Sistemas distribuidos

Sistemas Operativos

Departamento de Computación, FCEyN, UBA

June 1, 2023

Estructura de la clase

Introducción

Algoritmos de consenso

- 2PC
- 3PC

Elección del líder

- LCR
- Flood Max
- Bully Algorithm

Introducción - Recordemos...

- En un sistema de cómputo distribuido tenemos varias computadoras autónomas que interactuan entre si a las cuales abstraemos como un unico sistema a la vista del usuario.
- Fortalezas: Paralelismo, Replicación, Descentralización.
- Debilidades:
  - Dificultad para la sincronización.
  - Dificultad para mantener coherencia.
  - No suelen compartir clock.
  - Información parcial.

Ejercicio 1

Una aplicación de homebanking permite realizar operaciones bancarias de manera remota. Entre otras cosas, comprar y vender diferentes tipos de divisas. Un usuario puede comprar cualquier tipo de moneda dentro del sistema con cualquiera que posea en su cuenta. El banco cuenta con un sistema que recibe los pedidos, que a su vez se puede comunicar con otro sistema que controla el stock de los diferentes tipos de divisas y un tercer sistema que mantiene registro del estado de cuenta de los usuarios.

- Describir un protocolo que permita a los usuarios comprar divisas manteniendo en todo momento la consistencia. Asuma que cada usuario solo puede mantener una conexión en simultáneo (es decir, una misma persona no puede realizar una operación mientras otra se está llevando a cabo). Los mensajes se pueden perden y el sistema que recibe los pedidos no falla y, si el resto falla, eventualmente se recupera al estado anterior a la falla.
- Explique brevemente que tendría que hacer uno de los sistemas intervinientes en caso de una falla.

Resolución

Un problema que podemos tener al realizar 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).

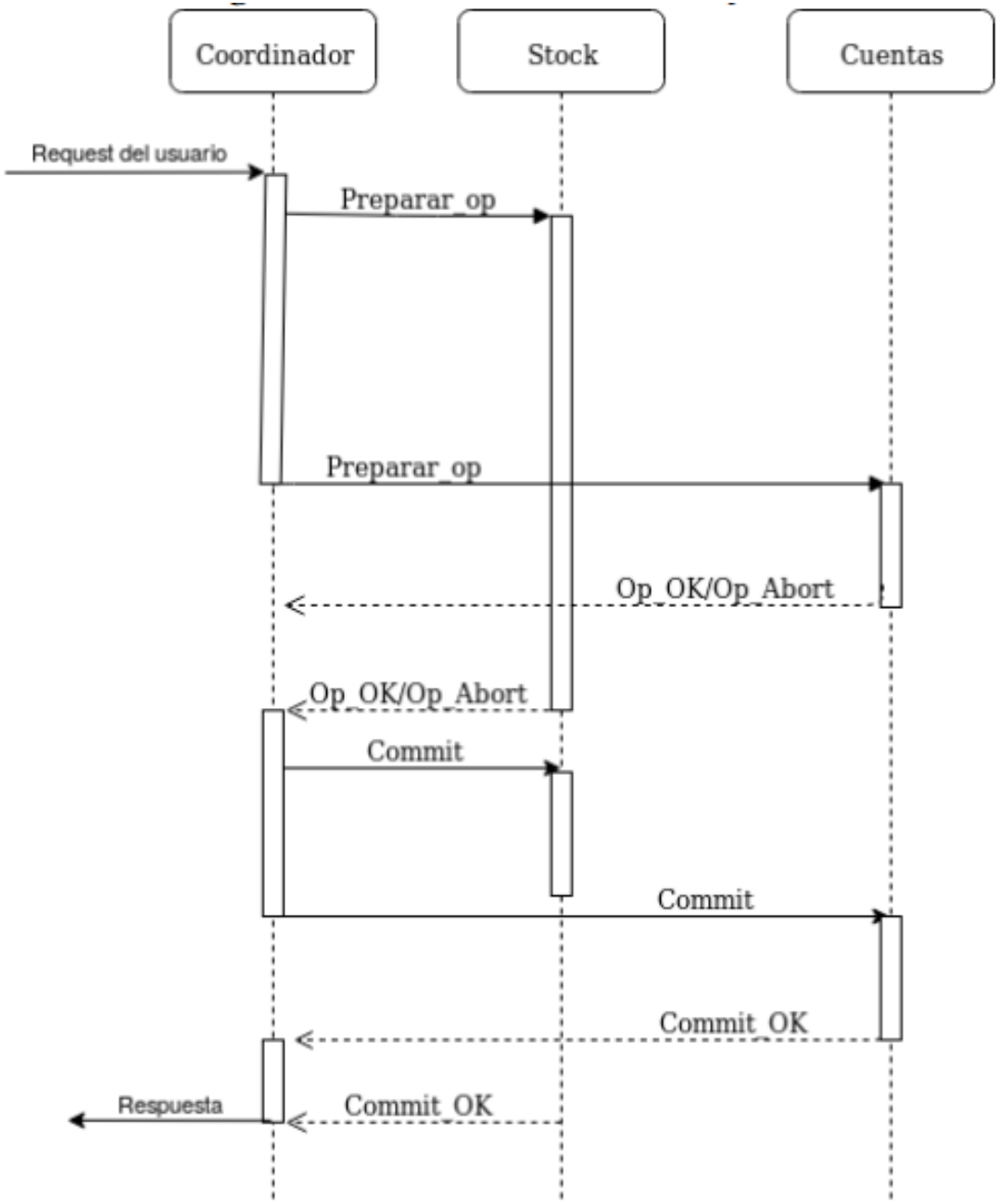
2PC

Notar que lo que necesitamos es un *locking distribuído*
Recordemos Two Phase Commit. Tenemos procesos que quieren coordinar un commit. Supongamos que tenemos un nodo lider. Como el nombre del algoritmo dice, tenemos dos fases.

- Fase Commit Request: En esta fase se pregunta si se puede realizar la operacion pedida
  - Paso 1: El lider le envia a todos los nodos un mensaje con el commit correspondiente y les pregunta si se puede realizar la operación pedida.
  - Paso 2: Los nodos receptores reciben el mensaje y proceden a ejecutar lo pedido, lockeando los recursos que tienen que utilizar. Sin embargo, no hacen "commit", sino que guardan en una tabla el proceso de como deshacer los cambios hechos.
  - Paso 3: Cada nodo contesta con un mensaje "OK" en caso de exito en su operacion o "ABORT" si hubo algún fallo.

- Fase de completacion (exito):
  - Paso 1: En caso que todos los nodos contestaron con "OK", entonces el lider envia a todos los nodos el mensaje de "commit".
  - Paso 2: Cada nodo receptor completa su operacion, liberando los recursos y el lock.
  - Paso 3: Cada nodo envia "COMMIT_OK" al lider.
  - Paso 4: El lider da por finalizado el proceso en caso de recibir "COMMIT" de todos los nodos.
- Fase de completacion (Fallo):
  - Paso 1: En caso que al menos un nodo contestara con "ABORT", se procede a enviar un mensaje de "rollback" a todos los nodos
  - Paso 2: Cada nodo receptor procede a deshacer su operacion, usando la tabla que uso en la fase anterior, luego liberando los recursos y unlockeando.
  - Paso 3: Cada nodo envia "COMMIT_OK" al lider.
  - Paso 4: El lider da por finalizado el proceso comunicando el fallo.

Volviendo al ejercicio



¿Que pasa en caso de error de algun nodo o un usuario diferente llegue?

- Cómo el enunciado sólo nos pide mantener la consistencia para el usuario, en caso de que una operación de un usuario diferente llegue, simplemente podemos demorarla hasta las transacciones anteriores sean realizadas.
- Bajo las suposiciones del enunciado, un mensaje puede no llegar nunca o los nodos pueden tener que reiniciarse volviendo al estado anterior a la falla. Debido a que el protocolo se basa en una secuencia ordenada de mensajes entre el coordinador y los otros nodos y los mensajes nunca se repiten (y por lo tanto, si llega un mensaje "repetido" a un nodo, no genera ambigüedades en el protocolo), basta con reenviar el último mensajes luego de un tiempo suficientemente grande (que el enunciado aclara que conocemos) en caso de no recibir respuesta.

