Sistemas distribuidos

Sistemas Operativos

Departamento de Computación, FCEyN, UBA

May 30, 2024

Introducción - Recordemos...

- En un sistema de cómputo distribuido tenemos varias computadoras autónomas que interactúan entre sí.
- Fortalezas: paralelismo, replicación, descentralización.
- Debilidades:
  - Dificultad para la sincronización.
  - Dificultad para mantener coherencia.
  - No comparten clock.
  - Poseen información parcial.

Ejercicio 1

En una aplicación de homebanking, los usuarios puede comprar cualquier tipo de divisa. El banco cuenta con un sistema que recibe los pedidos (al que llamaremos `backend`), que a su vez se comunica con otro sistema que controla el stock de los diferentes tipos de divisas (que llamaremos `divisas`) y con un tercer sistema que mantiene registro del estado de cuenta de los usuarios (que llamaremos `cuentas`).

- Describir un protocolo que permita a los usuarios comprar divisas manteniendo en todo momento la consistencia. Suponer:
  - que cada usuario puede realizar una única operación a la vez,
  - que los mensajes se pueden perder,
  - que `backend` no falla y,
  - que si `divisas` o `cuentas` fallan, eventualmente se recuperan al estado anterior a la falla (cada uno por separado).
- Explique brevemente qué tendría que hacer uno de los sistemas intervinientes en caso de una falla.

Resolución

Un problema que podemos tener al diseñar el protocolo es que uno de los sistemas acepte la operación (y empiece a realizarla) y el otro no. De esta manera, el estado general es inconsistente (porque uno de los sistemas habría computado la compra como exitosa y el otro no).

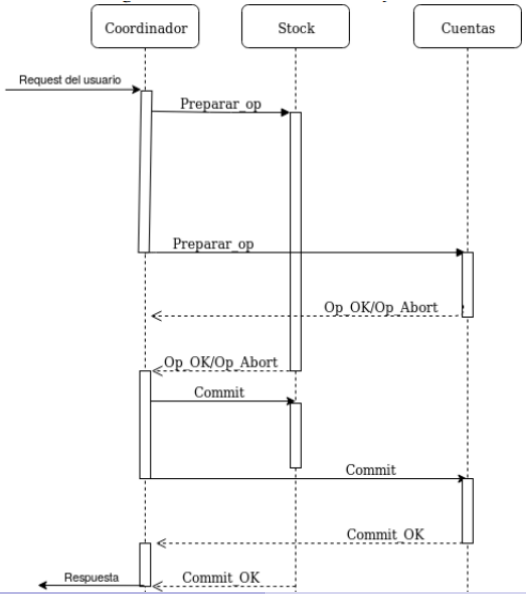
- ¿Podemos resolverlo con timestamps?
- Lo que necesitamos es *locking distribuido*.

Recordemos *Two Phase Commit*. Tenemos procesos que quieren coordinar un commit. Supongamos que tenemos un nodo líder. Como el nombre del algoritmo dice, tenemos dos fases.

- Fase *commit request*: En esta fase se pregunta si se puede realizar la operación pedida.
  - Paso 1: El líder le envía a todos los nodos un mensaje de `COMMIT_REQUEST` preguntándoles si pueden realizar la operación pedida.
  - Paso 2: Los nodos receptores reciben el mensaje y proceden a "ejecutar" lo pedido, bloqueando los recursos que tienen que utilizar. En realidad, no lo hacen commit de los cambios, sino que guardan en una tabla cómo deshacer los cambios hechos.
  - Paso 3: Cada nodo contesta con un mensaje `OK` en caso de éxito en su operación o `ABORT` si hubo algún fallo.

