Pràctica AXC – Magatzem de Dades Distribuït

Autors: Pablo Chacin, Xavier Vilajosana, Joan Manuel Marquès

Taula de Continguts

1 Introducció

L'objectiu principal d'aquesta pràctica és posar-vos en contacte amb un cas d'estudi on s'apliquin els conceptes vists al curs de manera integral i es posi de manifest la problemàtica dels sistemes distribuïts, no només des del punt de vista teòric, sinó també de la seva utilització en una aplicació real.

A més, volem introduir-vos a les arquitectures, pràctiques i tecnologies rellevants en el desenvolupament d'aplicacions distribuïdes en el mon real com les que fen servir cada dia.

Més concretament, volem que us familiaritzeu amb conceptes de Web 2.0, on els serveis oferts per les aplicacions es poden compondre, mitjançant mecanismes propis de la web, per crear serveis de valor afegit.

Aquests conceptes s'han concretat en REST (Representational State Transfer), un estil d'arquitectura per aplicacions distribuïdes que se sustenta en dos principis bàsics:

- L'adreçament de recursos i col·leccions de recurs mitjançant URLs
- La manipulació de aquests recurs mitjançant les operacions bàsiques del protocol HTTP (PUT, GET,POST, DELETE)

Els objectius principals que persegueix REST són la simplicitat, la facilitat d'ús, l'escalabilitat, i la possibilitat d'aprofitar la infraestructura d'aplicacions web existent. Aquestes característiques el fan molt convenient per aplicacions interactives que s'executen al navegador (per exemple, aquelles que segueixen el model AJAX). Com a conseqüència, REST s'ha popularitzat i molt serveis oferts pels gegants de la web com Google, Yahoo, Flickr, Twitter, Amazon i molts més, han estat basats en aquest model.

1.1 Treball a realitzar

L'objectiu principal d'aquesta pràctica és el desenvolupament d'un servei per emmagatzemar dades el qual, amb el més pur estil Web 2.0, es pugui fer-se servir per implementar serveis complexos com àlbums de fotos, bitàcoles (blogs), compartir fitxers i més. Per tant, aquest servei haurà de permetre guardar, accedir, modificar i esborrar objectes (fitxers) donat un identificador i fent servir el protocol HTTP.

S'espera que el servei pugi escalar i suportar un nombre molt alt de peticions que segueixen un patró dominat per les lectures (és a dir, molt més lectures que escriptures o modificacions), molt comú a la web.

Per tal d'aconseguir aquest objectius, es requereix la implementació d'una arquitectura web multiestrat i els mecanismes per la coordinació dels servidors que la componen.

Més concretament, s'ha plantejat implementar una arquitectura amb replicació activa de servidors que hauran de coordinar-se per mantenir la consistència de les dades (veure l'apartat "Arquitectura" per més detalls). Aquesta coordinació requereix dos algoritmes. Per una banda, un algoritme d'elecció de líder per triar un coordinar pel grup de servidors replicats i de l'altre banda un algoritme de confirmació per garantir que les actualitzacions s'apliquin de manera consistent a totes les rèpliques.

Per completar aquesta pràctica, no partireu des de zero. Us donarem un conjunt de components que vosaltres heu de completar i integrar. La pràctica posa l'èmfasi més en el disseny i comprensió de la solució que en els detalls de programació de baix nivell. Tot i això, haureu de programar en Java i per tant s'espera que tingueu coneixements genèrics de programació i coneixements bàsics d'aquest llenguatge.

1.2 Pla de treball

La pràctica consisteix en tres fases:

Fase 1: Disseny de la solució al problema, identificant les opcions disponibles i avaluació de la solució proposada.

Fase 2: Implementació d'un algoritme distribuït per l'elecció d'un coordinador o líder dels servidors replicats.

Fase 3. Integració de totes les peces de l'aplicació. Heu de fer que els servidors es comuniquin per l'elecció del líder i per mantenir la integritat de les dades fent servir l'algoritme de confirmació que us donem fet.

1.3 Avaluació

1. Lliurar només la Fase 1 permet optar a una nota màxima de C-.
2. Lliurar la Fase 1 i la Fase 2 permet optar a una nota màxima de B.
3. Fer totes les Fases permet optar a la nota màxima de A.
4. Fer les fases 1 i 3 amb la implementació de l'algoritme d'elecció que us donem fet (que es descriu a l'annex III) en lloc de l'algoritme a desenvolupar en la fase 2), permet optar a una nota màxima de C+.

L'avaluació considerarà aspectes com: el correcte funcionament de l'aplicació, la qualitat de l'anàlisi i les reflexions sobre la solució, la qualitat de la documentació. S'espera no només que feu funcionar l'aplicació, sinó que també sapigueu com funciona i quin és el seu comportament esperat. A més, heu d'entendre com funcionen els components que us donem fets, per la qual cosa trobareu informació en aquest document i al codi dels components.

1.4 Grups

La pràctica es pot fer individualment o en grups de dues persones. En cas de fer-la en grups de dues persones cal avisar al consultor de quins són els components del grup com a molt tard una setmana després de la publicació de l'enunciat. Un cop passada aquesta data no s'admetran nous grups ni canvis de grups. Recomanem que feu la pràctica en grups.

1.5 Lectures complementaries

Aquesta pràctica es desenvolupa al voltant de conceptes molt actuals al món dels serveis web que feu servir cada dia. Tot i que això no es necessari per tal de completar la pràctica, considerem que conèixer més detalladament aquest conceptes us facilitarà la comprensió de les implicacions que podrien tenir les decisions de disseny, a més de donar-vos una idea de la seva aplicabilitat en situacions reals i la seva importància al futur del web i els sistemes distribuïts..

Per tant, us recomanem els següents articles (en anglès)

Web 2.0

1. Tim Oreilly, What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software, Communications & Strategies, No. 1, p 17, First Quarter 2007, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1008839

SOA

2. Huhns, M.N. and Singh, M.P. Service-oriented computing: key concepts and principles,

IEEE Internet Computing, 9(1):75-81, January 2005,
http://www.cs.utt.ro/~ioana/cbse/SOC2005.pdf

REST

3. Roy T. Fielding and Richard N. Taylor, Principled Design of theModernWeb Architecture, Proceedings of the 22nd international conference on Software Engineering, 2000, http://www.ics.uci.edu/~fielding/pubs/webarch_icse2000.pdf
4. Alex Rodriguez, RESTful Web services: The basics, IBM Developer Works, http://www.ibm.com/developerworks/webservices/library/ws-restful/index.html
5. Rohit Khare and Richard N. Taylor, Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems, Proceedings of the 26th International Conference on Software Engineering, http://*www.ics.uci.edu/~rohit/AR**REST**ED-ICSE.pdf*

AJAX

6. Ali Mesbah and Arie van Deursen, An Architectural Style for Ajax, Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture, 2007, http://arxiv.org/abs/cs/0608111

2 Descripció general del sistema

2.1 Arquitectura

Com es pot veure a la figura hi ha un servidor de dades, un grup de servidors cau replicats i un servidor intermediari[1]. El servidor de dades s'encarrega de mantenir les dades de manera permanent mentre que els servidors cau mantenen en memòria les dades més freqüentment accedides. El servidor intermediari s'encarrega de distribuir les peticions entre els servidors cau de forma aleatòria per tal de balancejar les carregues.

Els servidors cau estan organitzats de manera descentralitzada, però en cada moment estan coordinats per un líder.

Tant la comunicació entre els clients, el servidor intermediari, els servidores cau i el servidor de dades, es realitzen mitjançant HTTP. La coordinació entre els servidors cau es fa mitjançant una llibreria per algoritmes distribuïts anomenada DAT que podeu veure a l'annex "Distributed Algorithms Testbed".

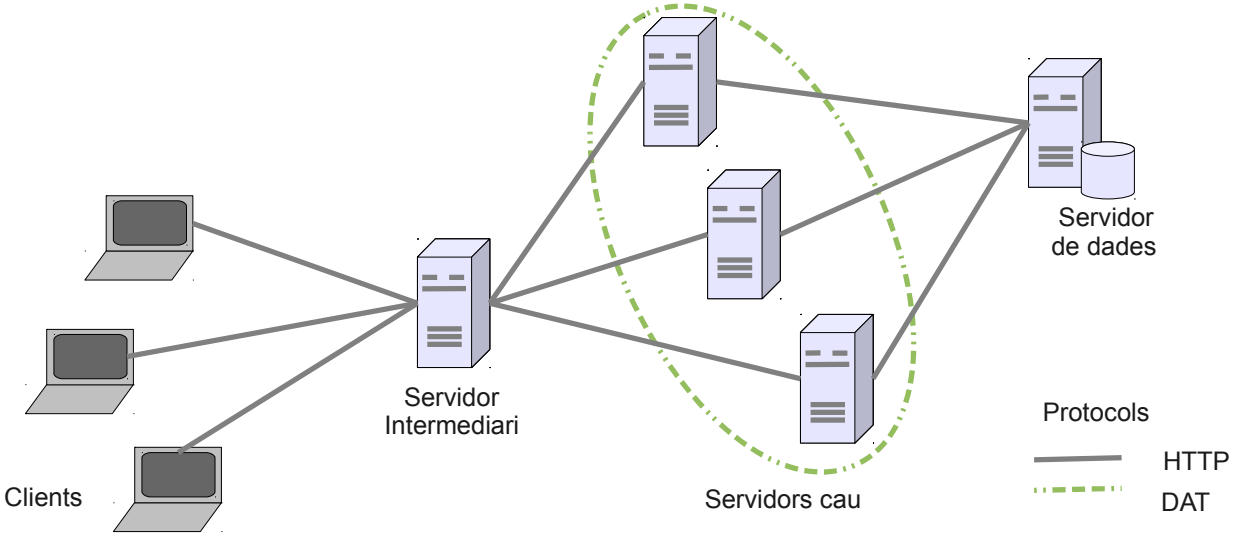


Figura 1. Arquitectura del sistema

2.2 Funcionament General

El funcionament del magatzem haurà de ser el següent:

