Sincronització, tolerància a fallades i reproducció

Leandro Navarro Moldes
Joan Manuel Marquès i Puig

P07/M2006/02840

UOC
www.uoc.edu

Índex

Introducció 5

Objectius 7

1. L'observació d'un sistema distribuït 9
1.1. Observació i relativitat 9

2. Temps i rellotges 11
2.1. La mesura del temps 11
2.2. Com s'ha d'ajustar el rellotge? 12
2.3. Rellotges lògics 15
2.4. Rellotges vectorials 17

3. Exclusió mútua 19
3.1. Algorisme centralitzat 19
3.2. Algorisme descentralitzat 19
3.3. Algorisme basat en un anell 20
3.4. Algorisme distribuït 21
3.5. Comparació d'algorismes 22

4. Algorismes d'elecció 24
4.1. L'algorisme de Bully 24
4.2. Algorisme anell 25
4.3. Elecció en entorns sense fil 26

5. Tolerància a fallades 27
5.1. Comunicació fiable en grup 30
5.2. Lliurament de missatges 32
5.3. Transaccions en presència de fallades 34

6. Consens 35
6.1. Algorisme de Paxos 36

7. Conceptes bàsics de reproducció 39
7.1. Màster únic enfront de màster múltiple 39
7.2. Sistemes síncrons enfront de sistemes asíncrons 40
7.2.1. Síncrons 40
7.2.2. Asíncrons 41
7.3. Models de reproducció síncrons 41
7.3.1. Reproducció passiva 41
7.3.2. Reproducció activa 42
7.3.3. Basats en quòrums 43

7.4. Algorismes per reproducció síncrona 44
7.4.1. Reserva en dues fases 44
7.4.2. Confirmació distribuïda 44
7.4.3. Confirmació en dues fases 45
7.4.4. Confirmació en tres fases 47
7.5. Reproducció optimista 48
7.5.1. Passos seguits per un sistema optimista per a arribar a un estat consistent 49
7.5.2. Alguns exemples de sistemes optimistes 51

Resum 52

Activitats 55

Exercicis d'autoavaluació 55

Solucionari 57

Glossari 58

Bibliografia 59

Introducció

En aquest mòdul didàctic es descriuen els problemes que presenta i els avantatges que ofereix un sistema informàtic distribuït format per processos i/o màquines que treballen d'una manera independent, que només es comuniquen mitjançant l'intercanvi de missatges per una xarxa que pot retardar, desordenar, duplicar o perdre els missatges.

Un sistema distribuït està format per multitud de components que interaccionen entre ells. L'objectiu del programari intermediari (*middleware*) és tractar amb la xarxa i presentar un model abstracte de programació a les aplicacions senzill i complet, ocultant els inconvenients i oferint serveis per a aprofitar els avantatges: encara que la seva complexitat és més gran que la suma de la complexitat de cada component, és desitjable que el sistema no falli quan falla qualsevol component (fragilitat), sinó que funcioni mentre algun component funcioni (robustesa o tolerància a fallades).

Un sistema distribuït format per diverses màquines pot funcionar de manera coordinada com si tingués una capacitat de procés com a suma de cada component.

El cost d'un sistema de capacitat N vegades superior a la capacitat d'una sola màquina de capacitat mitjana pot costar N vegades el cost de la màquina mitjana, mentre que una sola màquina de capacitat N pot ser impossible de construir o costar massa (C^N).

En realitat, el raonament anterior es pot aplicar tant a la capacitat de processar peticions, com a la capacitat de tolerar errors: amb sistemes distribuïts dissenyats adequadament es poden construir sistemes més ràpids i que funcionen durant més temps que els seus components.

El primer problema que hi ha en un sistema distribuït és que no ens és possible de fer una foto "instantània" o una pel·lícula "global" de tot el sistema per conèixer el seu estat o treure conclusions de com ha passat alguna cosa. L'observador és també un procés i rep missatges com qualsevol altre component d'un sistema distribuït. Es veurà que succeeixen fenòmens similars als que descriu la teoria de la relativitat.

El segon problema és que el temps es mesura localment a cada lloc, no està sincronitzat globalment i pot derivar (accelerar-se o retardar-se respecte d'altres): no hi ha un rellotge global. Es podrien rebre missatges amb marques de temps que sembla que vénen del futur o seqüències de missatges que sembla que tenen un ordre estrany veient les marques de temps que porten del seu origen.

Hi ha protocols per a sincronitzar mútuament els rellotges de cada computador d'un sistema distribuït mitjançant l'intercanvi de missatges i garantir que amb "rellotges lògics" es pot certificar que els missatges es lliuren a les aplicacions respectant les relacions de causalitat (els missatges arriben amb l'ordre en què han ocorregut si un ha influït en l'altre).

