[ARQUITECTURA DISTRIBUÏDA]

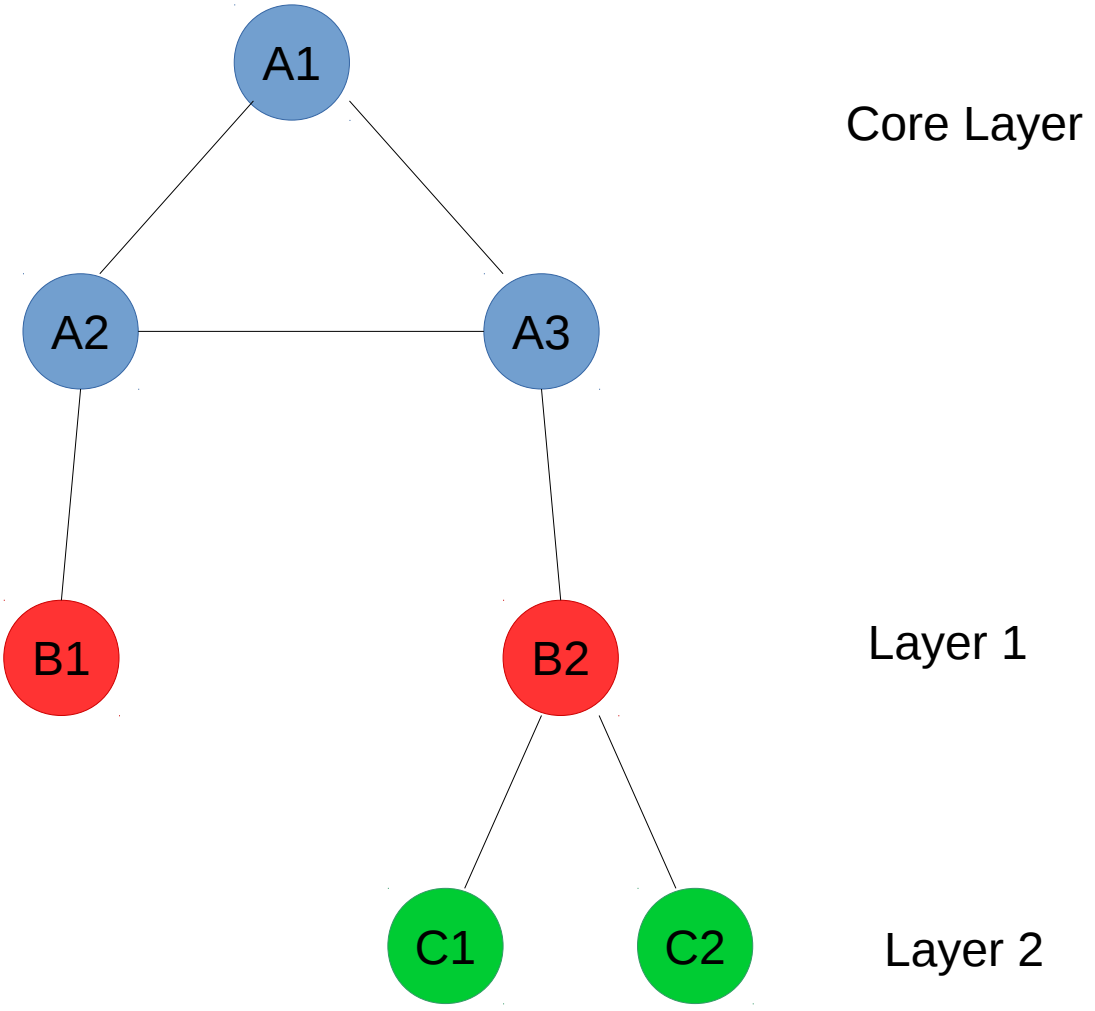
PRACTICA 4

Replicació

Índex

Introducció

Aquesta pràctica es basa a fer un sistema de replicació. Tenim 3 capes: la "Core Layer", la "Layer 1" i la "Layer 2". Tots els nodes poden rebre trames de lectura, però només els nodes de la "Core Layer" poden rebre trames d'escriptura. La "Core Layer" s'actualitzarà mitjançant les directrius: "update everywhere", "active" i "eager replicaction". La "Layer 1" s'actualitzarà cada 10 ordres d'escriptura que rebi la "Core Layer" mitjançant: "lazy", "pasiva" i "primary backup". Finalment la "Layer 2" s'actualitzarà cada 10 segons i rebrà la informació de la "Layer 1" amb les mateixes directrius que aquesta: "lazy", "pasiva" i "primary backup".



Funcionament

Primerament es configuren tots els sockets del sistema i es queden els 7 nodes escoltant pel socket del client esperant que aquest s'iniciï. Un cop està tota la configuració llesta el funcionament de l'algorisme depèn de la capa que implementi, com es mostra a continuació:

"Core Layer"

1. Espera trama.

2. Si es trama client Pas 3, si no Pas 10.

3. Si es trama de lectura Pas 4, si no Pas 5.

4. Genera trama de resposta amb els valors sol·licitats, torna al Pas 1.

5. Per cada operació d'escriptura actualitza l'array de valors, incrementa el comptador d'actualitzacions per a la "Layer 1", afegeix l'actualització a la llista per a la "Layer 1" i afegeix l'operació al String de replicació de la "Core Layer", si es igual a 10 Pas 11, si no Pas 5 fins que s'acabi la trama.