1. Els usuaris creen i accedeixen a entrades (objectes) en el magatzem utilitzant una aplicació client, típicament des del navegador web. Cada objecte tindrà un identificador únic que permetrà l'accés a les seves dades. Les dades d'un objecte es tracten com un objecte binari sense fer-se cap suposició prèvia sobre el seu format.
2. Tots els servidors cau funcionaran sota un model de rèpliques actives: tots podran atendre peticions dels clients i tots tindran accés a totes les dades.
3. La replicació del servidors cau serà transparent per les aplicacions clients que podran introduir, consultar, actualitzar i esborrar les dades d'un objecte des de qualsevol servidor.
4. Els servidors cau estan organitzats de manera descentralitzada però estan coordinats per un

1 Podríem haver-hi més d'un intermediari, però com que aquest no té cap estat propi per sincronitzar amb altres replicacions, aquesta cas no ens interessa. En el cas del servidor de dades, també podria estar replicat amb sota un model de replicació passiva, però això seria transparent en l'arquitectura.

líder escollit per consens dels mateixos servidors.

5. El sistema ha de funcionar encara que el líder canviï o falli i encara que, en algun moment, hi hagi servidors que no coincideixin en qui és el líder (això pot passar, per exemple, quan un servidor es connecta o falla).
6. En rebre una petició el intermediari es comunicarà amb un dels servidors cau escollit aleatòriament. En cas que el servidor no contesti, ho reintentarà amb un altre servidor també escollit aleatòriament.
7. En rebre una petició de consulta d'un objecte, el servidor cau accedirà a les seves dades des del servidor de dades i les guardarà en memòria. Futures consultes al mateix objecte les respondrà sense accedir al servidor de dades.
8. En rebre una petició de modificació, el servidor cau haurà de demanar al líder que coordiní amb la resta dels servidors cau per realitzar l'operació demanada, garantint la consistència de les replica de la memòria cau.
9. El líder fa servir un algoritme de confirmació per mantenir la integritat de les rèpliques. Quan tots el servidors cau responen, el líder actualitza les dades en el servidor de dades.

   Aquest flux de missatges es descriu de manera general en la figura 2. El missatges concrets que s'intercanvien entre els servidores dependrà del protocol de confirmació específic que s'utilitzi.

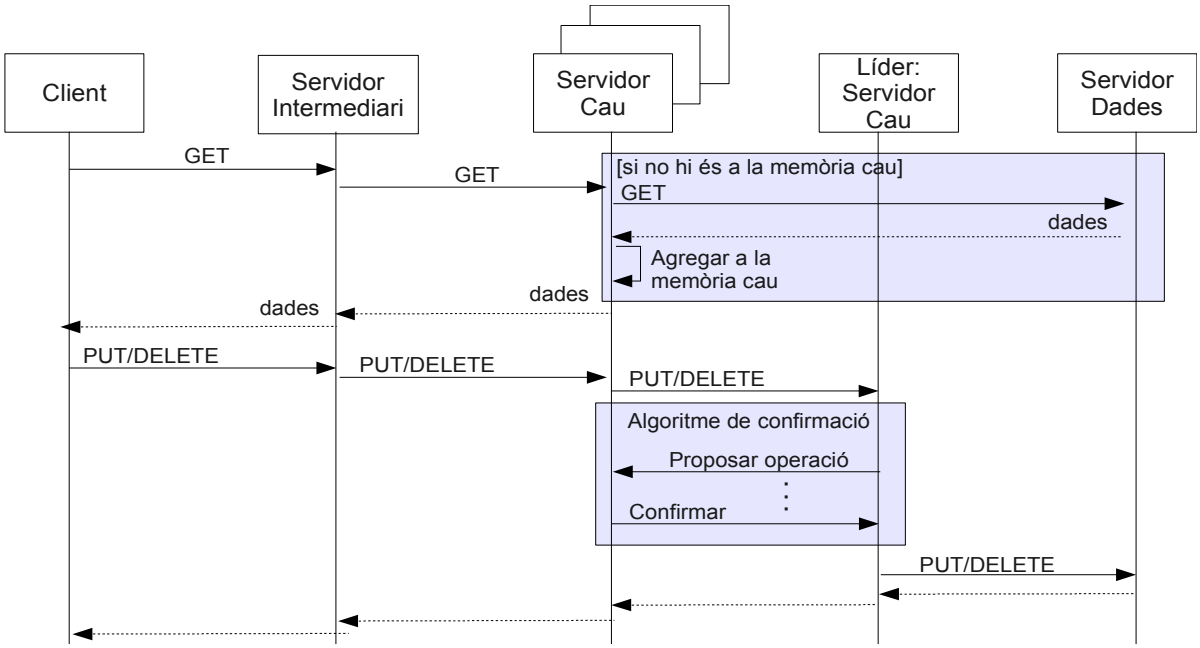


Figura 2. Interaccions entre els servidors.

2.3 Suposicions

- Els servidor intermediari i de dades no fallen mai (es podria pensar que hi ha varis d'aquest servidors en mode de replicació passiva i que s'activen en cas que el servidor principal falli)
- Qualsevol servidor cau es pot desconnectar o fallar en qualsevol moment però sempre hi haurà la meitat més un dels servidors disponibles. (Per exemple, si hi ha 5 servidors, podem comptar que sempre n'hi haurà 3 de disponibles).
- Els missatges no es perden.

3 Descripció del treball

3.1 Fase 1: Disseny preliminar

L'objectiu principal és relacionar el problema plantejat amb la teoria vista al curs, i familiaritzar-vos amb l'entorn i les eines de treball.

Per aquesta fase heu de lliurar un informe que ha de considerar, però no limitar-se, als aparts següents:

- Descriure la problemàtica de coordinació dels servidors cau en el context plantejat
- Descriure i comparar les opcions propostes en aquest enunciat amb altres alternatives de solució quant a la seva efectivitat, rendiment, complexitat, adequació a les suposicions i objectius del cas d'estudi, etcètera.
  Es valoraren positivament els judicis crítics ja que es pot donar el cas que les alternatives propostes no siguin les millor o fins i tot que no funcionin en tots el casos.
- Afegir un breu resum de les referències que heu fer servir per sustentar les vostres decisions (un parell de paràgrafs per cadascuna). Es valorarà positivament si feu servir fonts addicionals a les propostes als materials del curs.
- Fer una descripció del funcionament dels components que us donarem fets així com de les eines de desenvolupament.
- Fer una descripció del funcionament esperat del components que heu de desenvolupar o modificar.

Per completar aquesta fase, podeu consider, però no us limiteu a, les següents preguntes clau:

- Quines són les funcions de cadascú dels servidors descrits a l'arquitectura?
- Quins són els algoritmes distribuïts que es fan servir?
- Quins són els rols que deu executar un servidor cau?
- Quines diferencies existeixen entre un servidor cau qualsevol i un que sigui el líder? Com processen els requeriments de dades cadascú d'ells ?
- Com es comuniquen els participants d'una transacció?. Quins mecanismes de comunicació fan servir? Com són les seves interaccions?
- Com es controla que només una transacció pugui fer modificacions a una entrada de la memòria cau alhora? Quins mecanismes de sincronització es fan servir? Per què es diferent a una simple sincronització entre fills d'execució en un mateix node?

En aquesta fase no podeu fer copiar i enganxar d'aquest enunciat, dels materials del curs o de cap altre font. **Heu de fer servir les vostres pròpies paraules.**

Les fonts que consulteu han de ser fonts primàries, es a dir, articles de recerca, tesis de mestria o doctorat o llibres. No s'admeteren monografies d'estudiants, articles d'opinió ni articles de wikipedia, tot i que podeu ferles servir per trobar altres referències.

3.2 Fase 2: Elecció del líder

Un algoritme d'elecció del lider serveix per a que un conjunt de processos en triïn un que coordini alguna tasca. Un procés comença el procés d'elecció de líder en les situacions següents:

- acaba d'entrar al sistema
- el líder actual no respon
- ha rebut un missatge d'elecció de líder d'un altre procés

En aquesta fase heu d'implementar amb la llibreria DAT l'algoritme ***d'elecció de líder*** anomenat LCR (Le Lann, Chang, Roberts), que està descrit a l'annex II. Per a aquesta implementació, us

donem un esquelet de codi amb la definició de la interfície i les classes principals.