- Fase *commit* (éxito):
  - Paso 1: En caso de que todos los nodos hayan contestado con `OK`, el líder envía a todos los nodos el mensaje de `COMMIT`.
  - Paso 2: Cada nodo receptor completa su operación, liberando los recursos y el lock.
  - Paso 3: Cada nodo envía `COMMIT_OK` al líder.
  - Paso 4: El líder da por finalizado el proceso en caso de recibir `COMMIT_OK` de todos los nodos.
- Fase *commit* (falla):
  - Paso 1: En caso que al menos un nodo conteste con `ABORT`, se procede a enviar un mensaje de `ROLLBACK` a todos los nodos.
  - Paso 2: Cada nodo receptor procede a deshacer su operación, usando la tabla que armó en la fase anterior, luego liberando los recursos y desbloqueando.
  - Paso 3: Cada nodo envía `COMMIT_OK` al líder.
  - Paso 4: El líder da por finalizado el procesamiento de ese pedido comunicando que falló.

Volviendo al ejercicio



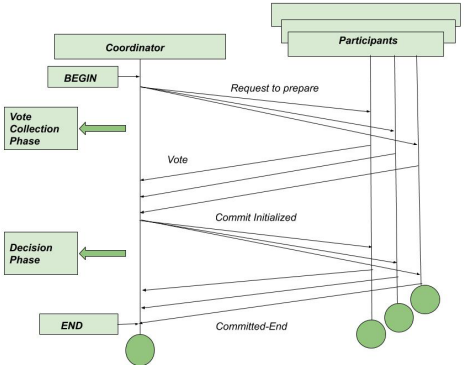
¿Qué pasa en caso de error?

- ¿Qué pasa si otro usuario envía otra transacción a `backend`?
- El ejercicio presupone que `backend` no falla, ¿qué pasa si falla?
- Notemos que si falla luego de la primera fase, `divisas` y `cuentas` quedan bloqueados hasta que `backend` se recupere.

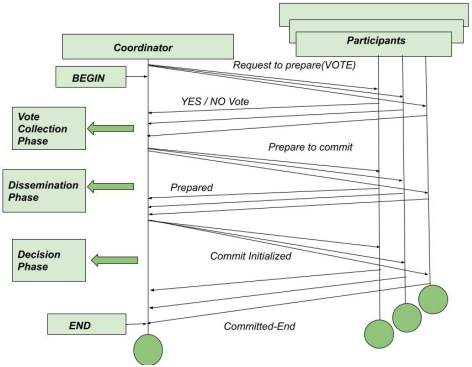
3PC

Igual que 2PC, pero se agrega una nueva fase entre las dos fases del algoritmo anterior

- Fase 1: igual que 2PC.
- Fase 2
  - Proceso líder junta todas las respuestas que recibió y si hay algún `ABORT` envía `ABORT` a todos los nodos y da por terminado el procesamiento de ese pedido.
  - Si no, manda un mensaje para que los otros procesos se pongan en `PREPARE TO COMMIT`.
  - Los nodos contestan si están en condiciones de commitear o no.
- Fase 3 es igual que la fase final de 2PC.



(a) 2 Phase Commit



(b) 3 Phase Commit

3PC - Caso muerte líder

- Si el líder muere en la primera fase o segunda fase
  - Se elije un nuevo líder (de alguna forma) y se puede abortar la ejecución o reiniciar al algoritmo sin el líder anterior.
  - Debe pedirle a los nodos que reenvíen su última respuesta.
  - Si alguien decidió ABORT entonces el nuevo líder decide ABORT y le manda este mensaje a todos.
- Si el líder muere en la última fase
  - Se elije un nuevo líder (de alguna forma) y éste le pregunta a los otros nodos en qué estado están. Si al menos un nodo dice que está en el estado de "commit iniciado", entonces se supone que el líder anterior había decidido pasar a la última fase, con lo cual se puede continuar el protocolo sin abortar.

Elección del líder

- Ahora bien, ¿cómo se puede elegir al líder?
- El algoritmo de elección va a depender también de la topología de nuestro sistema.
- Vamos a ver tres algoritmos muy simples que se pueden usar para elegir el líder.

