REPLICACIÓ EPIDÈMICA

Pràctica de projectes en arquitectura distribuïda

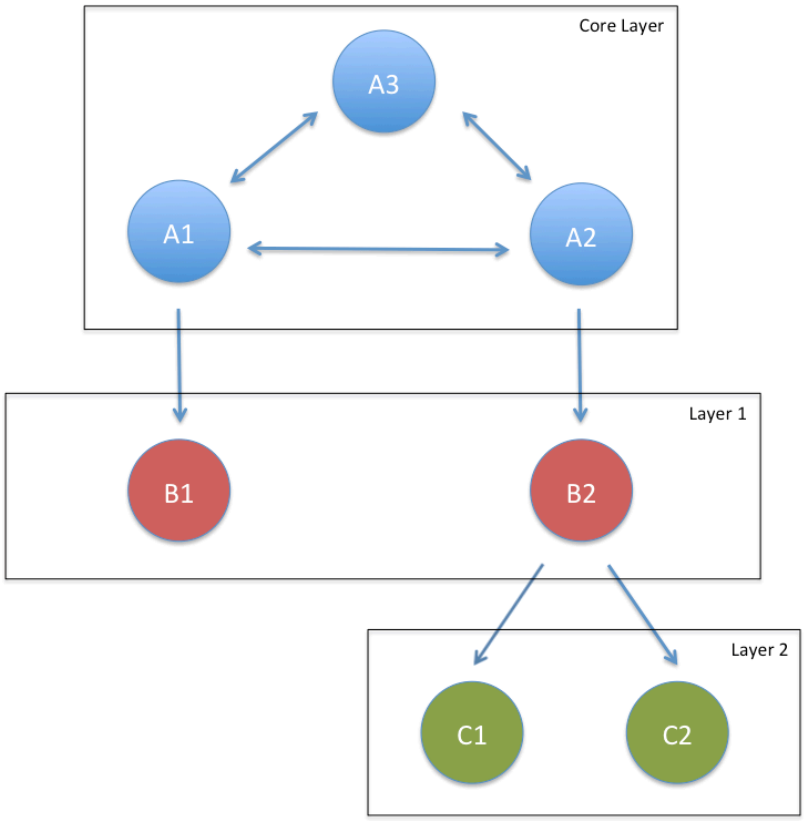
Curs 2015-2016

Balbina Virgili Rocosa
ls27476

Índex

1-. Introducció

L'objectiu principal d'aquesta pràctica és implementar una aplicació distribuïda que s'encarregui de replicar dades de manera epidèmica, aconseguint així una arquitectura de dades multi-versionada.
L'estructura a implementar és la següent, on tots els nodes de la mateixa capa han de tenir sempre la mateixa versió, mantenint sempre un fitxer de logs amb totes les versions que va guardant:



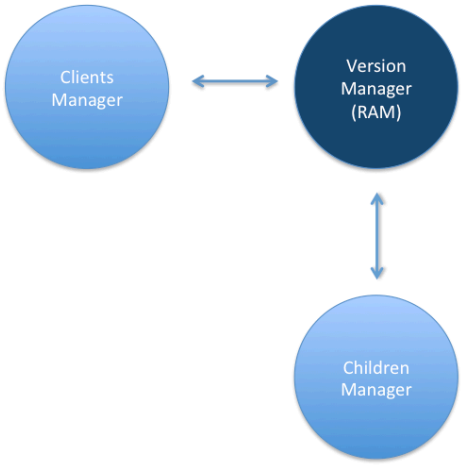
La capa core layer sempre ha de mantenir una copia amb la versió més nova, utilitzant, Update everywhere, active i eager replication.
La capa layer1 rep noves dades cada 10 actualitzacions utilitzant, lazy replication, i primary backup. Finalment, la capa layer2, utilitzant el mateix que la capa Layer1, rep noves dades cada 10 segons.

A més a més, hi ha clients que poden llançar transaccions (read-only, no read-only) des d'un fitxer local a qualsevol de les capes de l'aplicació.