L'algoritme d'elecció de líder ofereix la interfície ***LeaderElection*** amb operacions que permeten a una aplicació (en aquest cas, els servidors cau) triar un líder entre un grup de rèpliques.

public NodeAddress getLeader(): retorna el líder actual o inicia l'elecció si no hi ha cap.
public NodeAddress electLeader(): demana l'elecció del líder i retorna el líder que ha estat elegit.
public boolean isLeader(): indica si aquest node és el líder actual. Si encara cap líder ha estat elegit, inicia el procés d'elecció.

L'elecció d'un líder pot iniciar-se en diferents casos:

- Quan un servidor s'incorpora al grup (per primera vagada o després de recuperar-se d'una fallada)
- Quan un servidor sospita que el líder actual ha fallat (per exemple, no respon a peticions).
- Quan un servidor que es creu el líder detecta que hi ha un conflicte, perquè rep una petició d'un altre "líder".

Que cal fer

Us donem la classe leader.lcr.LCR.java amb l'esquelet de l'algoritme, la qual heu de completar seguint les indicacions que trobareu al codi.

Proves

Les proves de l'algoritme d'elecció del líder es faran de manera independent de la resta dels components de l'aplicació i només s'ha de verificar que aquest algoritme efectivament és capaç de triar un líder i recuperar-se en cas de fallades.

Aquestes proves es fan amb l'aplicació ***dat.algorithms.leader.LeaderElectionApplication*** que us donem com part del codi. Aquesta aplicació demana periòdicament l'elecció del líder i desplega un missatge a la consola amb la identificació del líder..

L'aplicació s'ha de executar amb el fitxer de configuració ***leader-test.properties***.

```
>java dat.core.DAT -config.file leader-test.properties
```

Nota: el fitxer de configuració leader.properties s'ha de modificar per especificar l'algorisme d'elecció de lider que s'hagi triat (bully.properties o lcr.properties)

```
#uncomment the one of the following lines to define the leader election algorithm
#include bully.properties
#include lcr.properties
```

Per simular fallades s'ha d'especificar el temps mitjà entre fallades d'un node. Per exemple:

```
>java dat.core.DAT -config.file leader-test.properties -node.failure.delay 30
```

Heu de tenir en compte que es possible, encara que sigui poc probable, que dos o més nodes tinguin una fallada alhora.

3.3 Fase 3: Integració dels components

El servidor cau ha d'implementar dos interfícies ben diferenciades: per una banda, funcionen com servidors web i reben les sol·licituds dels clients (mitjançant el servidor intermediari) i per tant han d'implementar les funcions PUT i GET. Per de l'altre banda han de participar en la coordinació de les modificacions a les dades.

Per tal de simplificar el codi, no s'ha implementat l'operació DELETE, però el funcionament seria molt semblant al PUT.

La classe ***CacheServer.java*** ofereix una implementació base pels servidors cau amb les funcions bàsiques per atendre les sol·licituds HTTP. En aquesta fase, heu de modificar-la per tal de fer servir els algorismes d'elecció del líder i de confirmació de les operacions de modificació de dades.

Integració amb l'algoritme d'elecció del líder

Quan un servidor cau s'inicia o es recupera d'una fallada, ha de contactar amb la resta dels servidors per tal de determinar qui és el líder.

També, quan un servidor fa una sol·licitud al líder i aquest no respon i per tant se sospita que ha fallat, haurà de iniciar aquest procés.

Finalment, pot donar-se una situació de conflicte quan un servidor que es considera el líder, rep una sol·licitud d'un altre servidor que també es considera el líder. En aquest cas, el servidor ha de rebutjar l'operació i començar el procés d'elecció del líder.

Integració amb l'algoritme de confirmació distribuïda

En el nostre cas, el servidors cau faran de manegadors de la memòria cau replicada i hauran de considerar las situacions següents:

- En rebre una sol·licitud d'accés a una entrada (GET), si és a la memòria cau , la retorna. En cas contrari la ha de llegir amb un GET al servidor de dades, posar-la a la memòria cau i retornar-la al client.
- En rebre una sol·licitud de PUT ha fer servir l'algoritme de confirm per garantir una actualització consistent en tots el servidors cau.

Un algoritme de confirmació permet coordinar les modificacions a un conjunt de recursos compartits entre un grup de participants. Aquest algoritme ofereix la interfície **TransactionCoordinator** a l'aplicació:

public void setResourceManager(ResourceManager manager): estableix qui és el manegador dels recurs. ***public boolean executeTransaction(String resource,String transaction,String operation,Object[] data):*** demana l'execució d'una operació sobre un recurs. La implementació ha de tenir cura del bloqueig del recurs, de l'execució i confirmació de l'operació. Retorna un valor booleà que indica si l'operació s'ha executat correctament (veritat) o no (fals).

public boolean executeTransaction(String resource,String operation,Object[] data): semblant a l'operació anterior, però amb una identificació de transacció definida pel manegador de transaccions.

Per tal de participar al procés de confirmació, el servidor cau ha d'implementar la interfície ***ResourceManager*** amb els següent mètodes:

public boolean lock(String recurs,String transaction): demana el bloqueig d'un recurs per executar una transacció. L'aplicació retorna un valor booleà que indica si la petició s'accepta (veritat) o es rebutja (fals).

public void unlock(String recurs,String transaction): demana el lliurement d'un recurs bloquejat per una transacció.

public boolean apply(String recurs,String transaction ,String operation,Object[] data): demana l'execució d'una operació sobre un recurs amb les dades donades. L'aplicació retorna un valor booleà que indica si la petició s'accepta (veritat) o es rebutja (fals). Cada manegador de recursos ha d'interpretar els paràmetres ***operaction*** i ***data*** segons les seves funcionalitats.

public void commit(String recurs,String transaction): confirma una operació. L'aplicació ha de fer els canvis permanents.

public void abort(String recurs,String transaction): cancel·la els canvis d'una transacció no confirmada i allibera el recurs.

En el nostre cas, els recursos que es manegen són les entrades de la memòria cau i l'única operació que es pot executar a la crida ***apply*** és l'actualització de les dades d'una entrada ("PUT").

Que cal fer

Heu de completar la implementació de la classe ***CacheServer***. Més concretament, heu de completar la implementació de les crides de la interfície ***ResouceManager*** que no s'hagin implementat al codi que us donem fet.

En aquesta classe us donem tot el codi per accedir i actualitzar la memòria cau i, si cal, el servidor de dades. També tindreu totes les estructures de dades que heu de fer servir.

Consideracions pel desenvolupament

En la implementació de les funcions que us demanem, heu de considerar els aspectes següents:

- El servidor cau rep com paràmetres l'adreça del servidor de dades (com una cadena de caràcters amb el seu URL) que es fa servir per accedir mitjançant el protocol HTTP a les dades dels objectes.
- El servidor cau obté mitjançant la llibreria DAT, les referencies als algoritmes d'elecció de líder i confirmació, tot dos definits al fitxer de configuració del DAT.
- Es poden rebre múltiples peticions de accés o modificació del mateix recurs o de diferents recursos.
- Quan es fa l'actualització d'un objecte a la crida ***apply***, s'ha de mirar si el node és el líder. En cas afirmatiu, s'han d'actualitzar les dades al servidor de dades. En cas contrari, només s'ha de actualitzar la entrada a la memòria cau.

- L'accés al servidor de dades es pot fer mitjançant la classe ***HttpUtils***, la qual ofereix els mètodes següents per accedir i modificar un objecte a un servidor amb el protocol HTTP

  public int get(String server, String objecte, byte[] content): retorna el contingut d'un objecte donat l'adreça del servidor corresponent (al paràmetre ***content***, en format binari) i un valor sencer amb el codi de retorn HTTP.
  public int put(String server, String objecte,byte[] content): envia al servidor el contingut d'un objecte, donat l'adreça del servidor, el nom de l'objecte i el seu contingut en format binari. Retorna el codi HTTP.

- S'ha de tenir en compte que després d'haver executat una actualització, es pot demanar que es faci un ***abort*** i s'ha de revertir els canvis. Per tant, cal de mantenir un registre (mitjançant una estructura de map) amb l'estat de l'entrada abans d'executar una modificació. Si es demani un ***abort***, s'ha de revertir el continguts de l'entrada a aquest valor previ. Quan s'executi un ***commit***, només cal esborrar-lo. El líder ha de tenir cura de revertir els canvis al servidor de dades.

- S'ha de tenir en compte que cada servidor cau executa múltiples fils d'execució per atendre els clients. Per tant, es requereix un mecanisme per garantir l'exclusió mútua durant el procés d'actualització d'una entrada, des del ***lock*** fins l'a***bort*** o ***commit***.