¿Que pasa en caso de error de algun nodo o un usuario diferente llegue?

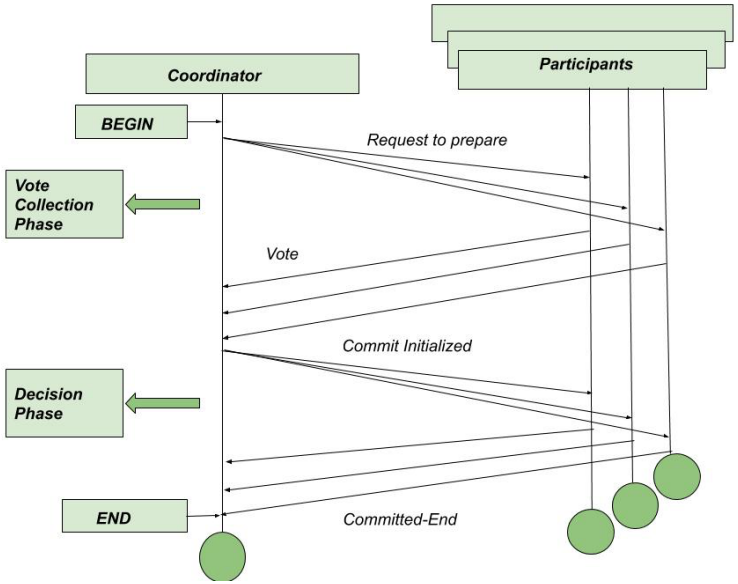
- Sin embargo, en el ejercicio nos pide una asumción clave, que el nodo de los pedidos no falle.
- ¿Que pasa ahora si el lider falla?
- Notemos que si falla luego de la primera fase, los nodos deberán esperar (bloqueando los recursos tomados) hasta que éste se recupere.

3PC

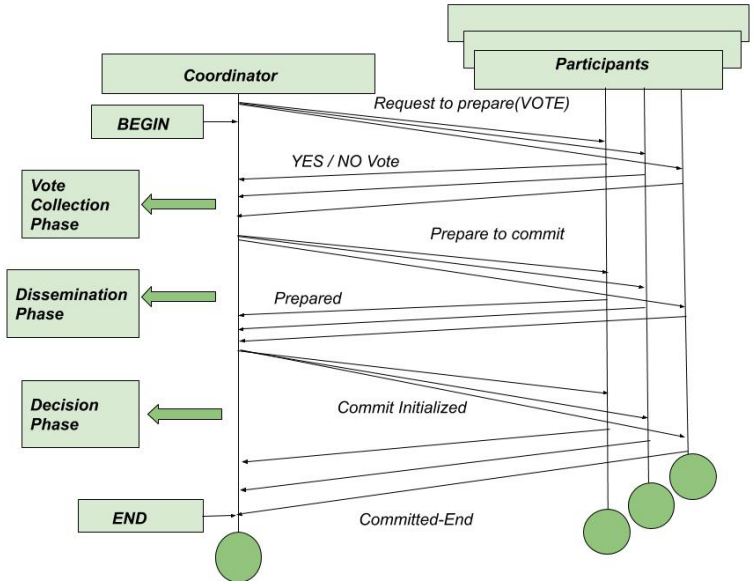
Igual que 2PC, pero se agrega una nueva fase entre las dos fases del algoritmo anterior

- Paso 1: igual que 2PC
- Paso 2
  - Proceso lider junta todos los valores que recibió, si hay algun "ABORT" decide enviar a todos el mensaje de aborto y terminar el proceso.
  - Si no manda un mensaje para que los otros procesos se pongan en 'Prepare to commit'.
  - Los nodos contestan si estan preparados para commitear o no
  - Paso 3 es igual a 2PC

3PC



(a) 2 Phase Commit



(b) 3 Phase Commit

3PC - Caso muerte lider

- Si el lider muere en la primera fase o segunda fase
  - Se elije un nuevo lider (de alguna forma) y se puede abortar la ejecucion o reiniciar al algoritmo sin el lider anterior.
  - No necesariamente tiene que tener respuesta de todos los procesos (algunos más pueden haber fallado).
  - Si alguien decidió ABORT entonces el nuevo lider decide ABORT y le manda este mensaje a todos
- Si el lider muere en la ultima fase
  - Se elije un nuevo lider (de alguna forma) y le pregunta a los otros nodos en que estado estan. Si un solo nodo dice que esta en el estado de "commit inicialized", entonces se asume que el lider anterior habia decidio pasar a la ultima fase, con lo cual se puede continuar el protocolo sin abortar

Elección del líder

- Muchos algoritmos distribuidos necesitan de un líder para hacer de arbitro
- 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 puede comunicarse sólo con el nodo siguiente o el anterior. Cada día los nodos de la red se prenden, generan un número mágico 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.

- Diseñe un protocolo para cumplir dicha tarea.

Escenario 2 - Elección de lider

Solucion informal, en 2 etapas:

- Etapa 1:
  - Cada proceso envia su identificación y su numero a través del anillo.
  - Cuando un proceso recibe una identificación y un numero, compara el numero con el suyo
  - Si el numero que llegó es mayor al suyo, lo reenvia.
  - Si es menor, envia su identificación y su numero.

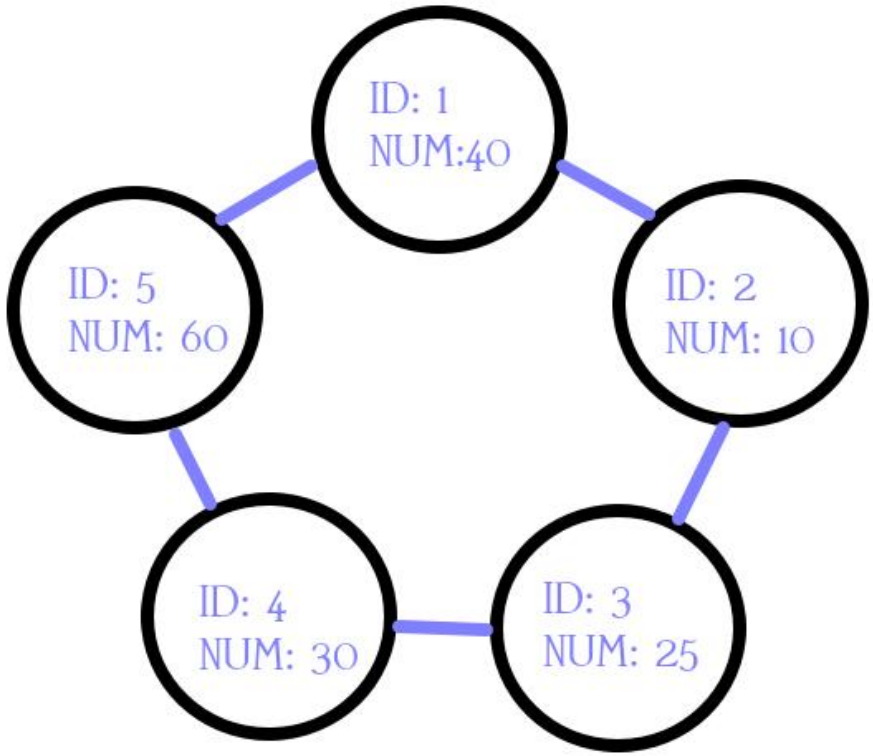
Escenario 2 - Elección de lider

- Etapa 2:
  - Una vez a un nodo le llega su misma identificación y número, este se declara lider.
  - El nodo lider envia un nuevo mensaje declarandose lider a travez del anillo.
  - Cada nodo no lider recibe el mensaje de quien es el lider y lo reenvia.
  - Una vez que al nodo lider le llega su mensaje diciendo que es el lider, termina el procedimiento.