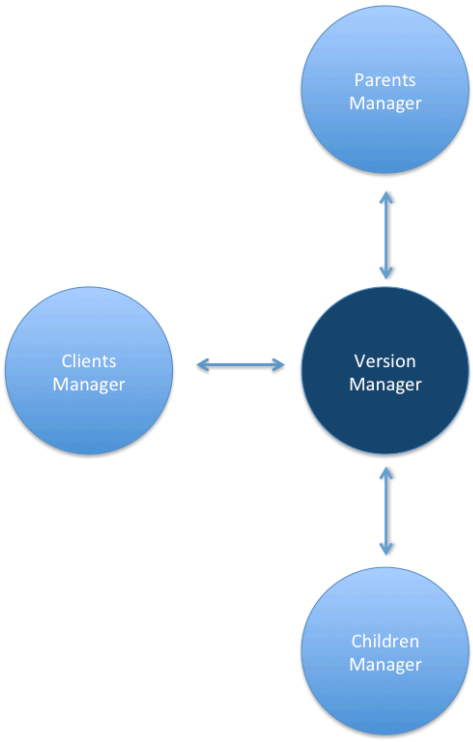
2-. Disseny general del sistema

Per tal de poder implementar aquesta aplicació distribuïda s'ha dividit el sistema en quatre grans blocs:

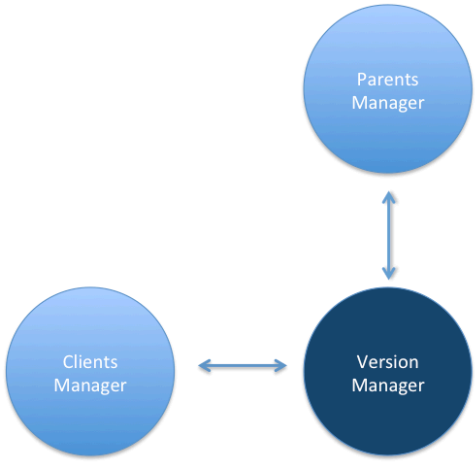
- CoreLayer: aquest mòdul conté una gestió de transaccions que li arriben dels clients que en aquest cas poden ser tant de lectura com d'escriptura, una gestió de versió de la informació la qual es comunica amb els altres nodes de la mateixa capa utilitzant una política d'exclusió mútua Ricart i Agrawala i, finalment, una gestió de nodes de la capa 1 als quals els hi transmet la informació de les noves actualitzacions quan és necessari.



- Layer1: aquest segon mòdul també hi ha una gestió de clients tot i que en aquest cas les transaccions que li arribin només podran ser de lectura, una gestió de versió de la informació actual, una gestió de nodes de la capa 2 als quals li transmet la informació de les noves actualitzacions quan és necessari i, se li afegeix una gestió per tractar la informació de les actualitzacions que li arriba de la capa core.



- Layer2: en aquest tercer mòdul hi ha la gestió per tractar la informació de les versions de les capes superiors, una gestió de la versió de la informació actual i, finalment, també una gestió de les transaccions dels clients que en aquest cas només poden ser de lectura.



- Client: per últim, s'encarrega de gestionar les connexions amb els diferents nodes en funció de la transacció que es vol executar en cada

moment. Obra una connexió amb el node amb el qual vol realitzar cada transacció.

Així doncs, en la implementació s'ha creat una classe base anomenada Layer, de la qual hereten les diferents capes afegint a cada una les funcionalitats que les altres no tenen. Aquesta classe Layer el que defineix són les propietats bàsiques de cada servidor (id, número del socket, versió actual...) i la gestió de clients, ja que és comuna en tots els nodes de les diferents capes.
Tots els nodes esperen la connexió mínima dels altres nodes de l'estructura definida per tal de començar la seva execució.

La comunicació entre nodes es realitza mitjançant sockets, i s'ha utilitzat un HasMap, relacionant id_node i socket, per tal de que un node pugui enviar fàcilment informació a un altre de la seva capa i, igualment llegir-lo.