Ejercicio 2

Se tiene una red en anillo donde cada nodo i puede comunicarse sólo con el nodo siguiente. Cada día los nodos de la red se prenden, generan un "número mágico" ($\texttt{NM}_i$) y se ponen a computar una función muy difícil. El resultado del cómputo de todos los nodos se debe guardar en el nodo que ese día haya sacado el número mágico más grande, y sólo deben empezar a computar una vez que todos los nodos sepan quién es el que tiene el número mágico más grande.
Este "número mágico" puede ser azaroso o una abstracción de algún criterio que sea pertinente al problema en cuestión.

- Diseñe un protocolo para cumplir dicha tarea.

Escenario 2 - Elección de líder

Solución informal, en 2 etapas (basada en LCR: LeLann-Chang-Roberts, 1979):

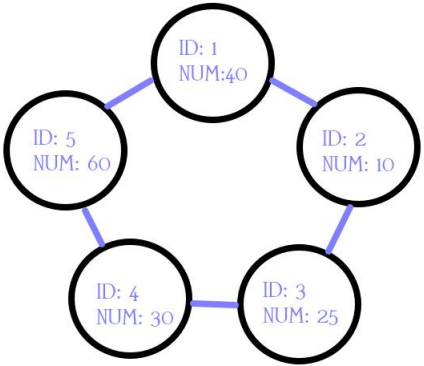
- Etapa 1:
  - Cada proceso i envía $(i, \texttt{NM}_i)$ al proceso $i + 1$.
  - Cuando un proceso i recibe un par $(j, \texttt{NM}_j)$, compara $\texttt{NM}_j$ con $\texttt{NM}_i$.
  - Si $\texttt{NM}_j > \texttt{NM}_i$, reenvía $(j, \texttt{NM}_j)$.
  - Si no, reenvía $(i, \texttt{NM}_i)$.

Escenario 2 - Elección de líder

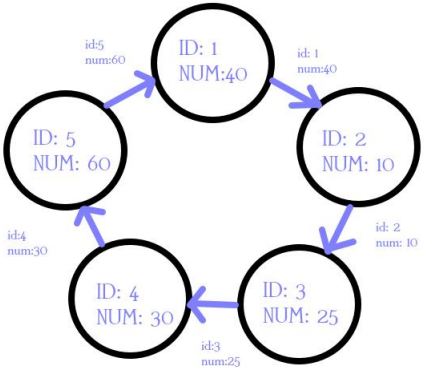
- Etapa 2:
  - Una vez que i recibe $(i, \texttt{NM}_i)$, se auto declara líder.
  - El nodo líder envía un nuevo mensaje declarándose líder a través del anillo.
  - Cada nodo no líder recibe el mensaje que indica quién es el líder y lo reenvía.
  - Una vez que al nodo líder le llega su mensaje comunicándole que él mismo es el líder, termina el procedimiento.

Veamos un ejemplo:

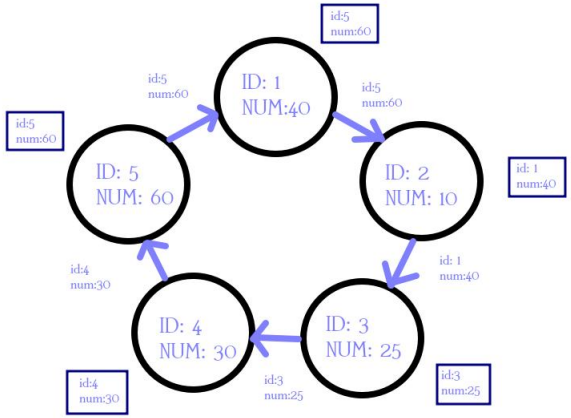
Ejemplo: tenemos un sistema con la siguiente estructura y vamos a correr el algoritmo.