Veamos un ejemplo:

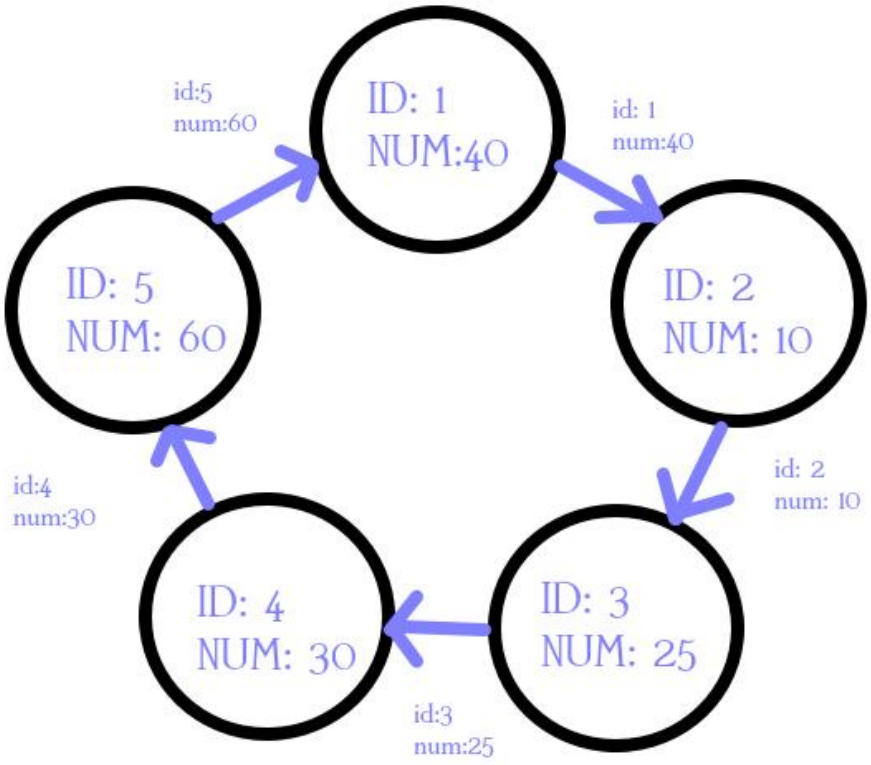
LCR

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



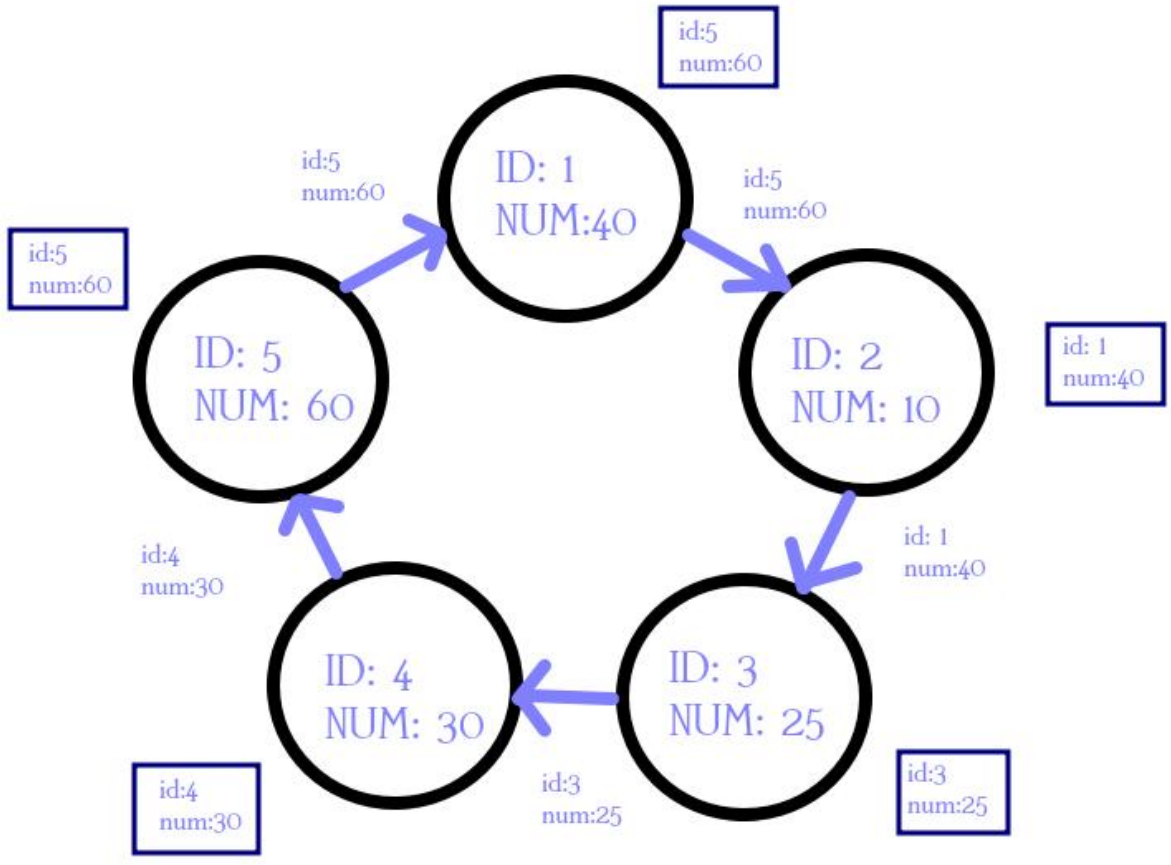
LCR

Al principio, cada nodo envia su identificador y numero.



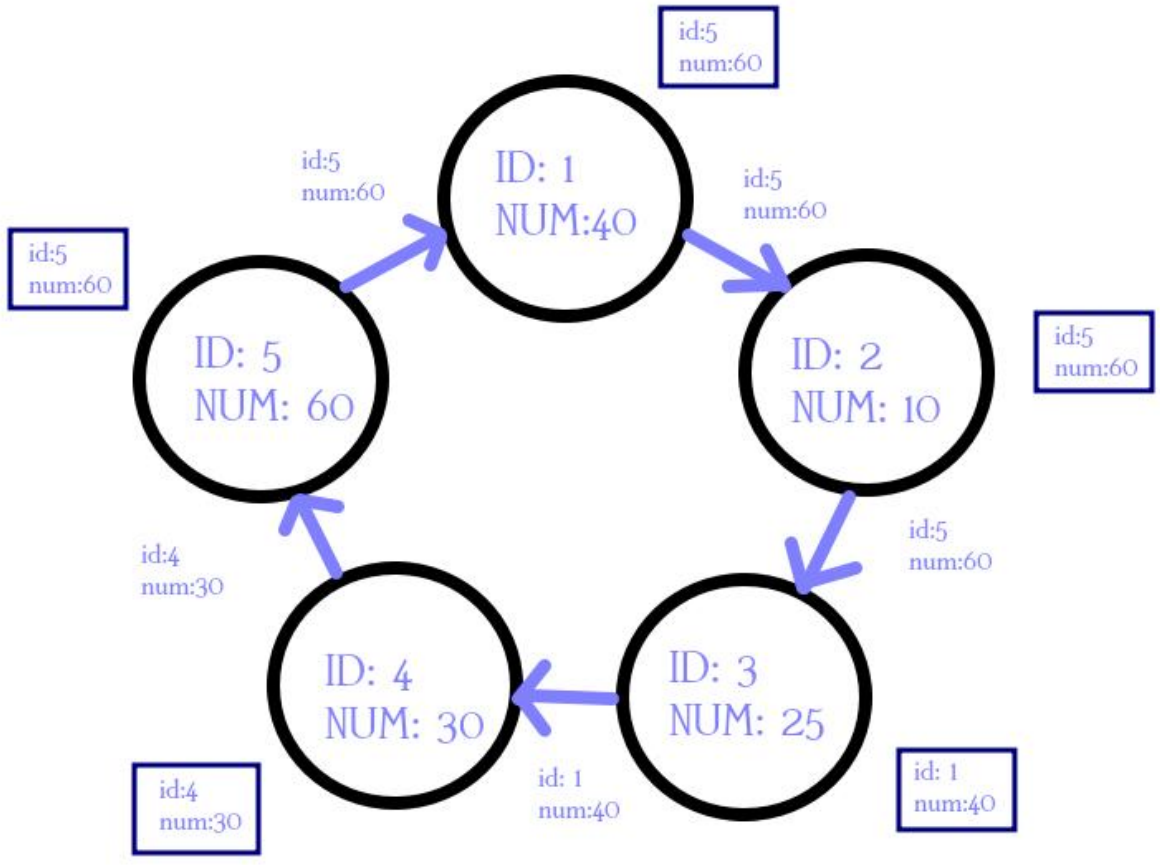
LCR

Cada nodo compara su numero con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvia. Caso contrario, envia el suyo.