La capa coreLayer utilitza la política d'exclusió mútua Ricart i Agrawala per tal d'actualitzar correctament les noves versions a partir de les transaccions que els diferents clients poden realitzar als diferents nodes de la capa. Quan un clients sol·licita una transacció d'escriptura, avisa a la gestió de versions que es comunicarà amb els altres nodes per poder realitzar l'actualització. Fins que la resta de nodes no li donen permís per accedir als valors, no s'admet cap més transacció dels clients. En quant se li permet, aquest realitzarà la transacció d'escriptura i informarà a la resta de nodes de la capa. Al mateix temps es podran acceptar noves peticions dels clients amb els valors ja actualitzats.
Aquest control dins del node s'ha implementat gràcies a un ReadWriteLock, el qual accepta múltiples lectures paral·leles per una única escriptura simultània.
També s'ha definit un semàfor, per tal d'actualitzar el valor i les accions de la versió actual del node.

Cada client manté un fitxer local, on cada línia és una transacció i cada vegada que en vol realitzar una es connecta al client, li envia la transacció, espera resposta i es desconnecta.

3-. Proves realitzades

Per tal de comprovar el funcionament de l'aplicació implementada s'ha executat tota l'estructura central de l'aplicació i s'han llançat transaccions des de diferents clients al mateix temps.

Per poder-ho fer, s'han hagut de definit els diferents valors per tal de que cada execució es configurés com a node diferent:

- *private final static int socket;*
    - 4000 + myId per nodes del coreLayer
    - 5000 + myId per nodes del Layer1
    - 6000 + myId per nodes del Layer2
- *private final static int numNodes;*
    - *Número de nodes de la mateixa capa*
- *private final static int myId;*
    - *Id del node (únic per capa)*
- *private final static int layer;*
    - *0 per coreLayer*
    - *1 per Layer1*
    - *2 per Layer 2*

Executant 7 vegades l'aplicació amb la els valors configurats correctament, s'aconsegueix l'estructura desitjada en l'anunciat.

A partir de llavors, només cal arrencar diferents clients canviant la propietat següent per tal de que llegeixi les transaccions del fitxer local que es desitgi en cada moment:

- private final static int numClient;
    - Número del client que es vol configurar

Es pot comprovar a partir dels logs de les diferents execucions que el funcionament és l'esperat, sobretot amb el retorn dels valors que veu el client.
Tot i això, el funcionament també es pot comprovar amb el fitxer que es crea a cada node amb les actualitzacions que va guardant.

4-. Problemes observats i conclusions

Una vegada acabada la pràctica es pot dir que ha sigut una pràctica molt completa de desenvolupar, ja que hi havia moltes coses a tenir en compte, i que pensar molt bé el disseny previ ha sigut clau per la seva implementació per tal de tenir molt clar les diferents funcionalitats de cada node, en funció de la capa en la que es tractava.

Precisament així ha sigut com s'ha pensat la seva implementació, fer-la el màxim de genèrica per tal de que poder reaprofitar funcionalitat, en la mesura del possible, per les diferent capes. El primer problema que va sorgir va ser doncs, encarar la seva implementació ja que hi havia moltes funcionalitats interrelacionades a tenir en compte, tot i que subdividint funcionalitats, poc a poc s'ha aconseguit el seu desenvolupament.

L'altra problema principal que s'ha trobat ha sigut en la implementació de la política de Ricart i Agrawala amb sockets, ja que en l'enviament de paquets alguns no arribaven mai al seu destí, degut a problemes de sincronització de temps entre els diversos servidors. Aquest problema s'ha solucionat esperant sempre un temps entre diferents paquets enviats.

Així doncs, crec que aquesta pràctica ha sigut molt productiva per veure una possible implementació d'un sistema distribuït amb diferents capes de backup, on diferents clients realitzen accions simultànies i tots han d'acabar obtenint el mateix resultat, com si estiguessin apuntant a un únic servidor.