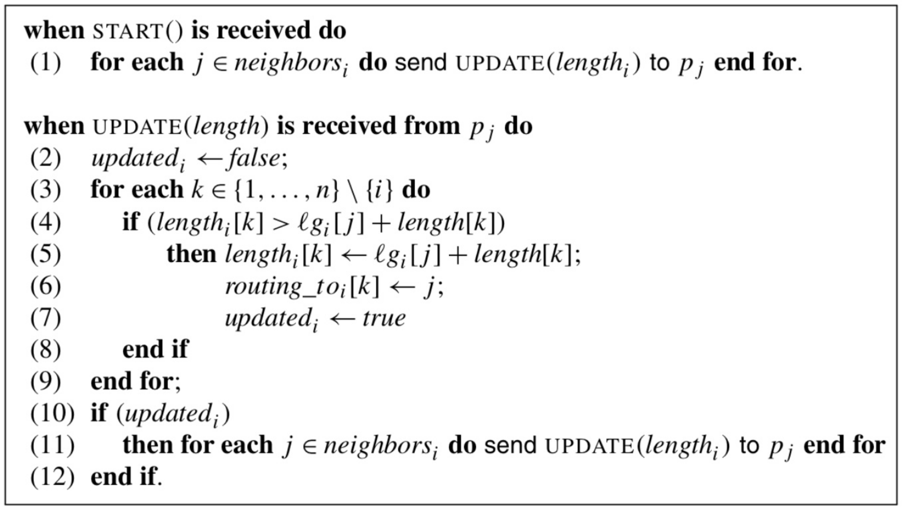
4 Algorithms

Firstly, I will be looking at a classic problem in a distributed system: leader election, where node with the highest identity number in a network would be elected as the leader. The algorithm below, presented by D.S.Hirschberg and J.B.Sinclair(1980), shows the procedure of each node in a bidirectional ring network (where each node of identity id_i has two neighbors $left_i$ and $right_i$ with whom it can communicate (send and receive messages)).
I will be modeling the behavior of each node, how they react to a message from another node. All nodes will receive a START message from exterior. Each node then sends ELECTION$(id_i, 0, 1)$ to both of its neighbors, presenting its own identity, the first round number $r = 0$ and the number of nodes its message has visited $d = 1$. During each round r, each node competes with its 2^r neighbors to the left and to the right, and only the winner of round r can continue to round $r + 1$. After one round, the identity of the winner of a neighborhood will be sent in a REPLY message back to the winner itself, a node receiving two REPLY messages from both sides learns that it is the winner of its $(2^r + 1)$ neighborhood and moves on to the new round. The process ends with the node of highest identity number sending an ELECTION message to itself and becoming the leader, and then, all nodes learning the identity of the leader thanks to the ELECTED message.

```
when START() is received do
 (1)  send ELECTION($id_i$, 0, 1) on both $left_i$ and $right_i$.

when ELECTION($id, r, d$) is received on $left_i$ (resp., $right_i$) do
 (2)  case ($id > id_i$) $\wedge$ ($d < 2^r$) then send ELECTION($id, r, d+1$) to $right_i$ (resp., $left_i$)
 (3)       ($id > id_i$) $\wedge$ ($d \geq 2^r$) then send REPLY($id, r$) to $left_i$ (resp., $right_i$)
 (4)       ($id < id_i$)                     then skip
 (5)       ($id = id_i$)                     then send ELECTED($id$) on $left_i$; $elected_i \leftarrow true$
 (6)  end case.

when REPLY($id, r$) is received on $left_i$ (resp., $right_i$) do
 (7)  if ($id \neq id_i$)
 (8)     then send REPLY($id, r$) on $right_i$ (resp., $left_i$)
 (9)     else if (already received REPLY($id, r$) from $right_i$ (resp., $left_i$))
 (10)             then send ELECTION($id_i, r+1, 1$) on both $left_i$ and $right_i$
 (11)          end if
 (12) end if.

when ELECTED($id$) is received on $right_i$ do
 (13) $leader_i \leftarrow id$; $done_i \leftarrow true$;
 (14) if ($id \neq id_i$) then $elected_i \leftarrow false$; send ELECTED($id$) on $left_i$ end if.
```

Figure 6: Leader election algorithm for bidirectional ring network

Secondly, I will turn to an algorithm that I have studied in the sequential context: Bellman-Ford's Shortest Path Algorithm, and try to model its distributed adaptation. The goal is that each vertex in a non-oriented weighted graph learns the shortest path from itself to every other vertex. It is based on the principle of dynamic programming where each vertex p_j stores an array $length_i[1..n]$ such that $length_i[k]$ will contain the length of the shortest path from p_i to p_k (initially, $length_i[i] = 0$ and $length_i[k] = +\infty$ for $k \neq i$). When a node p_i updates its $length_i$ (by comparing the length of the newly learned and the old path and updating if the new path is shorter), it will signal all of its neighbors p_j to update their own $length_j$ array. For a node p_i, $l_{g_i}[j]$ denotes the length associated with the channel (i, j) and $routing\_to_i[1...n]$ is such that $routing\_to_i[k] = j$ means that p_j is a neighbor of p_i on a shortest path to p_k. The algorithm ends when each node acquires the knowledge of the shortest path to other nodes, there will not be anymore UPDATE message as the condition in which $update_i$ becomes $true$ can no longer be satisfied.



```
when START() is received do
(1)  for each $j \in neighbors_i$ do send UPDATE($length_i$) to $p_j$ end for.

when UPDATE($length$) is received from $p_j$ do
(2)  $updated_i \leftarrow false$;
(3)  for each $k \in \{1, \ldots, n\} \setminus \{i\}$ do
(4)      if ($length_i[k] > \ell g_i[j] + length[k]$)
(5)          then $length_i[k] \leftarrow \ell g_i[j] + length[k]$;
(6)               $routing\_to_i[k] \leftarrow j$;
(7)               $updated_i \leftarrow true$
(8)      end if
(9)  end for;
(10) if ($updated_i$)
(11)     then for each $j \in neighbors_i$ do send UPDATE($length_i$) to $p_j$ end for
(12) end if.
```

Figure 5: Distributed adaptation of Bellman-Ford's Shortest Path algorithm

5 Conclusion

For the next step of the project, I will be working with the PNML Framework library implemented in Java, which allows creating models in PNML format required by the MCC. The first model will be the continuation of the Client-Server problem, by developing a generator which puts together pieces of each type of actor in the communication scenario, I can obtain Client-Server models of different scales. Once having mastered the basis of the library, I will begin modeling the above-mentioned algorithms, starting with analyzing them. By looking at the characteristics of these algorithms, I will be able to identify their scaling capabilities (for example here, the amount of nodes/vertices), the interactions of interest between elements of the algorithms and all the possible scenarios. After that, by producing multiples models, I hope to be able to provide the MCC with some interesting ones.

References

C.Girault, R. (2001). *Petri nets for systems engineering.* Springer-Verlag.

Raynal, M. (2012). *Distributed algorithms for message-passing systems.* Springer.