6. Demana espai a la regió crítica mitjançant un Lamport Logical Clock.

7. Espera resposta dels altres nodes de la "Core Layer"

8. Envia l'actualització de tots els valors de la trama a la resta dels nodes de la "Core Layer" i allibera la regió crítica.

9. Actualitza el fitxer local amb els valors.

10. Per cada operació d'escriptura actualitza l'array de valors, incrementa el comptador d'actualitzacions per a la "Layer 1", afegeix l'actualització a la llista per a la "Layer 1" i afegeix l'operació al String de replicació de la "Core Layer", si és igual a 10 Pas 11, si no Pas 5 fins que s'acabi la trama guarda els valors al fitxer local i torna al Pas 1.

11. Fa pop de l'array d'actualitzacions fins que estigui buit mentre genera la trama per la "Layer 1".

12. Envia la trama d'actualització a la "Layer 1" i torna al Pas 1.

"Layer 1"

1. Espera trama.

2. Si es trama d'actualització Pas 3, si no Pas 4.

3. Actualitza els valors de l'array i afegeix el valor al array d'actualitzacions per a la "Layer 2", per cada operació de lectura, en finalitzar la trama guarda els valors al fitxer local i torna al Pas 1.

4. Comprova si hi ha actualitzacions per a la "Layer 2", si les hi ha, genera la trama i li envia, torna al Pas 1.

"Layer 2"

1.1. Espera trama.

2.1. Si es trama actualització, afegeix els valors a l'array, un cop llegida tota la trama guarda els valors al fitxer local, torna al Pas 1.

1.2. Espera 10 segons.

2.2. Demana actualització.

Protocol Lamport

El funcionament d'aquest protocol és senzill, cada node pot enviar 3 tipus de missatge:

REQUEST: Demanar accés a la regió compartida.

REPLY: Contestar en rebre un missatge del tipus REQUEST.

RELEASE: Alliberar la regió compartida.

En començar tots els nodes envien en broadcast una petició REQUEST i s'encuen a la llista. En rebre una petició REQUEST els nodes afegiran aquesta petició a la llista, l'ordenaran de menor a major timestamp i seguidament enviaran un missatge REPLY al node en qüestió. En rebre un REPLY s'augmentarà un comptador, si aquest comptador és 2, voldrà dir que ja ha rebut REPLY d'ambdós nodes, per tant pot comprovar si ell és el primer a la llista, si no és el primer, haurà d'esperar a rebre un RELEASE, desencuar al primer de la llista i tornar-ho a comprovar. Si és el primer de la llista, podrà accedir a la regió compartida, escriure 10 cops per pantalla el seu identificador, esperar 1 segon, seguidament enviar en broadcast un missatge RELEASE, desencuar el primer node de la llista (ell mateix) i enviar en broadcast un missatge REQUEST. Sempre que un node envia un missatge augmenta el seu timestamp en 1 unitat. Sempre que un node rep un missatge actualitza el seu timestamp al màxim entre el timestamp rebut i el propi, seguidament l'augmenta una unitat.

Classes

- LamportClock.java: Classe que implementa un rellotge lògic de Lamport.
- LamportQueueNode.java: Classe per fer la llista de peticions pendents a ambdós protocols. Conté timestamp i id.
- CoreNode.java: Processos A1, A2 i A3.
- Layer1Node.java: Processos B1 i B2.
- Layer2Node.java: Processos C1 i C2.
- Main.java: Crea tots els nodes i els inicia
- StdListener.java: Crea un procés per poder llegir de 2 sockets alhora sense bloquejar el Thread principal.
- Layer2Waiter: Procés que espera 10s i envia petició d'actualització a Layer2.

Conclusions i problemes observats

El principal problema al realitzar aquesta pràctica ha estat el disseny general de la mateixa, ja que vist des de fora feia por, però un cop vaig començar a picar va anar tot fil per randa. Ja que primerament vaig dissenyar només la "Core Layer" i quan aquesta funcionà vaig seguir baixant, capa a capa.

El dubte més gran a l'hora de dissenyar va ser la comunicació entre els nodes de la "Core Layer", tot i que finalment va ser una de les coses més fàcils d'implementar en reaprofitar el Lamport Logical Clock de la pràctica 2.

L'últim mur amb el qual em vaig topar va ser les actualitzacions cap a la "Layer 1" i la "Layer 2", necessitava quelcom per no transmetre els 100 valors de l'array. Ho vaig solucionar utilitzant una llista, afegint i esborrant valors segons anava actualitzant.

En general m'ha semblat una pràctica molt completa i entretinguda, he après molt realitzant aquesta assignatura, tot i la tardança en la realització de les pràctiques.