El tercer problema rau en la manera de caracteritzar i tractar un sistema en què algun component pugui fallar, algun missatge es pugui perdre, o en què la demanda d'un servei pugui requerir el treball de diversos components d'una manera coordinada. Es presenten les maneres de comportar-se d'un sistema en presència de fallades.

El quart problema és com garantir que la compartició de recursos es faci de manera coordinada. Es presentaran uns quants algorismes que permeten gestionar aquest accés.

Un altre aspecte que tractarem són els algorismes d'elecció, els quals són molt importants en sistemes distribuïts. Molts algorismes necessiten que un procés actuï com a coordinador, iniciador o desenvolupi un paper especial. En aquests casos tant se val qui ho fa, però cal que algú ho faci.

El cinquè problema consisteix a pensar un model de programació que permeti dissenyar aplicacions en les quals diversos components cooperen per proveir un servei més ràpidament, amb una disponibilitat més gran. Aquest és el model de comunicació en grup.

En el mòdul també es presenten els conceptes bàsics relacionats amb la reproducció en sistemes distribuïts. La reproducció permet augmentar la disponibilitat i el rendiment dels sistemes distribuïts. També contribueix a millorar-ne l'escalabilitat. Aquesta reproducció pot ser síncrona –les actualitzacions s'apliquen a totes les còpies de l'objecte com a part de l'operació d'actualització– o asíncrona –com a part de l'operació d'actualització no cal que s'actualitzin totes les còpies de l'objecte. Posteriorment es fa arribar l'actualització a la resta de reproduccions.

Objectius

Els objectius d'aquest mòdul didàctic són els següents:

1. Conèixer els problemes i conceptes bàsics per a observar sistemes distribuïts (temps, ordre, causalitat, errors, grups).

2. Conèixer els avantatges que es poden obtenir d'explotar les "aparents" debilitats (sincronització, tolerància a fallades, alt rendiment).

3. Conèixer les possibilitats que ofereixen alguns entorns per a facilitar la programació, presentant el sistema en un format més tractable (comunicació en grup).

4. Ser capaços de triar les característiques relacionades amb la distribució per a organitzar un sistema que funciona a Internet.

5. Conèixer els conceptes bàsics de reproducció en sistemes distribuïts, especialment en sistemes optimistes.

6. Conèixer alguns algorismes distribuïts populars per a solucionar els aspectes bàsics relacionats amb la construcció de sistemes distribuïts.

1. L'observació d'un sistema distribuït

Un sistema distribuït està format per persones, màquines, processos, "agents" situats en *llocs diferents*. Es pot entendre abstractament com un conjunt de processos que *cooperen* per solucionar un problema, intercanviant missatges. Cada procés és autònom i sovint asíncron: funciona al seu propi ritme. Els missatges poden patir retards arbitraris o perdre's, i no hi ha cap rellotge global.

Per exemple, un computador amb diversos processadors no encaixaria en aquest model: hi ha un rellotge únic, el sistema és síncron, els retards dels missatges són fixos.

Per a tenir una idea clara del que passa en un sistema distribuït i de com programar-lo per a aprofitar els seus avantatges primer s'ha de veure la forma com es pot observar un sistema distribuït, i constatar la dificultat d'arribar a una conclusió sobre l'estat d'un cert moment o l'ordre en què van passar les coses, causat perquè l'observador està subjecte també a l'asincronia del sistema. Passa com en el món físic, que sembla que es regeix per les lleis de Newton i, en realitat, es regeix per les lleis de la teoria de la relativitat d'Einstein. Aquí el problema és que la velocitat de propagació de cada missatge és variable.

1.1. Observació i relativitat

Per a observar una computació distribuïda s'han de rebre missatges de control de tots els processos i construir una "imatge" del sistema, però els missatges de control tenen diferents temps de transmissió: diversos processos "mai" no es poden observar alhora, per tant no es poden fer afirmacions sobre l'estat global.