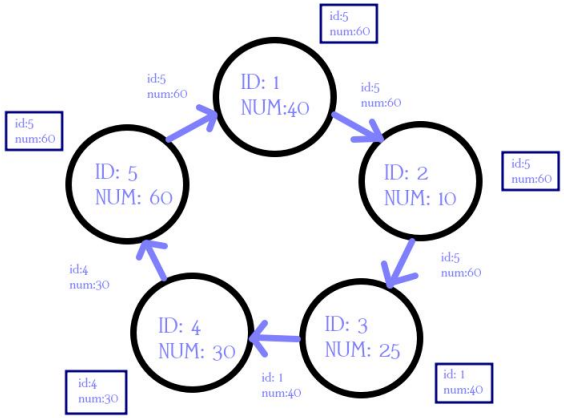
Al principio, cada nodo envía su identificador y número mágico.



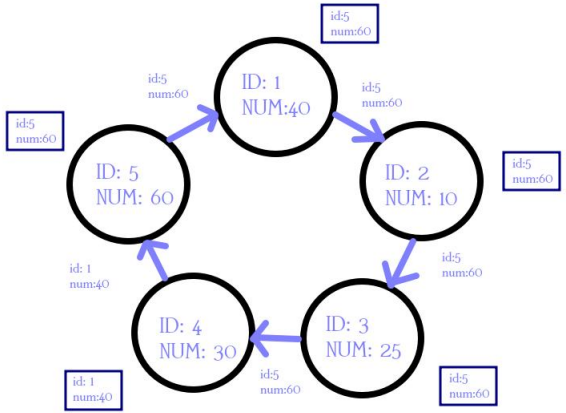
Cada nodo compara su número mágico con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvía. Caso contrario, envía el suyo.



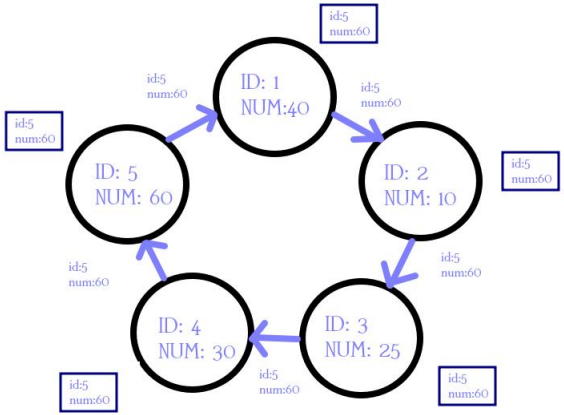
Cada nodo compara su número mágico con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvía. Caso contrario, envía el suyo.



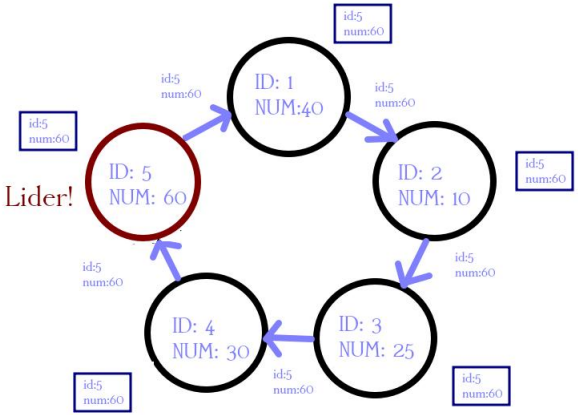
Cada nodo compara su número mágico con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvía. Caso contrario, envía el suyo.



Cada nodo compara su número mágico con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvía. Caso contrario, envía el suyo.



Cada nodo compara su número mágico con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvía. Caso contrario, envía el suyo.



¿Qué prerrequisitos tiene este algoritmo?

- Topología en anillo.
- Nadie se cae.
- Mensajes no se pierden definitivamente.

Veamos alternativas que no tienen esos prerrequisitos (tiene otros).

Algoritmo FloodMax:

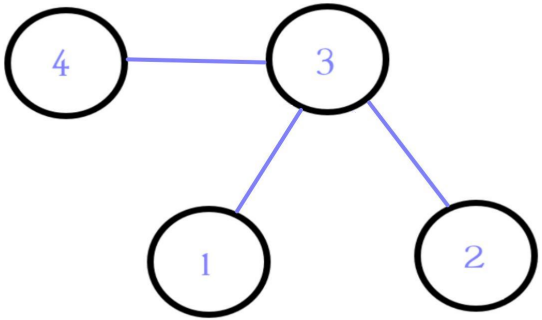
- No necesita que todos los nodos se conecten con todos pero sí necesitamos que siempre exista un camino entre dos nodos distintos.
- Desventaja: requiere saber el diámetro del grafo. Es decir, necesitamos saber la longitud del camino más largo. Notamos ese diámetro con n.
- Ventaja: garantiza que en n pasos se determina un líder.