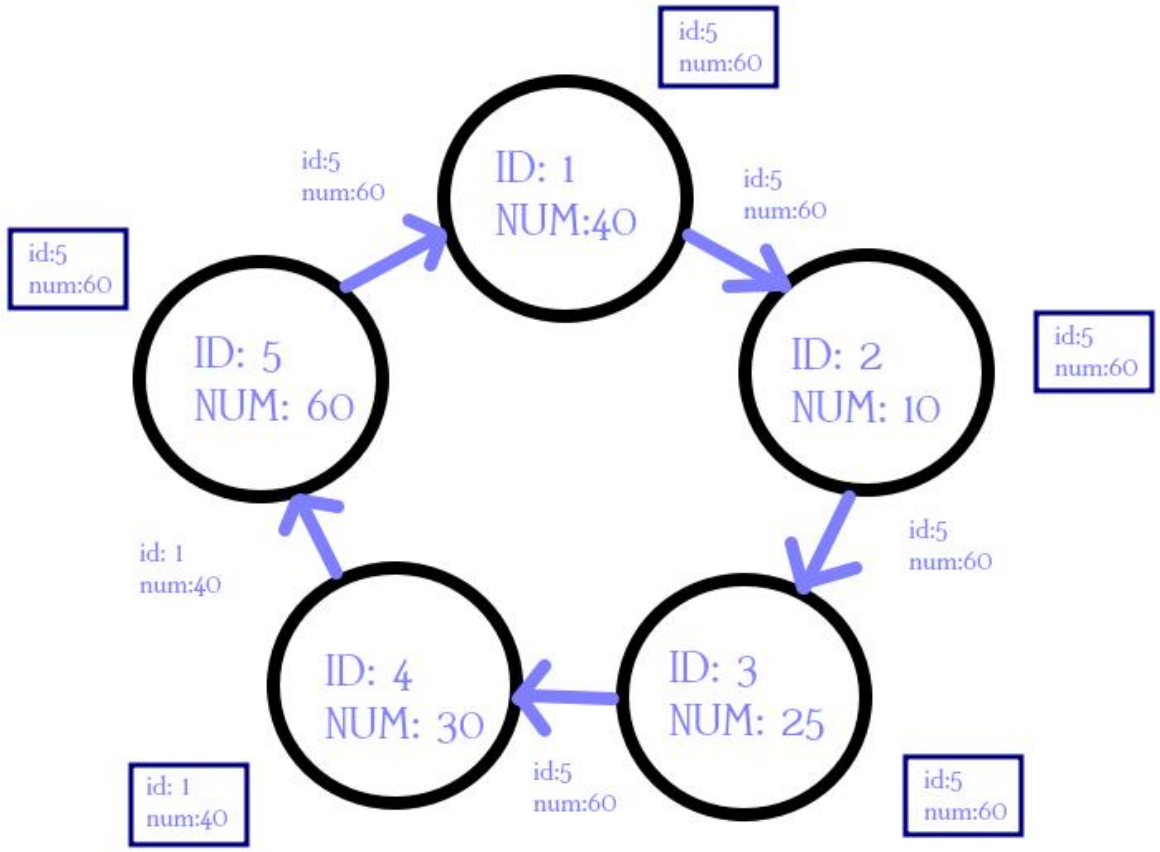
LCR

Cada nodo compara su numero con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvia. Caso contrario, envia el suyo.



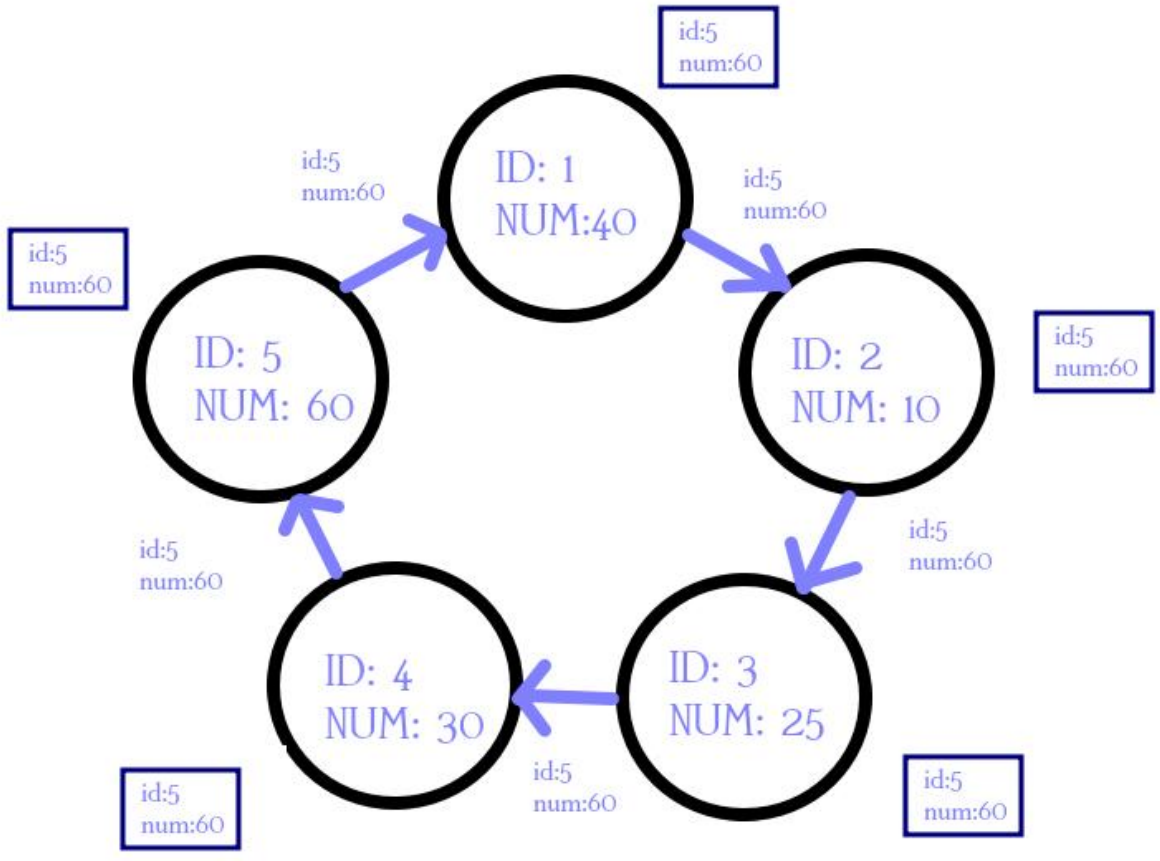
LCR

Cada nodo compara su numero con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvia. Caso contrario, envia el suyo.