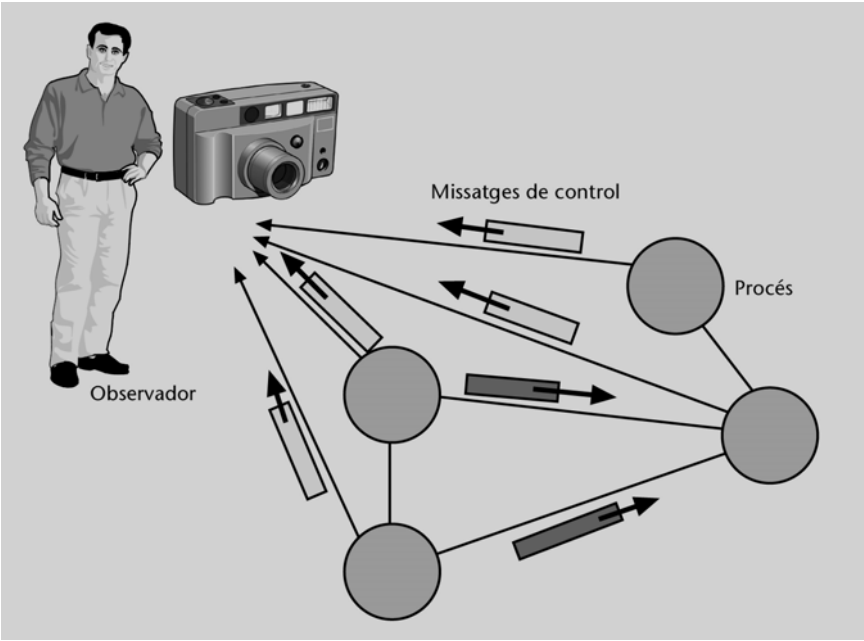


Figura 1. L'observació d'un missatge es fa com un procés més del sistema distribuït

Per exemple, quan es mira el cel en una nit clara, s'està observant una imatge "distorsionada" de l'Univers: veiem la llum que les estrelles van emetre fa moltíssims anys, més anys enrere com més lluny de nosaltres es trobin. En aquest moment és difícil de fer afirmacions sobre l'estat de l'Univers mirant el cel.

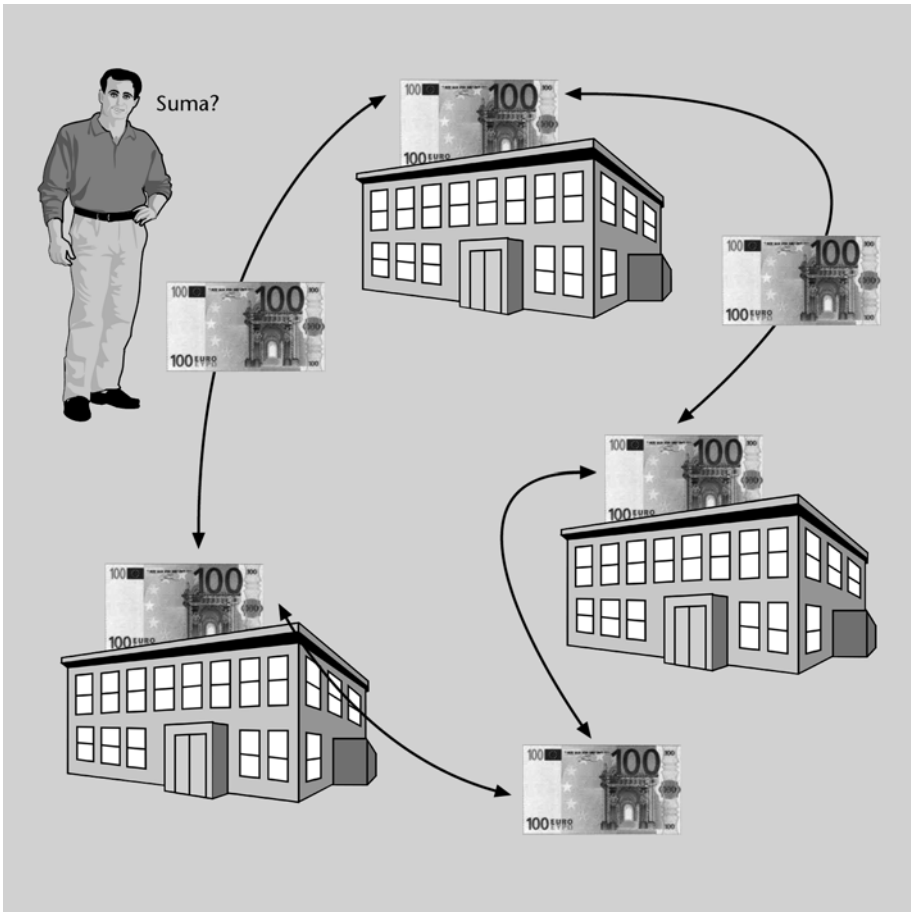


Figura 2. Comptar el valor global del nombre de bitllets en circulació

Un altre exemple: si volguéssim comptabilitzar quants diners existeixen en total (o només els diners emmagatzemats en bancs) hauríem de comptabilitzar els diners que hi ha a cadascun. Es podria fer enviant un missatge a tots alhora demanant-los que ens diguessin l'import total dels seus comptes. Mentre comptem, els diners flueixen i el nostre missatge arriba a cada banc en temps diferents. No podríem calcular-ho bé, ja que podríem haver deixat de comptar, o haver comptat dues vegades, certes quantitats. Per què? No hi ha una visió global (una "foto" global instantània: només un observador omnipresent ho podria saber...).