  No obstant això, els mecanismes que ofereix Java no resulten apropiats en un escenari distribuït (Per què?). Per tant, hem proveït de les funcions corresponent a la classe ***EntryLock***.

  public void reserveEntry(String entry): demana l'accés exclusiu d'un element donat el seu identificador. Si aquest element ha estat reservada, el fil d'execució es bloqueja fins que s'alliberi.

  public void releaseEntry(String entry): allibera un element donat el seu identificador. Si n'hi ha fils d'execució bloquejats esperant per aquesta entrada, desbloqueja al primer de la cua.

  Aquesta classe ofereix una implementació molt simple que no té cura de possibles problemes de abraços mortals (dead locks).

Proves

Les proves de integració es faran amb una eina de proves que permet executar les accions de crear, accedir i modificar d'objectes, tal com es descriu a l'annex V. Aquestes proves es realitzaran amb tots els servidors descrits a l'arquitectura del sistema: intermediari, cau (replicats) i de dades.

Les proves que s'han de realitzar són les següents

1. Afegir, un objecte des d'un client
2. Accedir l'objecte des d'un altre client i comparar els continguts.
3. Modificar l'objecte i consultar-lo des d'un altre client per constatar que s'han aplicat els canvis.

   Lliuraments

Terminis de lliurament

Hi haurà dos lliuraments:

1.Lliurament 1: correspon a la fase 1.

2.Lliurament 2: correspon a les fases 2 i 3.

Les dates dels lliuraments estan publicades a l'aula.

Continguts del lliurament

- Informe sobre el treball realitzat
- Codi generat
- Imatges que mostrin els resultats de les proves

L'informe ha se seguir la pauta següent:

1. Identificació dels membres del grup
2. Introducció, amb un aclariment de quines fases heu fet i què conté la resta del informe
3. Per cada fase heu de crear un capitul que descrigui:
   - Fins a on he arribat respecte als objectius de la fase? Què no he pogut fer ?
   - Documentació del treball executat, resultats, depenent dels objectius de la fase (veure més a baix)
   - Dificultats trobades
   - Conclusions

Per la fase de disseny, la documentació del treball ha d'incloure

- Decisions preses
- Descripció dels elements de la solució
- Anàlisis de la solució: avantatges, limitacions, problemes potencials
- Referències

Per cada fase de programació, la documentació del treball ha d'incloure

- Decisions d'implementació preses
- Estructura de la solució (fitxers desenvolupats o modificats)
- Per cada fitxer desenvolupat o modificat, línies de codi importants
- Limitacions de la implementació
- Indicacions de com compilar el codi, si cal.
- Indicacions sobre dependències externes, si hi hagués alguna
- Proves realitzades i descripció dels resultats obtinguts

Instruccions lliurament

Enviar un missatge a la bústia de lliurament de treballs amb dos fitxers adjunts:

- Informe
- Un fitxer comprimit amb tots els elements del lliurament que s'ha d'anomenar: Cognom1_Cognom2_Nom-AXC-P1.<ext>, on <ext> pot ser una de les extensions següents: rar, zip, tgz, tar.gz, tbz2, i tar.bz2.

El fitxer comprimit ha se seguir la estructura següent:

Fitxer o directori	Continguts	Observacions
LLEGEIX-ME.TXT	Comentari sobre el lliurament	Indicacions de com compilar i executar el codi i executar les proves
/src	Fonts dels programes	Ha de contenir tot el codi, tant els fitxers desenvolupats com els que heu fet servir i siguin necessaris per compilar-lo.
/proves	Resultats de les proves	Ha de contenir fitxer de traces, imatges capturades, vídeos, etcètera.

4 Aspectes generals d'implementació

4.1 Materials de suport

Us proporcionem els següents documents de suport

1. Guia de programació de DAT (annex IV)
2. Guia de programació concurrent en Java

4.2 Eines de desenvolupament

Entorn de programació

Cal implementar l'aplicació en Java. Es requereix la versió 6 de Java SE o superior. No cal fer servir cap entorn de programació però us recomanem que en feu servir algú per facilitar el procés de programació i compilació.

Podeu implementar la pràctica tant en Linux com en Windows. Totes les classes funcionen en ambdós entorns.

Implementació dels servidors

Tots el servidors funcionen com aplicacions Java independents que er implementen un servidor web senzill i fan servir les classes del paquet ***com.sun.net.httpserver*** per rebre peticions HTTP i despatxar-les a un manegador (handler). Aquest manegador té accés a les dades de la petició HTTP com el URL, l'operació (GET, PUT, POST, DELETE), les capçaleres i el continguts enviats pel client, si n'hi hagés algun.

En el nostre cas, hi ha una super classe, ***WebServer.java*** que implementa un manegador que ofereix les funcions bàsiques per atendre les peticions HTTP: identificar l'objecte que es vol accedir, rebre els continguts (en el cas d'un PUT), enviar la resposta i el continguts associats (en el cas d'un GET).

Les subclasses només hauran de implementar el processament d'aquests continguts en les funcions ***getContent*** i ***putContent***. Aquestes funcions han de retornar el codi de resposta HTTP que s'ha de retornar al client (200, 404, 501, etcètera) .

Implementació dels algoritmes distribuïts

La implementació dels algoritmes distribuïts es farà amb la llibreria de comunicació DAT (Distributed Algorithms Testbed) que permet el intercanvi de missatges entre els servidors. A l'annex IV trobareu una breu introducció.

4.3 Preparació de l'entorn de treball

Pel desenvolupament cal instal·lar les llibreries de suport i configurar l'entorn de treball.

Instal·lació

El codi proporcionat el trobareu en el fitxer ***AXC_CODE.zip***

bully.properties

```
cache.properties
dat.properties
lcr.properties
leader.properties
leader-test.properties
log4j.properties
transaccion.properties
src
 |- web
 |- leader
lib
  |- commons-collections-3.1.jar
  |- commons-configuration-1.2.jar
  |- commons-lang-2.1.jar
  |- commons-logging.jar
  |- dat.jar
  |- log4j-1.2.15.jar
```

El directori ***src*** conté el codi font que us proporcionem i ha estat organitzat en els paquets següents:

- web: servidors web (intermediari, cau, dades)
- leader: algoritme d'elecció del líder

El directori ***lib*** conté les llibreries de suport necessaris per les aplicacions i algoritmes desenvolupats.

Els diversos fitxers amb extensió ".properties" contenen la configuració requerida per executar les aplicacions.

Configuració

Per tal de poder executar correctament les aplicacions, cal configurar l'entorn Java. Més concretament, es necessari establir com a part del class path (encaminament de llibreries de java) les llibreries que us donem.

En sistemes Linux

```
>export CLASSPATH=./:lib/dat.jar:lib/commons-collections-3.1.jar:lib/commons-configuration-1.2.jar:lib/commons-lang-2.1.jar:lib/commons-logging.jar:lib/log4j-1.2.15.jar
```

En sistemes Windows

```
>set CLASSPATH=.\;lib\dat.jar;lib\commons-collections-3.1.jar;lib\commons-configuration-1.2.jar;lib\commons-lang-2.1.jar;lib\commons-logging.jar;llib\log4j-1.2.15.jar
```

Codi proporcionat

Juntament amb l'enunciat de la pràctica i la documentació de suport us proporcionem un seguit de fitxers i classes per a que us serviran de punt de partida per la realització de la pràctica.

Servidor de dades

La classe ***DataServer.java*** implementa la funcionalitat per l'emmagatzematge dels objectes com fitxers en un directori.

Servidor Intermediari

La classe ***ProxyServer.java*** implementa el servidor intermediari. Aquest servidor no fa cap processament de les peticions HTTP, però simplement les redirigirà cap a un altre servidor web escollit aleatòriament entre una llista de URLs de servidors que rep com a paràmetre.

Servidor Cau

Us donem l'esquelet del servidor cau ***CacheServer.java*** amb el codi necessari per inicialitzar-lo i processar les diferents peticions HTTP. Aquest codi s'ha de completar per integrar els algoritmes de coordinació dels servidors (elecció del líder i confirmació).

Algoritme d'elecció de líder

El paquet ***dat.algorithms.leader.bully*** conté les classes que implementen l'algoritme d'elecció de líder Bully, que podreu utilitzar com alternativa per poder completar la fase 3 si no heu implementat la fase 2 (veure l'apartat Avaluació de la secció Introducció).

Algoritme de group membership

El paquet ***dat.algorithms.membership.fixed*** conté les classes que implementen l'algoritme de group membership que permet a un procés identificar quins altres processos participen en l'elecció del líder. Aquesta implementació es mot senzilla i es basa en una llista estàtica que es proporciona com part del fitxer de configuració.

L'algorisme no implementa cap mecanisme per detectar fallades i per tant pot retornar en la llista nodes que no s'hagin disponibles. Per tant, els algoritmes han de tenir cura d'aquest tipus de situacions.

Algoritme de confirmació

Com part del codi trobareu la implementació, fent servir la llibreria DAT, de l'algoritme de confirmació en dues fases com es descriu a l'annex I.

Aplicació client

Les proves de la funcionalitat integrada es realitzaran amb una aplicació client que permet executar peticions al magatzem de dades per crear, accedir, actualitzar i esborrar objectes. A més permet realitzar una sèrie d'operacions de manera aleatòria. L'annex V ofereix una descripció de la

utilització d'aquesta eina.

