Tarea 2 - Cómputo distribuido

Dalia Camacho, Gabriela Vargas

1 Topología de un problema en los primeros tres estados

Elegimos el problema de cambiar el NIP de una tarjeta bancaria a partir de la banca en línea. Los primeros tres estados son los siguientes.

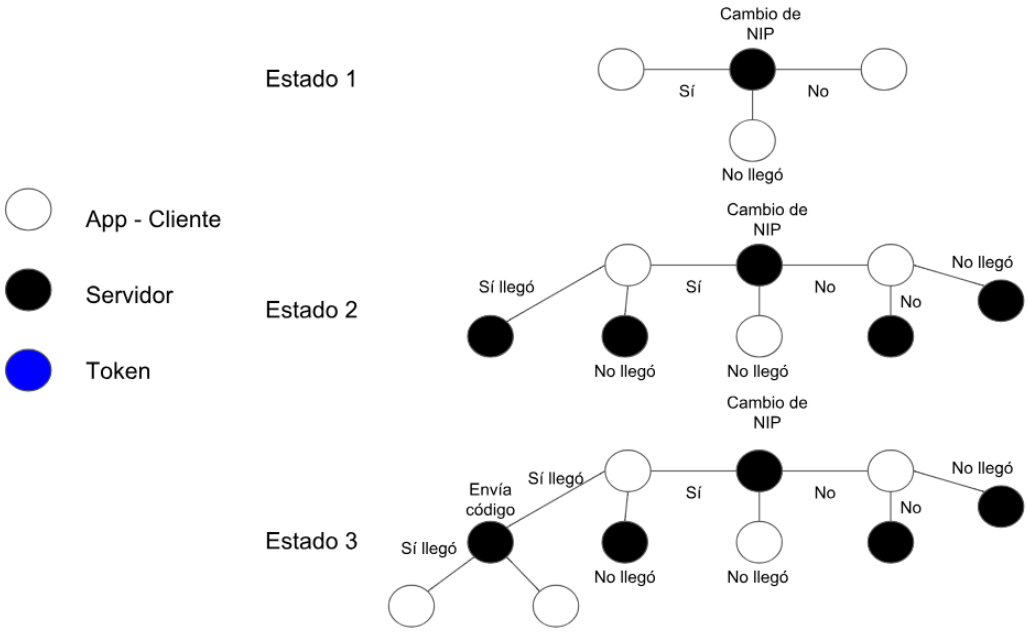

Figure 1: Primeros tres estados posibles

A continuación se muestra todo el proceso.

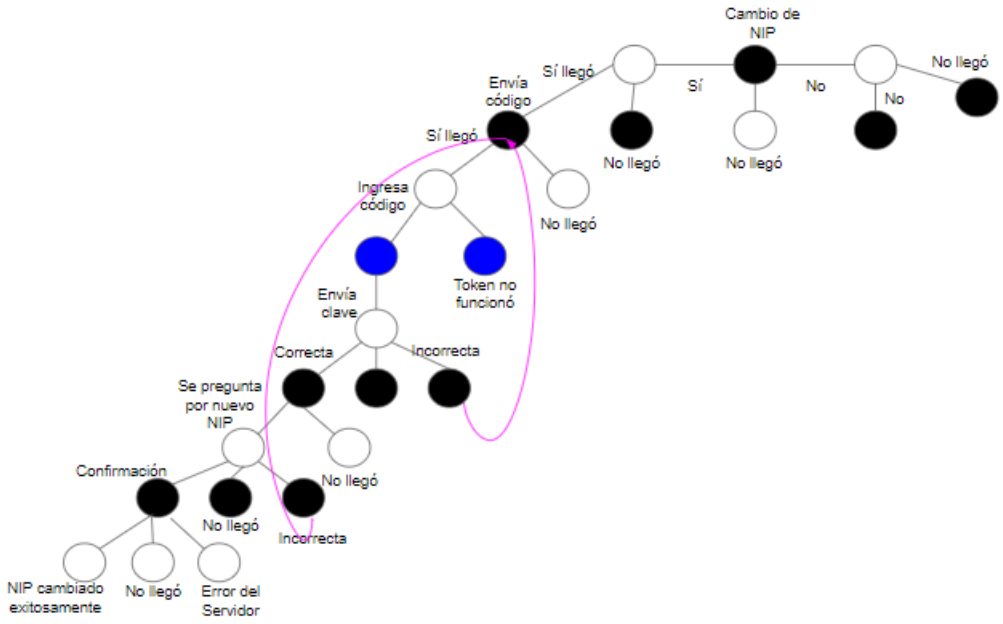

Figure 2: Gráfica completa

2 Problema de consenso

2.1 ¿Qué corresponde a *safety* y a *liveness* en el problema de consenso?[1]

En el problema de consenso *safety* consiste en:

- Sólo se puede elegir un valor que haya sido propuesto.
- Sólo puede elegirse un único valor.
- El proceso garantiza que el valor elegido ha pasado por al menos $n - t$ aceptantes.