L'efecte "relativista" de la gravetat sobre la llum no és res comparat amb el que passa a Internet cada dia: la velocitat de propagació és molt menor, les dades que circulen formen cues i pateixen retards per travessar encaminadors, es poden perdre perquè una cua vessa o un bit canvia de valor per error, es poden desordenar, triguen a ser processats per la càrrega del servidor, poden ser emmagatzemats a l'espera de ser reexpedits, etc.

Figura 3. Carta d'Albert Einstein a G.E. Hale del 1913 per a demanar-li que mesurés l'efecte de la gravitació a la llum d'una estrella. No van poder respondre fins que va finalitzar la Primera Guerra Mundial.

2. Temps i rellotges

El temps és una propietat fonamental en sistemes distribuïts. Per a mesurar el temps i saber amb la major exactitud possible el moment en què va succeir algun esdeveniment: és necessari sincronitzar el rellotge del computador amb una font de referència externa.

Una definició de temps

Let us consider first what we mean by time. What is time? It would be nice if we could find a good definition of time. Webster defines "a time" as a "a period" and the latter as "a time" which doesn't seem to be very useful. Perhaps we should say: "Time is what happens when nothing else happens.' Which also doesn't get us very far. Maybe it is just as well if we face the fact that time is one of the things we probably cannot define (in the dictionary sense), and just say that it is what we already know it to be: it is how long we wait!

Richard Feynman

Quan es parla del *temps psicològic* s'han de diferenciar tres tipus de conceptes: el primer tracta l'ordre i la simultaneïtat; el segon els intervals de temps i durada; el tercer és l'experiència que ens permet de distingir entre passat, present i futur.

2.1. La mesura del temps

Un computador és un sistema síncron: està regit per un senyal de rellotge que indica el transcurs del temps. En realitat es tracta d'un comptador d'esdeveniments: es mesura el temps en què ocorre algun fenomen amb regularitat.

Fins al 1955, el patró científic del temps, el segon, es basava en el període de rotació terrestre i es definia com 1/86.400 del dia solar mitjà. Quan es va comprovar que la velocitat de rotació de la Terra, a més a més de ser irregular, estava decreixent gradualment, es va fer necessari redefinir el segon.

El 1955 la Unió Astronòmica Internacional va definir el segon com 1/31.556.925,9747 de l'any solar en curs el 31 de desembre de 1899. La definició de segon des del 1967 és: 1 segon = 9.192.631.770 períodes de la radiació corresponent a la transició entre dos nivells hiperfins de l'estat fonamental de l'àtom de Cs^{133}.

Els computadors solen utilitzar un rellotge de quars (inventat el 1927): un dispositiu que mesura oscil·lacions gràcies a l'efecte piezoelèctric. La seva precisió és elevada: 10^{-6}, és a dir, 1 segon cada 10^6 segons = cada 11,6 dies.

Per a mesures de temps de referència més precisos s'usen rellotges atòmics de Cesi que tenen una precisió de 10^{-13} (10^7 vegades més precís que un rellotge de quars).

En 1/1000 segon = 1 ms:

- En el sistema de posicionament global (GPS) les referències de temps s'obtenen a partir de tres o quatre referències de satèl·lits amb rellotges atòmics amb una precisió d'1 ms.
- Un rellotge de quars (10^{-6}) es pot desviar 1 ms cada 20 minuts!
- La llum, a 3^8 metres/s viatja a 300 km/ms.
- Un computador amb un processador d'1 GHz (10^9) pot executar fins a 1 milió d'instruccions en 1 ms!

El temps universal coordinat (UTC) és un estàndard internacional de temps. Es basa en el temps atòmic i s'emet per emissores de ràdio d'ona curta i per satèl·lit, com el sistema de posicionament global (GPS) dels Estats Units o aviat el sistema europeu Galileu. Els receptors permeten d'obtenir el temps amb precisions de 0,1 a 10 ms.

Amb els rellotges atòmics...

... s'han descobert fluctuacions de 10^{-3} s en el període de rotació de la Terra, degudes a canvis de les marees, els corrents marins i atmosfèrics, fenòmens com "el Niño", els corrents de convecció del nucli terrestre, les erupcions volcàniques i els terratrèmols.

L'any 2000 es va arribar a precisions de 10^{-14} amb rellotges atòmics de rubidi refrigerats fins a prop de 0 K amb làser, i es pensa aconseguir 10^{-16}: una incertesa d'1 segon en mil milions d'anys, amb la mateixa tècnica de microgravetat a bord de l'Estació Espacial Internacional (ISS).

La conseqüència és que en una xarxa de computadors cada computador té un rellotge propi, amb un valor temporal que varia d'una manera diferent i que necessita un ajust continuat perquè tots els computadors tinguin un temps aproximadament igual. S'ha de tenir en compte que el rellotge d'un computador es pot accelerar o alentir però no pot ni retrocedir ni fer salts, ja que es podrien produir problemes (si un programa s'executa a una hora concreta, en retrocedir el rellotge el programa es podria executar dues vegades i, si el rellotge fa un salt, es podria no executar).

