laSalle
UNIVERSITAT RAMON LLULL

Projectes en arquitectura distribuïda

Exercici 0.
Distributed systems. A naive view.

Àlex Jordà

24 de setembre de 2018

1. Explica el funcionament del teu algorisme.

   El funcionament d'aquest algorisme es basa en la topologia *token ring*.

   La topologia *ring* (o anell) es caracteritza per un model lògic on cada node es connecta a un únic node mentre que només rep la connexió d'un altre. D'aquesta manera, tots estan entrellaçats en forma d'anell.

   Anant un pas més enllà, incorporant l'ús d'un *token* podem sincronitzar-los per evitar col·lisions entre els servidors i podem senyalitzar quin és el node que ara li toca dur a terme un tasca concreta.

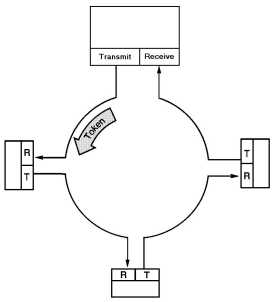

Figura 1: Topologia *token ring*.

En aquesta solució, cada node consta de la informació sobre el port on ell escolta i sobre la següent adreça de la xarxa. També necessita reservar espai per emmagatzemar el *token* i el valor que es necessita conèixer.

El funcionament és el següent:

(a) Quan s'inicia el programa, es llegeix al fitxer `turn.txt` la posició que se li assigna al node. I escriu al mateix fitxer el torn següent, pel pròxim node. Si el seu torn és el 0 vol dir que és el primer node de la xarxa.

(b) Si no és el primer node de la xarxa, realitza una petició al port `INITIAL_PORT` sol·licitant ser afegit a la xarxa. Si la resposta inclou un port vàlid, el nou node s'assigna aquest port com a següent node.

(c) Si és el primer node de la xarxa, assigna el seu següent node com a `INITIAL_PORT`.

(d) Inicia un nou *thread* que crea un servidor escoltant al port `INITIAL_PORT + turn`. Aquest servidor permet rebre dues trames diferents:

   i. Si rep `REQUEST_INSERT`, vol dir que nou node està sol·licitant ser inserit a la xarxa. Aquesta mateixa trama empaqueta també el port al qual es troba assignat aquest nou node. El servidor contestarà amb una trama `REPLY_INSERT` indicant la següent adreça del node, mentre que s'assigna que el seu nou següent node és el que ha sol·licitat la inserció.

   ii. Si rep `SEND_VALUE`, vol dir que el node que apunta a aquest està enviant el valor per a que se'l guardi. Es contesta `REPLY_OK` o `REPLY_KO` segons si la trama és

correcta. Si tot ha funcionat correctament, també implica que es transmet el *token* al node actual.

(e) El *thread* principal entra a un bucle infinit on espera `SLEEP_TIME` segons.

(f) Si el node sí que té el *token*, li envia una trama amb `SEND_VALUE` al següent node, tot indicant el valor actual. Si la comunicació ha anat correctament, el node actual deixa de tenir el *token*.

(g) Torna a l'inici del bucle.

A més a més, en aquesta implementació hi ha previstes diverses fallades com la caiguda d'un node, moment en el que s'intentarà recuperar la caiguda buscant (pràcticament a cegues) un nou node de la xarxa. Motiu pel qual tots els nodes són capaços de permetre una inserció en qualsevol moment de la execució.

2. Quines limitacions li veus al teu algorisme?

   La limitació principal és que un node no té la certesa de tenir el valor correcte fins que té ell el *token*.

   I aquesta situació només es dóna el $\frac{1}{n_{nodes}}$ dels casos, cosa que implica que, per tenir certesa del valor correcte, un node ha d'esperar $(n_{nodes} - 1) \cdot t_{sleep}$ segons.

   On n_{nodes} és el nombre de nodes actual que formen la xarxa i t_{sleep} és el temps d'espera afegit amb la funció `sleep`.

   Tot i estar implementades diverses maneres de recuperar-se d'una caiguda d'un servidor, hi ha un cas fatal en el qual el servidor caigut és també aquell que té el *token*. En aquest cas no hi ha implementada cap manera de recuperar-se'n. Per fer-ho es podria tenir un sistema central el qual gestionés el *token* i sabés qui el té en tot moment. A més a més, podria anar fent *ping* per saber si els servidors estan aixecats i poder recuperar el flux d'execució tot i la caiguda del servidor del *token*. Encara que si caigués aquest "gestor del *token*", estariem en les mateixes. I és per això que el possible sistema de recuperació hauria d'estar implementat en tots els nodes.

3. Com reacciona el rendiment del teu algorisme quan s'incrementa arbitràriament el nombre de processos?

   A causa de la limitació explicada, per cada node que s'insereix a aquesta xarxa, el temps d'espera per a tenir certesa del valor actual s'incrementa t_{sleep} per cada nou node.

   Per arreglar això, el que s'hauria de fer és eliminar l'espera generada amb el procés `sleep`, cas en el que el temps afegit per cada nou node a la xarxa queda reduït a t_{notice}.

   On t_{sleep} és el temps d'espera afegit amb la funció `sleep` i t_{notice} és el temps que triga el *thread*-client d'un node a adonar-se que ell té el *token*.

4. Què caldria canviar, o com de complicat seria permetre que nous servidors s'afegissin al sistema a mitja execució?

   No cal canviar res, el sistema descrit ja permet insercions de nodes a mitja execució.