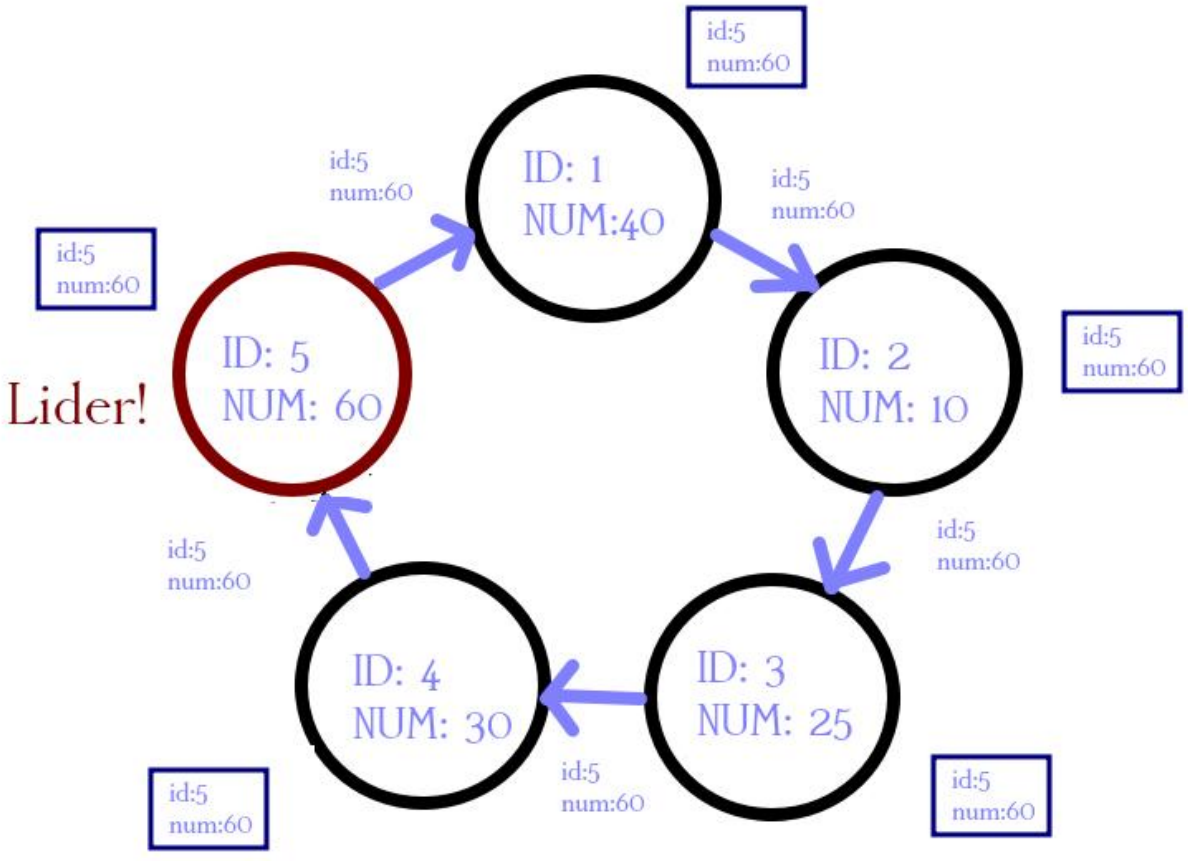
LCR

Cada nodo compara su numero con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvia. Caso contrario, envia el suyo.



LCR

Cada nodo compara su numero con el mensaje que llega. En caso que sea mayor lo recibido, lo reenvia. Caso contrario, envia el suyo.



Flood max

- No necesitamos que todos los nodos se conecten con todos pero si necesitamos que siempre exista un camino entre dos nodos distintos.
- Una desventaja es que necesitamos saber el diámetro del grafo que se forma. Es decir necesitamos saber la longitud del camino más largo. Notamos el diámero con n
- Nos garantiza que en n pasos conseguimos un líder

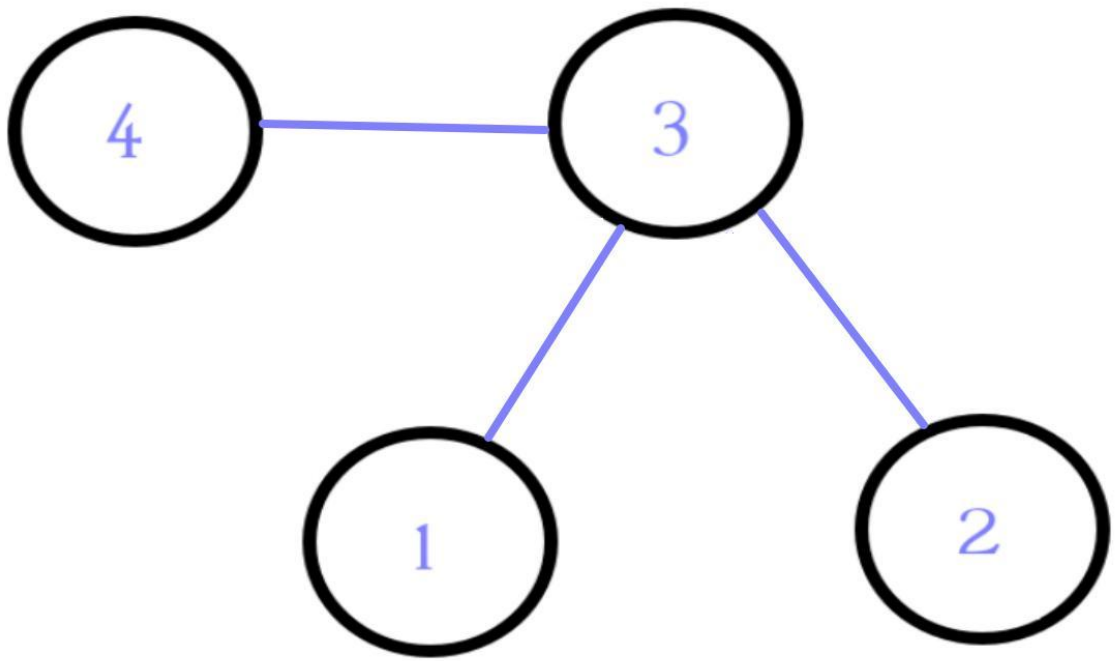
Flood max - Algoritmo

- Cada nodo manda el máximo valor que vio a todos los nodos con los que esta conectado. En la primer ronda envía su ID porque es el único que vio hasta ahora.
- Actualiza el máximo con los resultados que le llegan
- Luego de n rondas el algoritmo termina y el máximo ID que se propago por la red es el líder

Veamos un ejemplo.

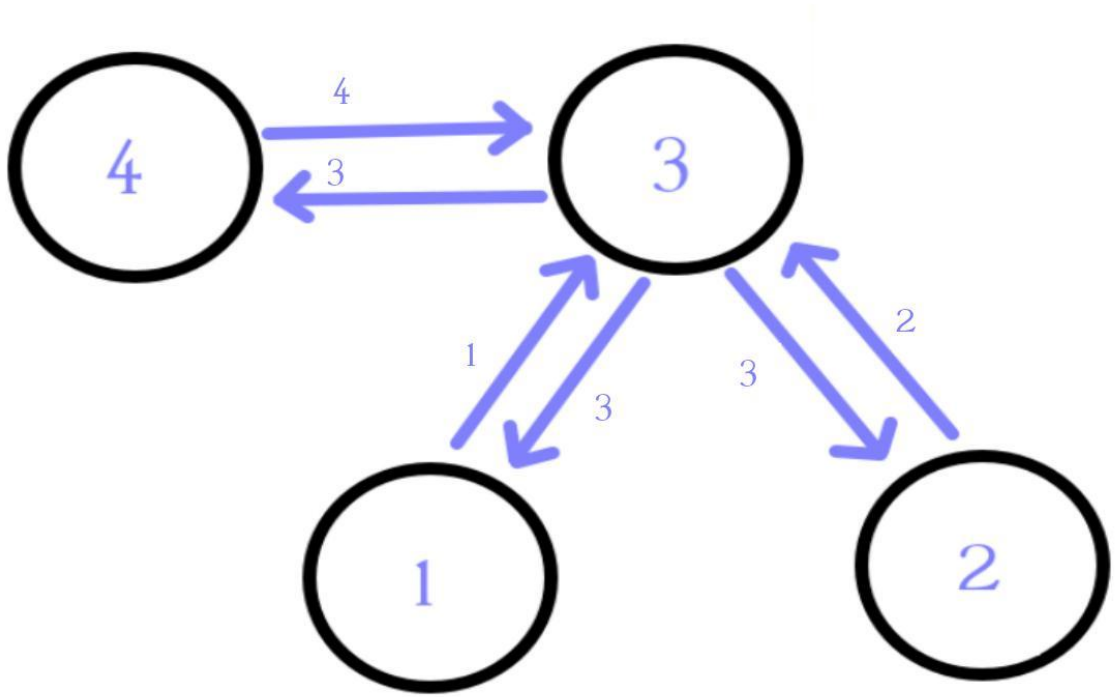
Flood max

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).