4.4 Proves

Per la realització de les proves, s'ha de executar els diferents components i configurar-los adequadament.

Configuració de l'entorn

Per tal de fer proves es simularà un sistema distribuït desplegant un servidor intermediari, un servidor de dades i un conjunt de servidors cau a la mateixa màquina. Cada servidor utilitzarà un port diferent per http i per la comunicació amb la llibreria DAT.

Servidor		Adreça	Port http	Port DAT
Intermediari		127.0.0.1	8090	-
cau	1		8091	8001
	2		8092	8002
	3		8093	8003
	4		8094	8004
	5		8095	8005
Dades			9090	-

Configuració dels servidors

Tots el servidors web tenen el següents paràmetres

Paràmetre	descripció	Valor per defecte
-port	Port per rebre peticions	8080
-range	Rang de ports, a partir del port inicial, per triar-ne un	10
-address	Adreça per fer bind	127.0.0.1

Servidor intermediari

El servidor intermediari rep els següents paràmetres de configuració des de la línia de comandes

Paràmetre	Descripció	Valor per defecte
-servers	Llista de servidors cap als quals es redirigiran les peticions	-

Servidor de dades

Paràmetre	Descripció	Valor per defecte
-path	Camí al directori del magatzem de dades	Directori d'execució
-port	Port per rebre peticions	8090

Servidors cau

Els servidors cau s'executaran mitjançant la llibreria de algoritmes distribuïts DAT. Per tant, els seus paràmetres es poden definir al fitxer de configuració c***ache.properties*** i també des de la línia de comandaments.

Paràmetre	Descripció	Valor per defecte
app.datastore	Url al servidor de dades	N/A
network.nodes	Nombre de repliques	1
node.failure.delay	Temps medi entre fallades, en segons	0 (no hi ha fallades)
node.failure.recovery	Temps de recuperació de fallades, en segons	3
network.param.socket. address	Adreça per contactar al node des de altres nodes (1)	127.0.0.1
network.param.socket. port	Port per contactar al node des de altres nodes	8000
network.param.socket. range	Rang de ports que pot utilitzar un node (2)	10
alg.membership.param. seeds	Adreces i ports dels servidors que participen en els algoritmes (per exemple, l'elecció de líder) Per a una adreça es pot especificar un rang de ports	127.0.0.1:8000-8010

(1) Si tots els nodes s'executen al mateix ordinador, es pot utilitzar l'adreça 127.0.0.1 (home address). Si els nodes- s'executen a diferents ordinadors, a cadascú se li ha de especificar la seva pròpia adreça.

(2) Quan més de un node s'executa al mateix ordinador, es pot especificar un rang de ports, per tal que cadascú en triï un i evitar conflictes.

Execució dels servidors

Servidor intermediari

El servidor intermediari s'executa com una aplicació java independent:

```
java web.ProxyServer -address 127.0.0.1 -port 8090 -servers
localhost:8091,localhost:8092,localhost:8093,localhost:8094,localhost:8095
```

Servidor de dades

```
java web.DataServer -address 127.0.0.1 -port 9090 -path /tmp/datastore
```

Servidors cau

Els servidors de dades s'executaran com una aplicació dins de l'entorn proveït per la llibreria DAT.

```
java dat.core.DAT -config.file cache.properties
```

Nota: el fitxer de configuració leader.properties s'ha de modificar per especificar l'algorisme d'elecció de lider que s'hagi triat (bully.properties o lcr.properties)

```
#uncomment the one of the following lines to define the leader election algorithm
#include bully.properties
#include lcr.properties
```

Execució de peticions

Les peticions a qualsevol servidor es podran realitzar amb l'eina de proba que us proveïm. L'annex V ofereix una descripció detallada de com utilitzar aquesta eina.

Proves amb fallades

Heu d'iniciar els servidors en mode de fallada, especificant com a paràmetres d'inicialització la mitjana del temps de fallada de cada mode, en segons

```
> java dat.core.DAT -config.file cache.properties -node.failure.delay 30
```

I Annex: Primari-secundari amb confirmació en dues fases

En aquesta pràctica utilitzarem un senzill model de primari-secundari o master-backup, en que un dels servidors farà la funció de líder i processarà totes les peticions d'inserció, modificació i esborrat de dades. Els servidors que no siguin líder poden rebre aquest tipus de peticions, però el que faran serà enviar-les al líder per a que aquest les processi. Un cop el líder rep una petició, l'aplica i l'envia a la resta de servidors per a que també l'apliquin, de manera que tots tinguin la mateixa informació. Per a les operacions de consulta, en canvi, els propis servidors secundaris, poden respondre als usuaris.

Aquest mètode es basa en el fet que sempre hi haurà un sol primari al sistema, i per tant aquest pot tenir potestat total per a decidir quins valors s'afegeixen i en quin ordre s'executen les operacions. En el sistema que volem implementar, però, considerem que cada node pot fallar en qualsevol moment, i per tant la identitat del líder pot anar variant. Fins i tot es pot donar el cas que hi hagi més d'un servidor que cregui que és el líder, i per tant, més d'un servidor intentarà modificar les dades.

Per tal d'evitar possibles conflictes en cas d'haver-hi més d'un líder, introduirem un protocol de confirmació en dues fases. El primer que farà un node que sigui líder quan vulgui executar una operació és contactar amb tots els altres servidors per informar-los de la seva intenció. En rebre aquest missatge, els servidors respondran que hi estan d'acord si no creuen que són líder (la majoria de les situacions) o respondran rebutjant l'operació si creuen que són líder (això passarà rarament, però pot passar tal i com s'ha dit anteriorment)

Si durant aquesta primera fase el líder rep una resposta negativa, això vol dir que no pot executar l'operació, ja que hi ha més d'un líder al grup. L'acció que cal fer llavors és iniciar una elecció de líder, per tal de solucionar el problema. Quan ja hi hagi només un líder, el servidor que tenia la petició tornarà a intentar l'operació des del començament, enviant-la al líder si s'ha triat un servidor diferent, o iniciant el protocol de confirmació en dues fases si torna a ser líder.

En cas que durant la primera fase del protocol només rebi respostes positives, en la segona fase s'enviarà a tots els servidors l'operació a executar. En rebre aquest segon missatge, els servidors executaran l'operació.

La primera fase hauria de servir com a reserva, de manera que, després d'haver rebut aquell missatge, els servidors no acceptin cap altra operació que no es correspongui amb la fase 2 de l'operació que han acceptat a la fase 1.

II Annex: Elecció de líder: algorisme LCR (LeLann, Chang, Roberts)

L'algorisme LCR serveix per a que un conjunt de nodes en triïn un que faci de líder. Es pressuposa una xarxa en forma d'anell, en que cada node coneix el seu veí en una direcció. Un node comença una elecció de líder en les situacions següents:

- acaba d'entrar al sistema
- el líder actual no respon
- ha rebut un missatge d'elecció de líder d'un altre node

Primerament, cada node inicialitza amb el valor del seu identificador una variable que contingui l'identificador més alt que ha vist fins el moment. Farem servir l'adreça del node com el seu identificador. El procés d'elecció consisteix a enviar un missatge d'elecció al node veí que conté l'identificador més alt vist. Quan un node rep un missatge d'elecció de líder, compara l'identificador rebut amb el que té com a més alt, i guarda i re-envia cap al seu veí el més alt dels dos.

Serà líder el node que obtingui un missatge en que el seu propi identificador sigui proposat com a líder. Això voldrà dir que el seu identificador és el més alt dels presents al grup, i per tant ha de ser el nou líder. Aquest node ha d'avisar als altres nodes del seu identificador per a que sàpiguen que ell és el nou líder.

En el nostre cas no tenim una xarxa en forma d'anell, sinó que tots els nodes poden contactar amb qualsevol altre. Tot i així, aquest algorisme és interessant com a exemple d'algorisme d'elecció de líder. Per implementar-lo, per tant, haurem de simular una xarxa d'anell. El que farem serà que cada node enviarà el seu missatge al node següent, ordenant-los de menor a major per la seva adreça (en cas del nodes que siguin a la mateixa adreça, s'ordenaran pel seu port, per example 127.0.0.1:8000, 127.0.0.1:8001, etc.). En cas que el node amb adreça immediatament superior no estigui disponible, perquè estigui en fallada, s'intentarà contactar amb el següent, començant pel de menor adreça en cas de que el node amb port major no respongui, fins a arribar al propi node. Si es dóna aquest cas, el node està sol al grup, i és per tant el líder. En qualsevol altre cas, els nodes envien missatges tal com s'ha explicat, fins que un rebi un missatge proposant-lo com a líder. En aquest moment es proclamarà líder enviant un missatge a tots els altres nodes (independentment de la seva posició en l'anell simulat).

Referències

[1] Gerard Le Lann. Distributed systems–towards a formal approach. In *IFIP Congress*, volume 7, pages 155–160, 1977.

[2] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 22(5):281–283, 1979.

[3] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, 1997.