2.2. Com s'ha d'ajustar el rellotge?

Com que cada rellotge dóna marques de temps a velocitats lleugerament diferents, diem que tenen deriva: velocitat de canvi respecte al rellotge ideal per unitat de temps. I com que aquesta deriva pot canviar amb la temperatura, la humitat, etc., és necessari sincronitzar el rellotge amb una font de referència o, en cas que no n'hi hagi, amb altres computadors propers.

Tot intercanviant missatges, els rellotges de diverses màquines es poden coordinar entre si o amb una màquina de referència: s'envien missatges demanant l'hora. Tenint en compte el temps d'anada i tornada del missatge, es pot ajustar el rellotge local, sense fer salts, accelerant-lo o alentint-lo durant un període de temps (fer-lo progressar més a poc a poc però sense fer-lo retrocedir) fins que incorpori l'ajust.

Això serveix perquè puguin funcionar correctament algunes aplicacions que necessiten que els rellotges dels computadors que participen estiguin sincronitzats. Per exemple, si un servidor web amb l'hora retardada modifica un fitxer HTML, el canvi podria passar desapercebut per al client que ha llegit la pàgina web i al cap d'una estona torna a visitar-la.

Si H(t) és l'hora que indica el rellotge d'un computador i s'hi ha d'afegir un ajust de δ(t) segons per arribar al valor desitjable S(t) = H(t) + δ(t).

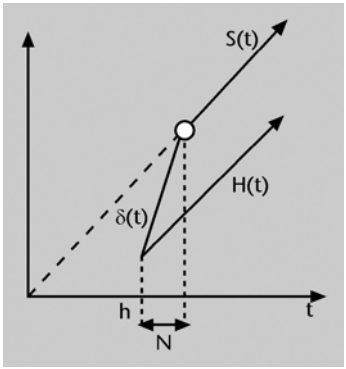


Figura 4. Ajust del rellotge progressiu en N passos

δ(t), com es pot veure en la gràfica, és una funció lineal del rellotge del computador: $\delta(t) = a \cdot H(t) + b$

Per tant, $S(t) = (1 + a) \cdot H(t) + b$

Si $S(t)=T_{despl}$ quan $H(t) = h$ en $t = T_{real}$, i si s'ha d'ajustar el rellotge en N passos de rellotge,

$$T_{despl} = (1 + a)h + b,\ T_{real} + N = (1 + a)(h + N) + b$$

Per tant, els valors per a ajustar el rellotge del computador una diferència $T_{real} - T_{despl}$ durant N cicles de rellotge són: $a = (T_{real} - T_{despl})/N$, $b = T_{despl} - (1 + a)h$

Mètode Christian per a sincronitzar rellotges intercanviant missatges (1989):

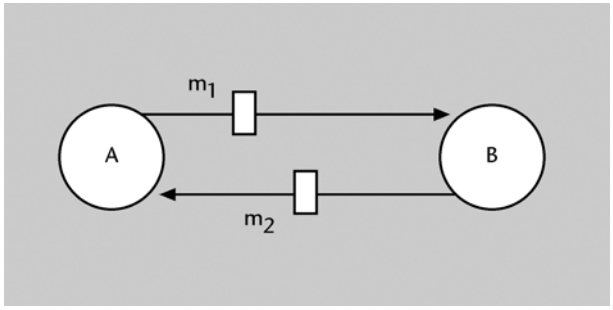


Figura 5. Intercanvi de missatges per a demanar marca de temps a B

A envia un missatge m_1 demanant una referència de temps a B; la resposta arriba en T_{IV}(temps d'anada i tornada) i m_2 conté el valor de temps t_B que tenia B quan ha enviat el missatge, T_{TRANS} enrere. $T_{TRANS} = mín. + x$; amb $x \geq 0$ *mín* pot arribar a estimar-se a base de mesures històriques, idealment quan la xarxa no tingui trànsit, com el valor mínim de T_{TRANS}.

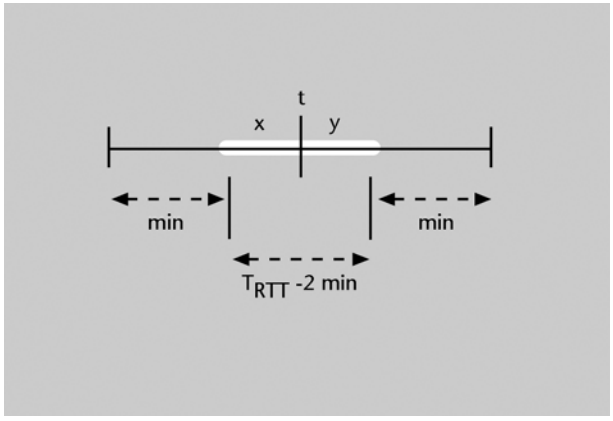


Figura 6. Marges de variació de la mesura de temps segons el retard

Es pot estimar en A que l'hora actual segons B és $t_A = t_B + T_{IV}/2$. Com més petit és T_{TRANS}, millor és l'estimació, ja que $T_{TRANS} = (mín + x) + (mín + y)$ i, com més petit és T_{TRANS}, x i y tendeixen a ser més insignificants.