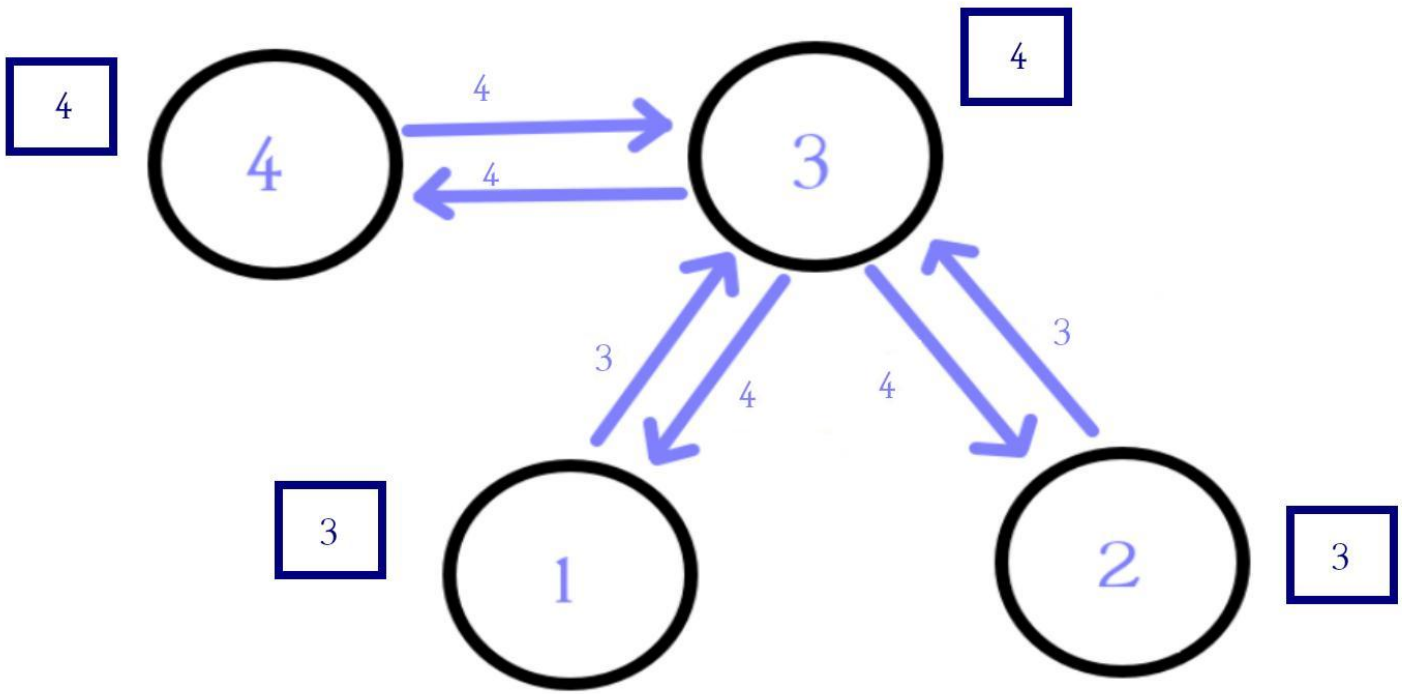
Flood max

Primera ronda: cada nodo propaga su ID por que es el único que vio hasta ahora.



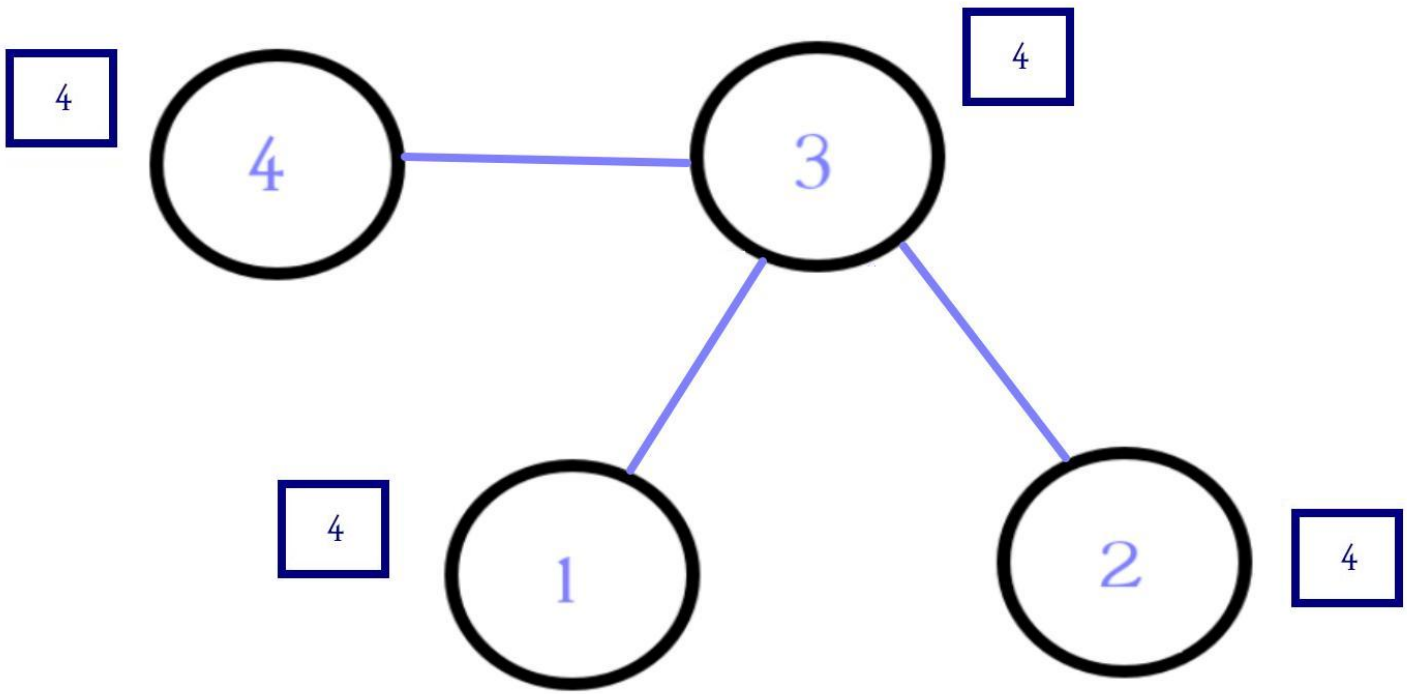
Flood max

Segunda ronda: cada nodo actualizo su máximo y ahora vuelve a propagar el mismo.



Flood max

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



Preguntas

- ¿Cuántos mensajes en total vamos a mandar?
- O(cantidad de ejes * diámetro).
- ¿Cómo se inicia la búsqueda del líder?