III Annex: Elecció de líder: algorisme de Bully

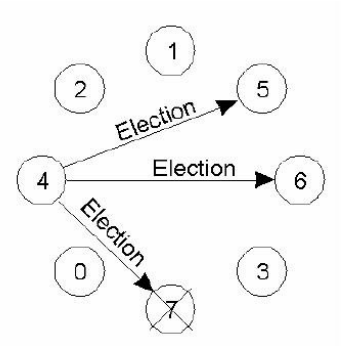
L'algorisme de Bully [1] serveix per a que un conjunt de nodes en triïn un que faci de líder. Un node comença el procés d'elecció de líder en les situacions següents:

- acaba d'entrar al sistema
- el líder actual no respon
- ha rebut un missatge d'elecció de líder d'un altre node

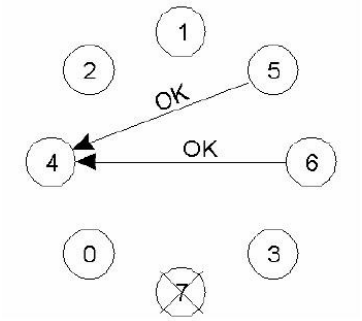
L'algorisme de Bully requereix que cada node tingui un identificador únic i consisteix en triar com líder el node amb l'identificador més alt. En el nostre cas, utilitzarem com identificador l'adreça del node. Un node inicia l'elecció enviant un missatge d'elecció a la resta dels nodes amb identificador superior al seu. (Es pot optimitzar deixant d'enviar elecció quan algun dels servidors amb identificador major respon). Si algun servidor respon, l'origen de l'elecció no fa res més.

Serà líder el node que no obtingui resposta als seus missatges d'elecció adreçats a nodes amb identificador superior al seu. Aquest node ha d'avisar als altres nodes del seu identificador per a que sàpiguen que ell és el nou líder.

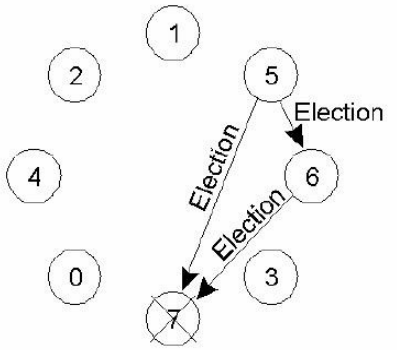
Exemple:



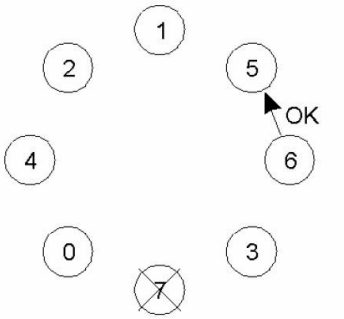
a) El servidor 4 comença una elecció perquè no ha aconseguit comunicar-se amb el node líder (7). Per tant, envia elecció a tots els nodes amb identificador superior al seu.



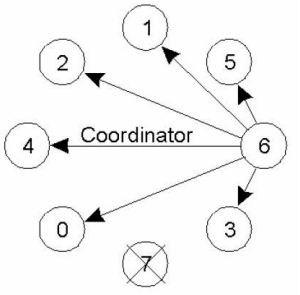
b) Els nodes 5 i 6 responen al 4 comunicant-li que no continuï amb l'elecció perquè ells tenen un identificador major



c) A continuació els nodes 5 i 6 comencen de nou l'elecció.



d) El node 6 li indica al 5 que no continuï, perquè ell te un identificador superior.



e) El node 6 ha sortit líder després de l'elecció i ho comunica als altres nodes.

Referències

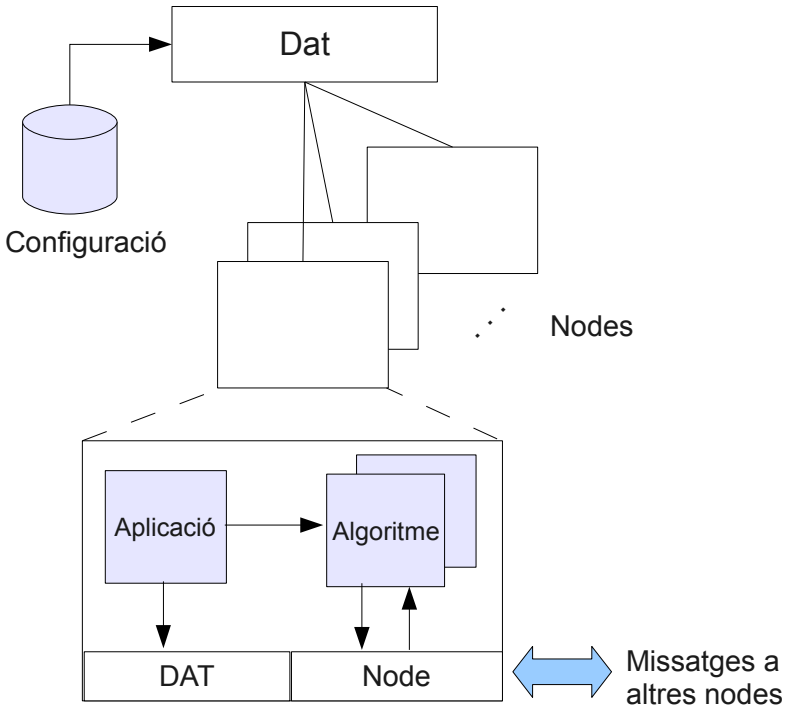
[1] Garcia-Molina, H.. Election in a Distributed Computing System. IEEE Trans. Comp. (31)1:48-59, Jan. 1982

IV Annex: Distributed Algorithms Testbed

A. Introducció

DAT (Distributed Algorithms Testbed) és una eina per desenvolupar algoritmes distribuïts. Ofereix una interfície de programació i les funcionalitats bàsiques per executar aquests algoritmes en múltiples nodes simulats a la mateixa màquina o distribuïts en diferents màquines. El model conceptual del DAT es mostra a la figura següent.

Cada node ofereix un context d'execució per un o més algoritmes i una aplicació que els fa servir per realitzar alguna tasca. Aquest context permet a un algoritme el intercanvi de missatges amb les instàncies del mateix algoritme en altres nodes. Cada algoritme implementa una o més interfícies que refinin les seves funciones (per exemple, elecció de líder, executar una transacció) que són utilitzades per la aplicació o per altres algoritmes. Un algoritme també pot fer servir un altre algoritme per tal de completar les seves funcions..



B. Interfície de programació

La interfície de programació de DAT es divideix en dos parts principals: la interfície per les aplicacions, que permet interaccionar amb els algoritmes i la interfície de programació dels algoritmes, que es accessible només des dels algoritmes en execució.

Un algoritme és una classe que implementa la interfície ***Algorithm,*** la qual defineix els mètodes que el permeten al node interaccionar amb l'algoritme. Generalment, també implementa una altra (o altres) interfície segons les seves funcions, com podria ser arribar a un consens, executar una transacció, reservar un recurs, etcètera.

Algoritmes fan servir la interfície ***Node*** per accedir a les funcionalitats del seu entorn d'execució.

Les aplicacions fan servir la interfície que ofereix la classe DAT per tal de interaccionar amb la plataforma d'execució i accedir als algoritmes.

Interfície Node

El desenvolupament dels algoritmes es fa principalment mitjançant la interfície ***Node*** la qual ofereix les funcionalitats de comunicació i l'accés a l'entorn d'execució local.

El mètode principal, si cal, d'aquesta interfície es ***send***, el qual permet a una instància d'un algoritme enviar missatges a instàncies del mateix algoritme en altres nodes. Com es pot veure, no cal especificar cap a quin algoritme s'enviarà el missatge perquè només es poden enviar missatges entre les instàncies d'un mateix algoritme.

public void ***sendMessage***(NodeAddress destination, Message message): sends a message to the instance of the same algorithm in another NetworkNode, given its NodeAddress.

NodeAddress és una adreça única per cada node que es fa servir per identificar-lo i ubicar-lo. Aquesta adreça té una localització única que la identifica i que es pot fer servir per ordenar els nodes.

Un objecte ***Message*** té un tipus (definit per l'algoritme), un identificador únic, i un conjunt d'atributs amb nom.

El identificador del missatge es genera automàticament quan es crea, però també es pot assignar un identificador específic. Això és molt important per exemple, per assignar a un missatge de resposta el mateix identificador del missatge depetició.

El tipus de missatge és normalment el nom de la classe java (veure l'apartat sobre subclasses de la classe Message més abaix), però l'algoritme pot establir un altre tipus definit per l'algoritme.

Els atributs es poden accedir i modificar amb els mètodes get i put, dependent del seu tipus (per exemple: getString, setString, getLong, setLong, etc) .

Per exemple, per crear un missatge del tipus “Request” amb un dos atributs:

```
Message msg = new Message();
msg.setType(“Request”);
msg.setString(“attribute1”,”value”);
msg.setLong(“attribute2”,100);
```

Per accedir als atributs

```
String value = msg.getString(“attribute1”);
Long num = msg.getLong(“attribute2”;
```

La interfície ***Node*** també permet planificar l'execució d'esdeveniments a un temps específic, com per exemple l'expiració d'un temps fora (timeout)

public void ***scheduleEvent***(*Event event*): schedule the occurrence of an Event at a given time in the

future. It is useful to handle timeouts or to execute periodic actions.