Liveness consiste en que se llegue a elegir un valor que haya sido propuesto.

2.2

Considere el siguiente algoritmo para el problema de consenso:

1. Definir $val = X_i$
2. Enviar val a los demás participantes.
3. Esperar hasta recibir $n - t$ mensajes (El sistema tolera t fallas).

4. Generar el vector V, donde la entrada $V[j]$ corresponde al valor del proceso j.

5. Regresar el valor máximo de V.

2.3 ¿En qué casos no sirve el algoritmo?

- Hay más de t fallas.
- Más de t participantes no envíen el mensaje.
- Si uno de los participantes no recibe el valor máximo en los primeros $n-t$ mensajes, éste participante no elegirá el máximo como valor de salida y no se logrará consenso.
- Si el mensaje que contiene el valor máximo no le llega a los demás participantes se podría llegar a un consenso, pero sin haber elegido el verdadero valor.

2.4 ¿Para qué condiciones de entrada C el algoritmo funciona?

- Funciona cuando todas las $X_i = X_j, \forall \;\; i, j$
- Cuando al menos hay $t+1$ X's que comparten el valor del máximo.
- Si hubiera nodo central, que éste conozca el valor máximo desde un inicio.

2.5 ¿Cómo se debe modificar el algoritmo para que funcione? [1]

Proponemos una implementación del algoritmo Paxos, en el que los nodos podrán desempeñar tres papeles:

- Proponente
- Aceptante
- Aprendiz

El algoritmo funciona en dos fases:

1. **Preparación:** Cada proponente selecciona un número k y envía una solicitud de preparación con ese número a los aceptantes. Si el aceptante recibe una solicitud de preparación con un número k mayor que cualquier otra solicitud que haya recibido anteriormente, éste responderá con una promesa de no aceptación a ninguna otra propuesta de número menor que k y con el valor del último número que había aceptado.

2. **Aceptación:** Si el proponente recibe respuesta de $n - t$ aceptantes, entonces envía una propuesta de aceptación con el valor máximo dentro de las respuestas de los aceptantes. Los aceptantes por su parte aceptan la propuesta de aceptación siempre y cuando no hayan recibido una propuesta de preparación con un mayor valor.

El rol de los aprendices es definir en qué momento una propuesta ha sido aceptada por la mayoría de los aceptantes. Para evitar errores por la pérdida de mensajes los aceptantes envían los valores aceptados a aprendices privilegiados quienes a su vez informan a otros aprendices. Esto con la finalidad de tener un sistema más robusto.

3 Resumen de la clase

Vimos el problema de los amantes inseguros en el que dos personas se ponen de acuerdo para verse a través de mensajes. Ellos no saben si el mensaje llegó o no y ninguno va a asistir a la cita si no están seguros que el otro va a ir. En este caso *safety* corresponde a evitar que sólo vaya uno de los dos y *liveness* corresponde a que ambos vayan. No importa el número de mensajes enviados, el último en enviar el mensaje no sabe si éste fue entregado no hay forma en que estén completamente seguros que el otro lo recibió y por lo tanto no están seguros de ir a la cita. Este problema se puede representar mediante una gráfica en cada paso del tiempo donde cada nodo corresponde al conocimiento posible de cada uno de los individuos en cada paso del tiempo. En este problema incrementar el intercambio sólo aumenta el tamaño de la gráfica, pero no la rompe. Esto indica que no hay forma de encontrar una solución al problema.

Además vimos el problema de consenso el cual tiene las siguientes características:

- Un número de participantes n.
- Un valor de entrada X_i para cada participante, que se puede ver como un vector $X = (X_1, X_2, \dots X_n)$.
- Un valor de salida Y_i para cada participante, que se puede ver como un vector $Y = (Y_1, Y_2, \dots Y_n)$.
- *Agreement:* $Y_i = Y_j \quad \forall i, j$
- *Validity:* La decisión corresponde a alguna X_i.
- *Termination:* Se decide por el valor correcto.

Definimos un algoritmo para cada participante con la finalidad de encontrar el valor máximo de los inputs. Este algoritmo no funciona en el caso general.

1. Definir $val = X_i$
2. Enviar val a los demás participantes.

3. Esperar hasta recibir $n - t$ mensajes (El sistema tolera t fallas).

4. Generar el vector V, donde la entrada $V[j]$ corresponde al valor del proceso j.

5. Regresar el valor máximo de V.

References

[1] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.