Bully Algorithm

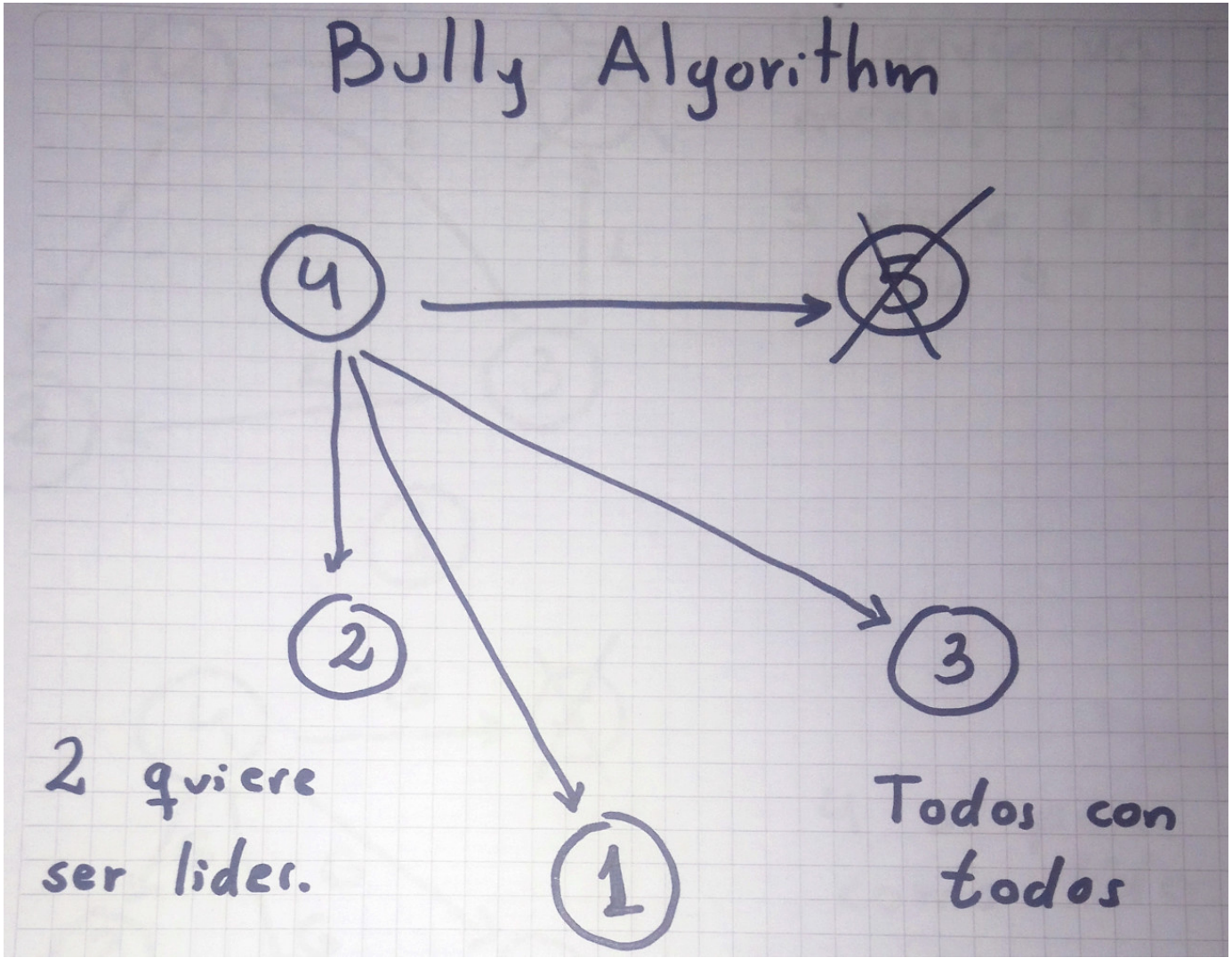
Prerequisito: A diferencia de flood max cada nodo del sistema esta conectado con todos los otros. Tenemos alta conectividad a cambio de menos rondas en total (en el caso de no falla).

- Un nodo va a empezar la elección mandando un mensaje a todos los otros nodos con ID mayor a el diciendo que quiere ser el líder.
- Si nadie le contesta se declara el líder.
- Cada nodo al que le llega el mensaje va a repetir el mismo esquema, pero también contestandole al nodo que le hablo que el se va a encargar de ser el líder.
- Si nadie le contesta se declara el líder y broadcastea un mensaje a toda la red declarandose victorioso.
- ¿Qué pasa si un nodo muere y luego revive?

Veamos un ejemplo

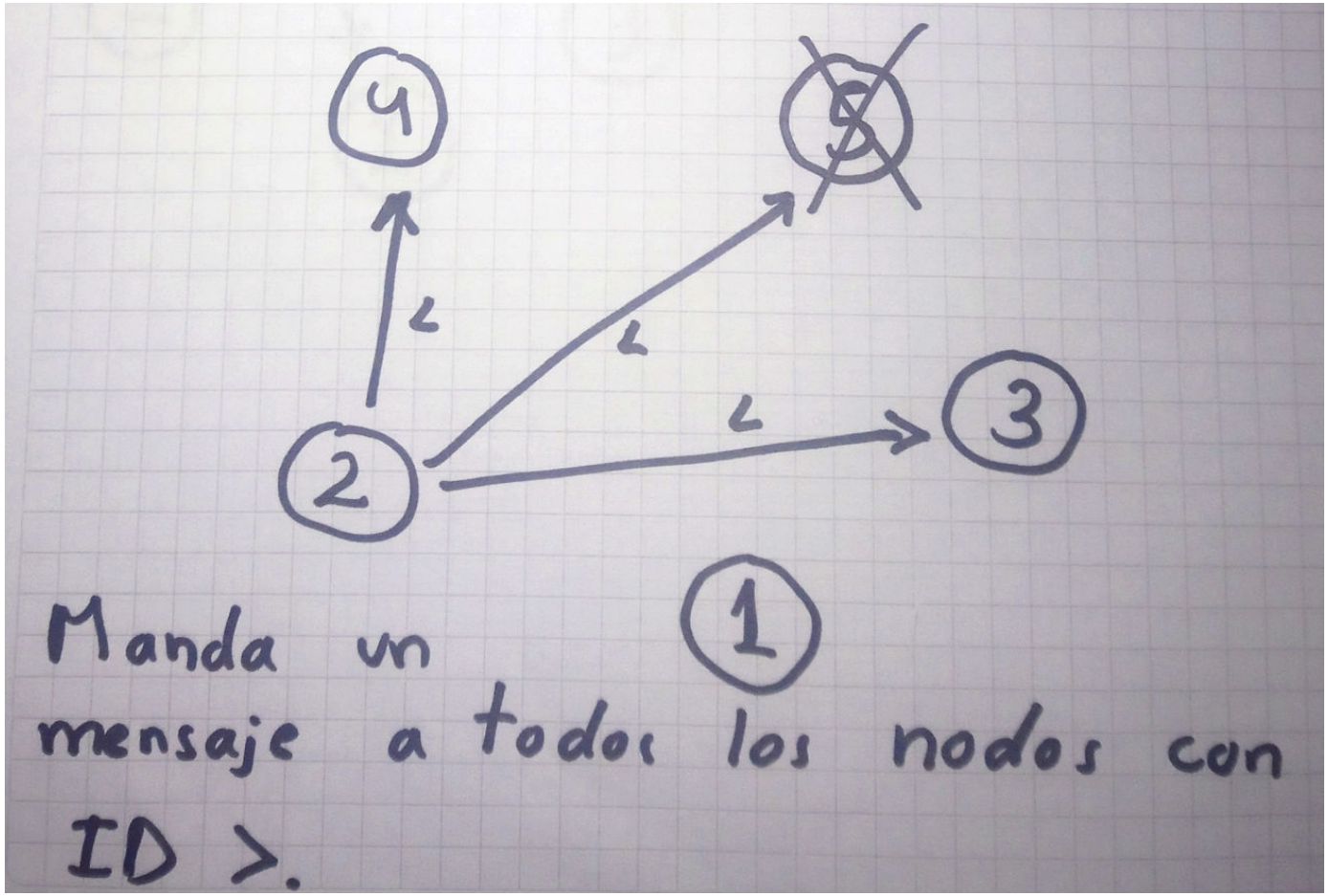
Bully algorithm

Tenemos la siguiente distribución. El nodo 2 quiere ser el líder. El nodo 5 está muerto.