Un ***Event*** te un identificador, un tipus, un temps d'endarreriment (delay) i un conjunt d'atributs amb nom (com els de la classe ***Message).***

El identificador de l'esdeveniment es genera automàticament quan es crea, però també es pot assignar un identificador específic. Això és molt important per exemple, per assignar a un esdeveniment de timeout per esperar una resposta el mateix identificador del un missatge de petició.

Per example, per planificar un esdeveniment del tipus "response.timeout" dins de 1 segon:

```
Event timeout  = new Event();
timeout.setTime(1000);
timeout.setType("response.timeout");
node.scheduleEvent(timeout);
```

El node també permet a l'algoritme accedir als paràmetres de configuració.

public Configuration ***getParameters()***: return the parameters passed to the algorithm in the configuration.

El node permet obtenir una referència a altres algoritmes que s'executen en el mateix node per tal de poder accedir a les seves funcions definida en una interfície pròpia de l'algoritme.

public Object ***getAlgorithm(****String name,Class interficies): gets a reference to another algorithm executed in the same node*

Node ofereix una bitàcola (log) on es poden escriure missatges informatius, per reportar esdeveniments importants i fins i tot, fer una traça de la seva execució. Aquesta bitàcola segueix la interficie Logger de la llibreria log4j.

public Logger ***getLog()****: gets a reference to the node's log, which uses the log4j interface.*

Finalment, els algoritmes poden obtenir els temps d'execució actual amb el mètode

public Long ***getTime()****: gets the current time for the node.*

Aquest mètode hauria de fer-se servir en lloc de ***System.currentTimeMillis()*** per tal de garantir que s'obté el temps correcte depenen de aspectes tal com la simulació de divers velocitats d'execució, l'ús de rellotges virtuals i altres factors que podrien fer variar aquest temps.

Interfície Algoritme

L'algoritme ha de implementar aquesta interfície per tal de poder interaccionar amb el node en el qual resideix i amb les altres instàncies en altres nodes.

El mètode ***init*** es cridat una vegada l'algoritme és carregat i li permet fer les tasques de inicialització. Rep una referència al node en el qual s'executa.

public void **init***(Node node)*: initializes the algorithm, receiving a reference to its execution context.

La tasca més importat de qualsevol algoritme distribuït es el processament dels missatges que rep des de la resta del nodes de la xarxa. Aquests missatges es processen al mètode ***handleMessage***.

public void ***handleMessage****(Message message)*: handles the reception of a message from another node

L'algoritme també ha d'atendre els esdeveniments que es generen durant la seva execució, com per exemple l'expiració del termini per rebre una resposta d'un altre node.

public public ***handleEvent****(Event event): handles the occurrence of an event, which it triggered either by the same algorithm or by the DAT platform. An Event has a type, an unique ID and a map of attributes that describes it.*

És important fer notar, que els algoritmes poden estendre els mètodes ***handleMessage*** i ***handleEvent*** per tal de processar tipus de missatges específics que estenen les classes ***Message*** i ***Event***. Per exemple:

```
public void handleMessage(MyMessageType message){

}

public void handleMessage(Message message){

}
```

On ***MyMessageType*** es defineix com:

```
public classs MyMessageType extends Message {

}
```

El mètode ***handleMessage(MyMessageType message)*** s'executarà quan l'algoritme rebi un missatge del tipus ***MyMessageType***, mentre que el mètode ***handleMessage(Message message)*** s'executarà quan rebi un missatge d'una altra classe. Es poden definir tant de tipus de missatges i els corresponent mètodes ***handleMessage*** con sigui necessari per l'algoritme. El mateix mecanisme també aplica als esdeveniments.

Interfície DAT

La classe DAT ofereix els mètodes per accedir a les funcionalitats de la llibreria des de les aplicacions

L'aplicació pot obtenir una referència a un algoritme que s'executi en el mateix node per tal de poder accedir a les seves funcions definida en una interfície pròpia de l'algoritme.

public Object ***getAlgorithm(****String name,Class interficies): gets a reference to another algorithm*

executed in the same node

Una aplicació també pot registrar una classe que implementi la interfície ***Algorithm*** com un algoritme sota el nom que especifiqui i amb els paràmetres de configuració que es passin al moment de registrar-lo..

public void **registerAlgorithm**(String name, Map configuration);

La aplicació també disposa d'una bitàcola (log) on es poden escriure missatges informatius, per reportar esdeveniments importants i fins i tot, fer una traça de la seva execució. Aquesta bitàcola segueix la interfície Logger de la llibreria log4j.

*public Logger **getLog()**: gets a reference to the node's log, which uses the log4j interface.*

C. Configuració

L'execució dels algoritmes amb la llibreria DAT es controla mitjançant un un fitxer de propietats de configuració.

Paràmetre	Descripció	Valor per defecte	Exemple (valor del paràmetre)
network.nodes	Nombre de nodes	1	
network.class	Mecanismes de comunicació entre els nodes	-	dat.network.sockets. SocketNetwork
node.algoritms	Llista d'algoritmes	-	leader,gcast,transaction
node.param.failure	Temps mitjà entre fallades	0	
node.param.recovery	Temps mitjà de recuperació d'una fallada	0	
node.app	Nom de la classe que s'ha de executar com aplicació	-	dat.algorithms.leader.Leader ElectionApplication

Cada algoritme s'especifica amb la configuració següent, que té com a prefix, el nom de l'algoritme

Paràmetre	Descripció	Exemple
alg.<name>.class	Nombre de la classe que implementa l'algoritme	alg.leader.class leader.lcr.LCR
alg.<name>.loglevel	Nivell de detall de la bitàcola de missatges	alg.transaction.loglevel DEBUG
alg.<name>.param. <param name>	Paràmetre de configuració propis de l'algoritme	alg.leader.param.timeout 30

L'aplicación també es pot configurar amb els paràmetres següents

Paràmetre	Descripció	Exemple
app.class	Nom de la classe que implementa l'aplicació	app.class leader.lcr.LCR
app.loglevel	Nivell de missatges de la bitàcola de l'aplicació	app.loglevel INFO
app.param.<name>	Paràmetre de configuració de l'plicació	app.param.delay 3000

D. Bitàcola d'execució

DAT genera diferents tipus de missatges que permeten analitzar l'execució d'un algoritme:

- DEBUG: fa una traça a l'execució de l'algoritme
- INFO: missatges informatius, que donen detalls sobre l'execució de l'algoritme, els esdeveniments comuns, dades de configuració, etcètera
- WARN: missatges d'advertència sobre situacions poc habituals i possiblement indicatives d'un error
- ERROR: missatges que reporten situacions excepcional que afecten el funcionament de l'algoritme.

Aquests missatges es generen amb la llibreria log4j, la qual es configura amb el fitxer log4j.properties. En general, aquesta configuració no cal modificar-la.

El nivell de detall dels missatges que es volen desplegar es pot especificar per a la aplicació, per a un algoritme o de manera general -- que aplica a menys que s'especifiqui un altre per l'aplicació o un algoritme.

El nivell general s'especifica en el fitxer dat.properties

```
dat.loglevel = INFO
```

El nivell per a un algorisme específic es defineix com part dels seus paràmetres:

```
alg.<name>.loglevel = DEBUG
```

Per als missatges de l'aplicació, el nivell es defineix com part dels seus paràmetres:

```
app.loglevel = INFO
```

La configuració que us donem genera missatges cap a la consola i al fiter log.txt. A l'annex VI trobareu un exemple d'aquesta bitàcola generada amb un nivel DEBUG per a l'algorisme “leader”.

V Eina d'execució de proves

Per tal de facilitar l'execució de proves, us donem una eina que permet l'intercanvi de dades amb un servidor fen servir el protocol HTTP. Les dades poden transferir-se des de i cap a un fitxer o la consola on s'executa l'eina.