FloodMax - Algoritmo

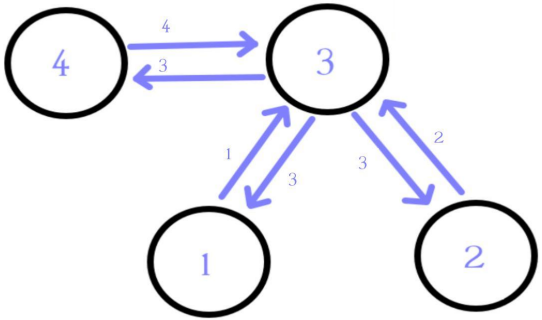
- Cada nodo mantiene como estado el máximo ID que vio circular. Inicialmente el máximo es el propio ID.
- En cada iteración envía este máximo a todos los nodos con los que está conectado.
- Cada vez que recibe un mensaje, actualiza el máximo y lo reenvía.
- Luego de n iteraciones el algoritmo termina y el máximo ID que se propagó por la red es el líder.

Veamos un ejemplo.

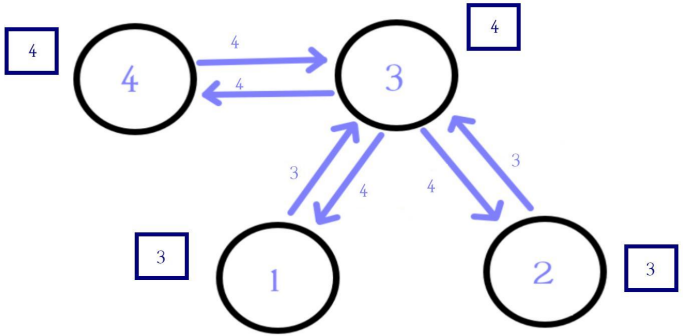
Ejemplo: Tenemos un sistema con la siguiente estructura y vamos a correr *FloodMax*. El diámetro es 2 (distancia de 4 a 2 o de 4 a 1).



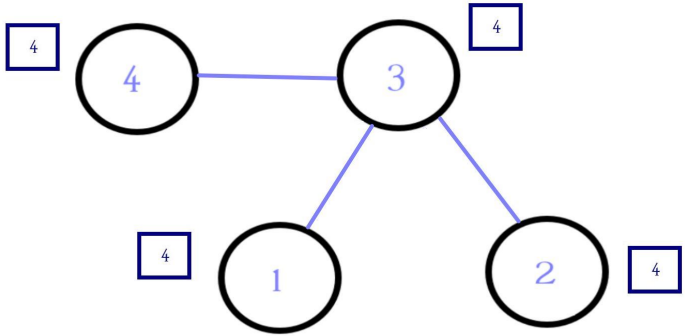
Primera iteración: cada nodo propaga su ID.



Segunda iteración: cada nodo actualizó su máximo y ahora vuelve a propagar el mismo.



Terminamos ($n = 2$) y todos los nodos saben cuál es el líder.



Preguntas

- ¿Todos los intercambios tienen que darse a la vez?
- ¿Cuántos mensajes en total vamos a mandar?
- O(cantidad de ejes * diámetro).
- ¿Podemos disminuir la cantidad de mensajes?

Algoritmo *bully*

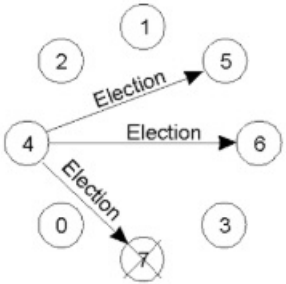
Prerrequisito: A diferencia de FloodMax cada nodo del sistema está conectado con todos los otros. Tenemos alta conectividad a cambio de menos rondas en total (en el caso de no falla).