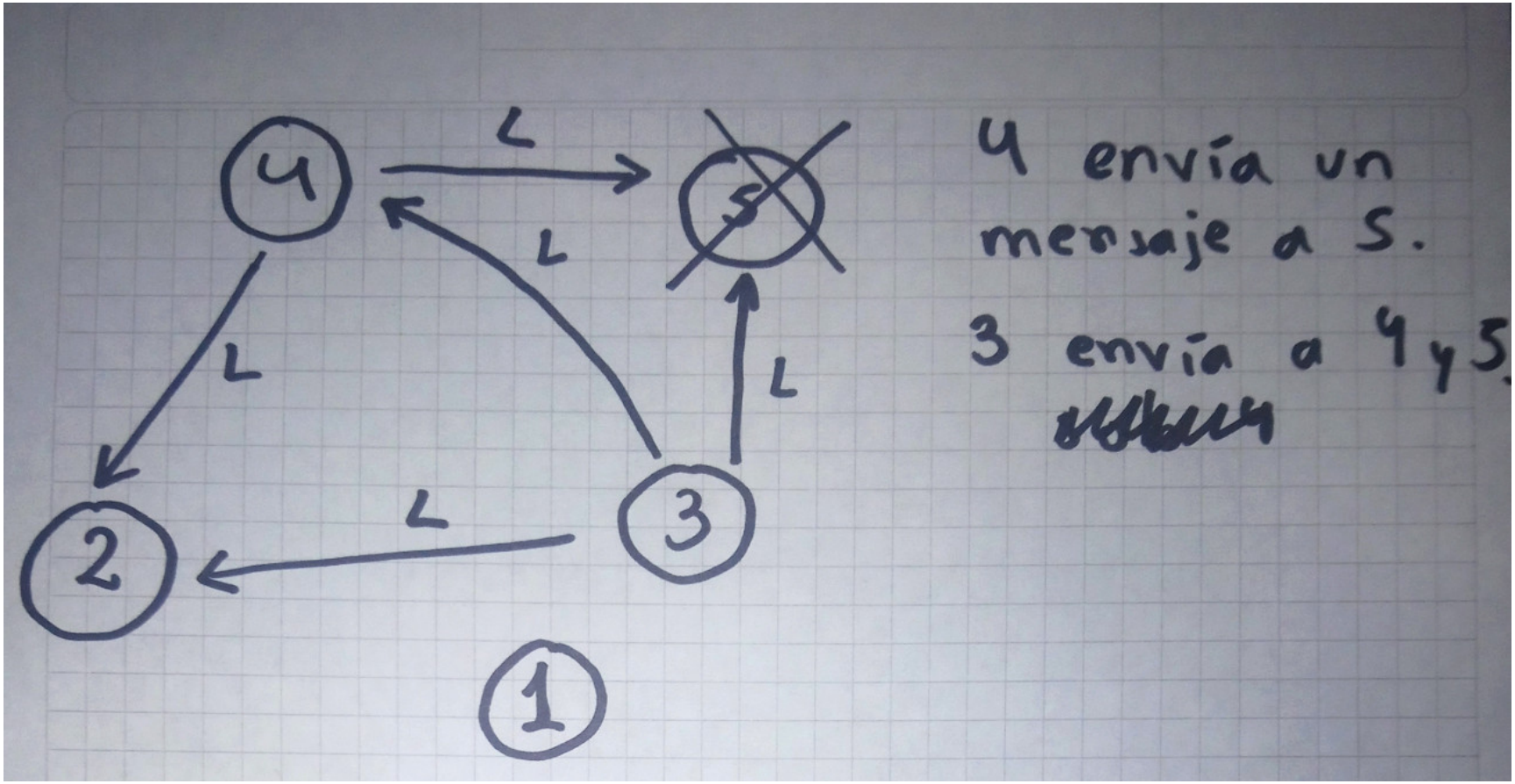
Bully algorithm

2 le manda un mensaje a todos los nodos mayores que él diciendo que quiere ser el líder



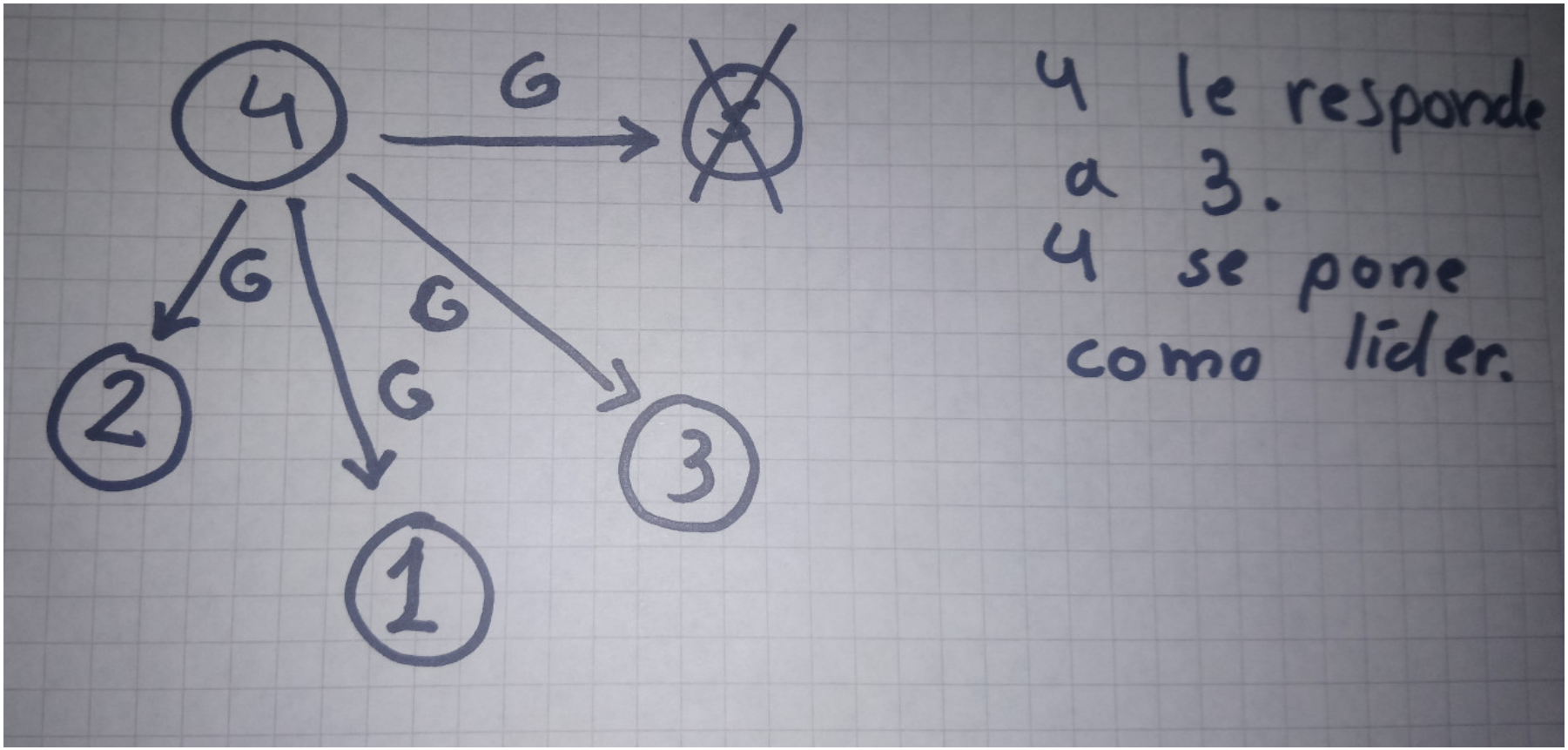
Bully algorithm

Tanto 3 como 4 que están vivos le responden a 2 que ellos van a ser los líderes



Bully algorithm

4 le responde a 3 que 3 no va a ser el líder y luego broadcastea a toda la red que el es el líder.



- ¿Qué hubiera pasado si 5 revivia en alguno de los pasos?
- ¿Cuántos mensajes se mandan aproximadamente?
- $O(n^2)$ con n la cantidad de nodos

Visto hoy

Introducción

Algoritmos de consenso

- 2PC
- 3PC

Elección del líder

- LCR
- Flood Max
- Bully Algorithm

¿Hasta donde puedo hacer?

Toda la guía 7: Sistemas distribuídos

¿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`