A. Arguments

HttpClient -m <mètode> <url> [-f <fitxer>]

- mètode: és el tipus d'operació HTTP que s'executarà: GET, PUT, DELETE
- url: conté l'adreça del servidor (incloent el port, si cal) i el camí al objecte que es vol accedir o modificar
- fitxer es el camí d'un fitxer local des de o cap al qual s'enviaran les dades (en el cas dels mètodes GET i PUT)

B. Exemples

Introduir les dades d'un objecte des de la línia de comandament

```
>java test.HttpTest -PUT -u http://localhost:8080/O1
Introduce content ending with with Ctrl-D.
CONTINGUT  DE L'OBJECTE 1
<ctrl-D>
Putting O1 . . .  Done. Size = 25 bytes, execution time 10 milseconds
```

Accedir el contingut d'un objecte

```
>java test.HttpTest -m GET -u http://localhost:8080/O1
Getting O1 . . .  CONTINGUT DE L'OBJECTE 1
Done. Size = 25  bytes, execution time 10 milseconds
```

Deixar les dades d'un objecte a un fitxer

El comandament següent crea el fitxer O1.txt amb les dades de l'objecte O1: :

```
>java test.HttpTest   -m GET -u http://localhost:8080/O1 -f  O1.txt
Getting O1. . .    Done. Size = 35 bytes, execution time 30 milseconds
```

El fitxer O1.txt hauria de contenir ara el text següent

```
CONTINGUT DE L'OBJECTE 1
```

Enviar el contingut fitxer al servidor

Si modifiquem el fitxer O1.txt amb el text següent

```
CONTINGUT MODIFICAT DE L'OBJECTE 1
```

El comandament següent posa aquest contingut al servidors de dades amb el nom O1:

```
>java test.HttpTest -m PUT  -u http://localhost:8080/O1 -f O1.txt
Putting O1 . . .  Done. Size = 35 bytes, execution time 10 milseconds
```

Si accedim a l'objecte, obtenim els resultat següent:

```
>java test.HttpTest -m GET -u http://localhost:8080/O1
Getting O1 . . .  CONTINGUT MODIFICAT DE L'OBJECTE 1
Done. Size = 35  bytes, execution time 10 milseconds
```

VI Annex: Exemple de bitàcola de missatges de la llibreria DAT

La bitàcola generada per la llibreria DAT mostra la següent informació per cada esdeveniments registrat:

- **Timestamp**: temps en mili-segons des del inici de l'execució.
- **Nivell**: nivell de severitat del missatge (INFO, ERROR; DEBUG)
- **Localidad**: Localidad del node on es va generar el missatge.
- **Context**: Qui ha generat el missatge. Pot ser el DAT (“dat), l'aplicació (“application”) o un algoritme (nom de l'algoritme).
- **Text**: Detall que descriu el esdeveniment. En el cas de l'enviament o recepció de missatges, el text conté els detall dels camps del missatge (node qui l'envia, destinatari, tipus, etcètera).
-

L'exemple següent mostra la bitàcola de l'execució de l'algoritme d'elecció de líder LCR amb 3 nodes.

```
130 INFO 127.0.0.1:8002 dat - Starting node
130 INFO 127.0.0.1:8001 dat - Starting node
130 INFO 127.0.0.1:8000 dat - Starting node
8467 DEBUG 127.0.0.1:8005 leader - Sending message {sender=127.0.0.1:8005}{destination=127.0.0.1:8000}{algorithm=leader}{type=lcr.election}{id=c5dd86ac-9d09-4205-8a10-2cc5c6a2998a}{attributes=[[candidate=127.0.0.1:8005]]}
8483 DEBUG 127.0.0.1:8000 leader - Processing Message {sender=127.0.0.1:8005}{destination=127.0.0.1:8000}{algorithm=leader}{type=lcr.election}{id=c5dd86ac-9d09-4205-8a10-2cc5c6a2998a}{attributes=[[candidate=127.0.0.1:8005]]}
8484 DEBUG 127.0.0.1:8000 leader - Sending message {sender=127.0.0.1:8000}{destination=127.0.0.1:8002}{algorithm=leader}{type=lcr.election}{id=50989fd3-1c29-4487-83c1-f1ba17f0d729}{attributes=[[candidate=127.0.0.1:8000]]}
8492 DEBUG 127.0.0.1:8002 leader - Processing Message {sender=127.0.0.1:8000}{destination=127.0.0.1:8002}{algorithm=leader}{type=lcr.election}{id=50989fd3-1c29-4487-83c1-f1ba17f0d729}{attributes=[[candidate=127.0.0.1:8000]]}
8494 DEBUG 127.0.0.1:8002 leader - Sending message {sender=127.0.0.1:8002}{destination=127.0.0.1:8005}{algorithm=leader}{type=lcr.election}{id=127.0.0.1:8001}{attributes=[[candidate=127.0.0.1:8002]]}
8497 DEBUG 127.0.0.1:8005 leader - Processing Message {sender=127.0.0.1:8002}{destination=127.0.0.1:8005}{algorithm=leader}{type=lcr.election}{id=127.0.0.1:8001}{attributes=[[candidate=127.0.0.1:8002]]}
```

8498 DEBUG 127.0.0.1:8005 leader - Sending message {sender=127.0.0.1:8005}{destination=127.0.0.1:8000}{algorithm=leader}{type=lcr.election}{id=c561c353-9941-4a2c-9c92-5fceaa6ae42a}{attributes=[[candidate=127.0.0.1:8002]]}

8500 DEBUG 127.0.0.1:8000 leader - Processing Message {sender=127.0.0.1:8005}{destination=127.0.0.1:8000}{algorithm=leader}{type=lcr.election}{id=c561c353-9941-4a2c-9c92-5fceaa6ae42a}{attributes=[[candidate=127.0.0.1:8002]]}

8500 DEBUG 127.0.0.1:8000 leader - Sending message {sender=127.0.0.1:8000}{destination=127.0.0.1:8002}{algorithm=leader}{type=lcr.election}{id=4dd18d1f-f639-402c-9c3b-e3fd68a9a9b4}{attributes=[[candidate=127.0.0.1:8002]]}

8503 DEBUG 127.0.0.1:8002 leader - Processing Message {sender=127.0.0.1:8000}{destination=127.0.0.1:8002}{algorithm=leader}{type=lcr.election}{id=4dd18d1f-f639-402c-9c3b-e3fd68a9a9b4}{attributes=[[candidate=127.0.0.1:8002]]}

8503 DEBUG 127.0.0.1:8002 leader - Sending message {sender=127.0.0.1:8002}{destination=127.0.0.1:8005}{algorithm=leader}{type=lcr.result}{id=faeb67b4-7829-42a2-9a72-977224a84061}{attributes=[]}

8504 DEBUG 127.0.0.1:8002 leader - Sending message {sender=127.0.0.1:8002}{destination=127.0.0.1:8000}{algorithm=leader}{type=lcr.result}{id=faeb67b4-7829-42a2-9a72-977224a84061}{attributes=[]}

8511 DEBUG 127.0.0.1:8005 leader - Processing Message {sender=127.0.0.1:8002}{destination=127.0.0.1:8005}{algorithm=leader}{type=lcr.result}{id=faeb67b4-7829-42a2-9a72-977224a84061}{attributes=[]}

8541 DEBUG 127.0.0.1:8000 leader - Processing Message {sender=127.0.0.1:8002}{destination=127.0.0.1:8000}{algorithm=leader}{type=lcr.result}{id=faeb67b4-7829-42a2-9a72-977224a84061}{attributes=[]}

8583 INFO 127.0.0.1:8005 leader - Leader:127.0.0.1:8002

10703 DEBUG 127.0.0.1:8004 leader - Sending message {sender=127.0.0.1:8004}{destination=127.0.0.1:8003}{algorithm=leader}{type=lcr.election}{id=d8ce717e-1c09-4b58-844f-93f3a6f37315}{attributes=[[candidate=127.0.0.1:8004]]}

10705 DEBUG 127.0.0.1:8003 leader - Processing Message {sender=127.0.0.1:8004}{destination=127.0.0.1:8003}{algorithm=leader}{type=lcr.election}{id=d8ce717e-1c09-4b58-844f-93f3a6f37315}{attributes=[[candidate=127.0.0.1:8004]]}

10706 DEBUG 127.0.0.1:8003 leader - Sending message {sender=127.0.0.1:8003}{destination=127.0.0.1:8004}{algorithm=leader}{type=lcr.election}{id=6c0632a6-d1c3-46c0-80da-b77c5086a369}{attributes=[[candidate=127.0.0.1:8004]]}

10708 DEBUG 127.0.0.1:8004 leader - Processing Message {sender=127.0.0.1:8003}{destination=127.0.0.1:8004}{algorithm=leader}{type=lcr.election}{id=6c0632a6-d1c3-46c0-80da-b77c5086a369}{attributes=[[candidate=127.0.0.1:8004]]}

10708 DEBUG 127.0.0.1:8004 leader - Sending message {sender=127.0.0.1:8004}{destination=127.0.0.1:8003}{algorithm=leader}{type=lcr.result}{id=fe9912e9-d3b3-4870-bc8d-9ff349a3af2f}{attributes=[]}

10761 DEBUG 127.0.0.1:8003 leader - Processing Message {sender=127.0.0.1:8004}{destination=127.0.0.1:8003}{algorithm=leader}{type=lcr.result}{id=fe9912e9-d3b3-4870-bc8d-9ff349a3af2f}{attributes=[]}

```
10821   INFO  127.0.0.1:8004 application - Leader:127.0.0.1:8004
13367   INFO  127.0.0.1:8000 application- Leader:127.0.0.1:8002
15294   INFO  127.0.0.1:8002 application - Leader:127.0.0.1:8002
19822   INFO  127.0.0.1:8004 application - Leader:127.0.0.1:8004
20368   INFO  127.0.0.1:8000 application - Leader:127.0.0.1:8004
22294   INFO  127.0.0.1:8002 application- Leader:127.0.0.1:8004
26822   INFO  127.0.0.1:8004 application- Leader:127.0.0.1:8004
```