La precisió és: $\pm(T_{RTT}/2\text{-mín})$

Per exemple en una xarxa local en què el temps de resposta variï entre 1-10 ms es pot aconseguir que el desajust entre ells sigui de pocs mil·lisegons i que la deriva del conjunt respecte a una referència externa sigui de l'ordre de 10^{-5}.

En realitat, si només hi ha un servidor pot ser problemàtic en cas de fallades, per això se solen usar mecanismes de sincronització amb diverses referències de temps; així és com es fa en els algorismes del protocol de temps en xarxa (NTP, *network time protocol*) o Berkeley (Unix BSD).

En l'algorisme de Berkeley una màquina pregunta a d'altres la seva hora i calcula l'hora local descomptant la propagació per la xarxa com fa l'algorisme de Christian. Calcula una mitjana amb tolerància de fallades (eliminant la influència de mesures errònies) i envia a cada màquina del conjunt l'ajust de rellotge que necessita.

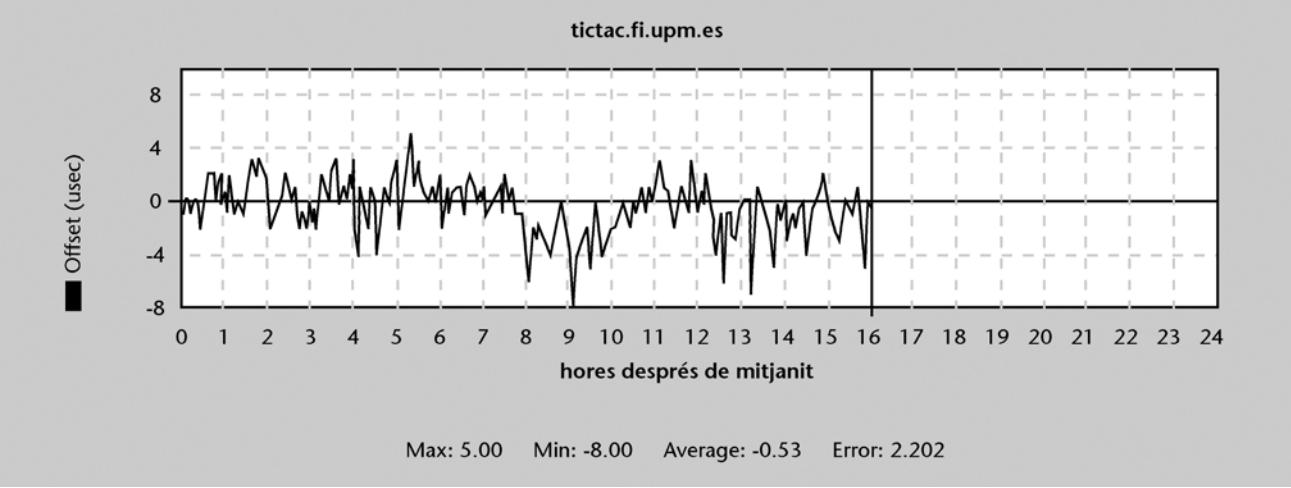
Sincronització de temps

A la xarxa acadèmica RedIRIS hi ha un grup de treball sobre temps. És interessant visitar la seva web: http://www.rediris.es/gtiris-ntp/drafts/

El protocol de temps en xarxa NTP (RFC 1305, RFC 2030) permet que una màquina pugui ajustar la seva hora per mitjà d'Internet. Les màquines estan organitzades d'una manera jeràrquica i tenen tres modes d'operació:

- *Multicast* o difusió selectiva: el servidor difon periòdicament informació de temps.
- *Procedure-call* o invocació: similar al mètode de Christian.
- Simètric: es manté una associació entre servidors que intercanvien informació de temps.

L'ajust de rellotges entre màquines d'una xarxa permet que la informació de temps sigui acceptada com a vàlida per les màquines que estan sincronitzades. Quan, per exemple, es comparteix un disc a la xarxa i les màquines que el comparteixen tenen els rellotges desajustats, es poden produir situacions estranyes si, en modificar un fitxer en una màquina amb el rellotge retardat, les altres màquines observen que el fitxer aparentment ha retrocedit en el temps.