- Un nodo i va a empezar la elección mandando `elección` a todos los nodos $j > i$.
- Cada proceso que recibe `elección` responde con `OK` y comienza él mismo una elección como en el paso anterior.
- Si recibe algún `OK` sabe que hay alguien con ID mayor y espera a enterarse quién mediante el mensaje `soy el líder`.
- Si el nodo i no recibe ningún `OK` se declara líder (porque nadie consideró tener un ID $> i$) y *broadcastea* el mensaje `soy el líder`.

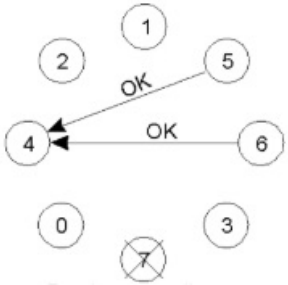
Veamos un ejemplo

Algoritmo *bully*

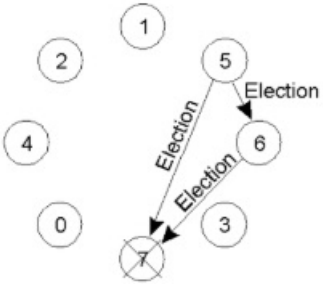
Tenemos la siguiente distribución. El nodo 4 quiere ser el líder y manda `elección` a todos los nodos con id mayor. El nodo 7 está muerto, pero 4 no lo sabe.



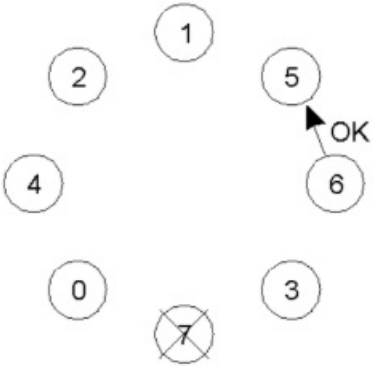
5 y 6 le responden que están vivos (`OK`) y por ende 4 sabe que perdió.



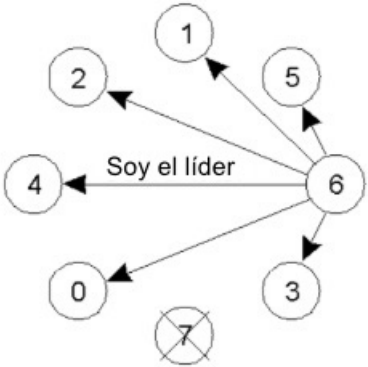
5 y 6 comienzan ellos una elección de líder.



6 le avisa a 5 que está vivo (mensaje `OK`). 5 sabe que perdió.



6 le avisa a todos que es el líder.



- ¿Cuánto hay que esperar?
- De alguna forma es un algoritmo sincrónico.
-
- ¿Qué hubiera pasado si 7 revivía en alguno de los pasos?
- ¿Cuántos mensajes se mandan aproximadamente?
- $O(n^2)$ con n la cantidad de nodos.

¿Preguntas?

Bibliografía

- "Communicating Sequential Processes", C. A. R. Hoare, Prentice Hall, 1985. http://www.usingcsp.com/
- "Parallel Program Design - A Foundation", K. Chandy y J. Misra, Addison-Wesley, 1988.
- L. Lamport. Time, clocks, and the ordering of events in a distributed system. CACM 21:7 1978. http://goo.gl/ENh2f7
- L. Lamport, R.Shostak, M.Pease. The Bizantine Generals problem. ACM TOPLAS 4:3, 1982. http://goo.gl/DY0Qis
- Ernest Chang; Rosemary Roberts (1979), "An improved algorithm for decentralized extrema-finding in circular configurations of processes", Communications of the ACM, 22 (5), ACM: 281–283, doi:10.1145/359104.359108