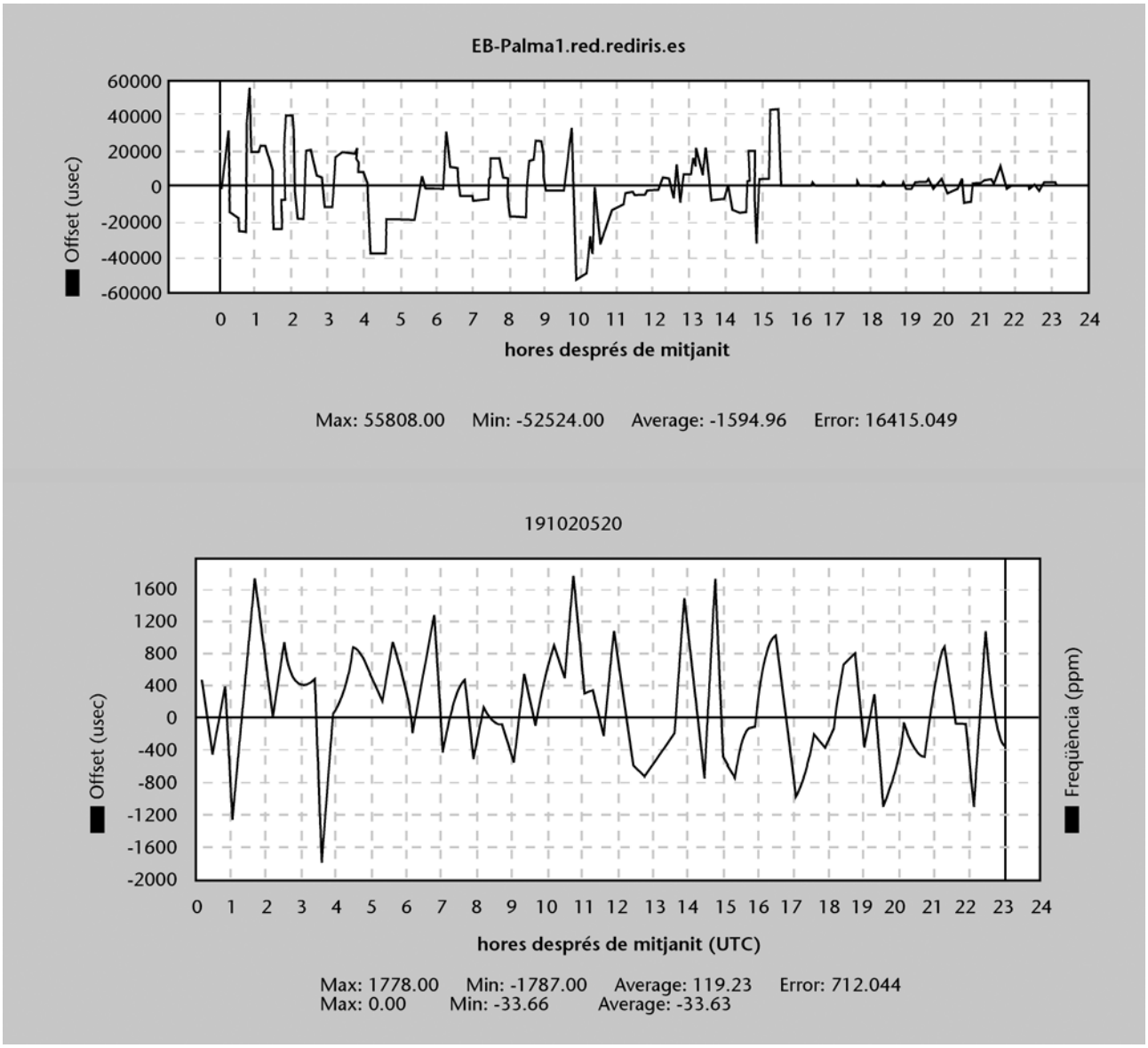


Figura 7. Variació de diversos servidors NTP en microsegons al llarg d'un dia (20/5/2002).
– "stratum 1": tictac.fi.upm.es, Sincronitza amb el GPS
– "stratum 2": eb-palma1.red.rediris.es, sincronitzat amb servidors "stratum 1" per la xarxa.
– "stratum 3": hora.usc.es, sincronitzat per la xarxa amb 3 servidors de "stratum 2".

2.3. Rellotges lògics

La teoria de la relativitat mostra que la velocitat de la llum és constant per a qualsevol observador i que la percepció del temps que passa és local. El pas del temps entre dos esdeveniments des de la Terra serà diferent que des de dins d'una astronau, especialment si hi ha acceleració: no hi ha un temps físic absolut al qual recórrer per a mesurar intervals de temps.

Per tant, en lloc d'usar l'hora (el rellotge físic) per a determinar l'ordre relatiu al succés de dos esdeveniments, Leslie Lamport, el 1978, proposa els rellotges lògics. També proposa de distingir entre relació de precedència (*happened-before* o causalitat potencial: $\rightarrow$) i concurrència ($||$) entre esdeveniments.

En el cas d'un sistema distribuït...

... pot no haver-hi un rellotge per marcar esdeveniments amb prou precisió perquè ens permeti saber en quin ordre succeïren certs esdeveniments, o si van ocórrer simultàniament. Un missatge enviat a diversos destinataris segurament mai no arribarà a tots alhora. Fins i tot en una mateixa màquina el sistema operatiu i la gestió de memòria, el disc i els processos, poden introduir retards inesperats.

Un rellotge lògic és un comptador d'esdeveniments (un nombre natural) que sempre augmenta i que no té relació amb el rellotge físic. Si un sistema distribuït està format per processos separats i cada procés P_i té un rellotge lògic L_i que s'usa per a marcar el temps (virtual) en què s'ha produït un esdeveniment: si l'esdeveniment e succeeix en el procés P_i, $L_i(a)$ és el valor del rellotge lògic per a aquest esdeveniment.

Observació causal

El problema no és nou:

"Quan un espectador observa un batalló fent exercicis des de certa distancia, veu com les persones corren abans de sentir l'ordre del comandament, però pel seu coneixement de les relacions causals sap que els moviments són el resultat de l'ordre i que, objectivament, l'última ha d'haver precedit la primera."

Christoph von Sigwart (1830-1904) *Logic*, 1889.

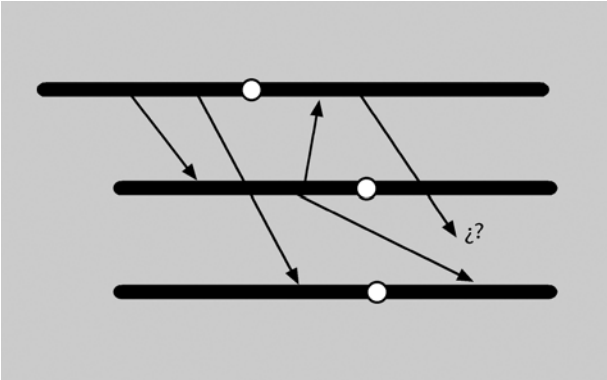


Figura 8. Diagrama d'una computació distribuïda.
Cada línia és un procés que envia o rep missatges en forma de fletxes. La inclinació indica la velocitat amb què viatja el missatge que pot variar per a cada un en funció de la càrrega de la xarxa i les màquines.
El missatge entre ¿? pot ser conseqüència dels anteriors: té una possible relació d'ordre amb els esdeveniments de tramesa o recepció de missatges en aquest procés. Hi observeu algun problema de causalitat?

Quan s'envia un missatge, s'inclou el valor de rellotge Lamport d'aquest procés. Això permet que en cada procés se sàpiga el valor del rellotge que hi havia quan es va enviar el missatge en el procés d'origen.

L'algorisme és el següent:

- L_i s'incrementa en cada esdeveniment en P_i: $L_i := L_i + 1$
- Quan un procés P_j rep un missatge que inclou el rellotge (m,t) es calcula $L_j := màx(L_j, t)$ i després aplica el pas anterior per a marcar la recepció del missatge.

Es pot veure que si $e \rightarrow e' \Rightarrow L(e) < L(e')$

Les relacions importants són les següents:

- Precedència (*happened-before* o $\rightarrow$) o causalitat potencial:

  - Si dos esdeveniments e i e' es donen en el mateix procés P_i, l'ordre és clar per a tot el sistema: $e \rightarrow e'$.
  - La tramesa d'un missatge succeeix abans que la recepció: tramesa(m) $\rightarrow$ recepció(m).
  - Si e, e' i e'' són esdeveniments que $e \rightarrow e'$ i $e' \rightarrow e''$, llavors: $e \rightarrow e''$.

- Concurrència (||) entre esdeveniments: quan dos esdeveniments no estan relacionats per $\rightarrow$ (no hi ha una relació causal).

Resulta que l'ordenació causal d'esdeveniments captura en molts casos la informació essencial per a descriure una execució.

L'inconvenient és que hi ha un rellotge Lamport per a cada procés i comparant el seu valor en dos missatges que ens arriben no podem concloure si un precedeix l'altre o si són concurrents. Per a això es van inventar els rellotges vectorials.

2.4. Rellotges vectorials

Un rellotge vectorial reflecteix l'evolució de tot el sistema, no solament d'un procés. La seva dimensió, el nombre d'elements del vector, coincideix amb el nombre de processos del sistema distribuït. D'aquesta manera, cada procés dóna compte del seu punt de vista de l'estat global: tot el que sap dels altres processos i d'ell mateix gràcies als missatges que rep.

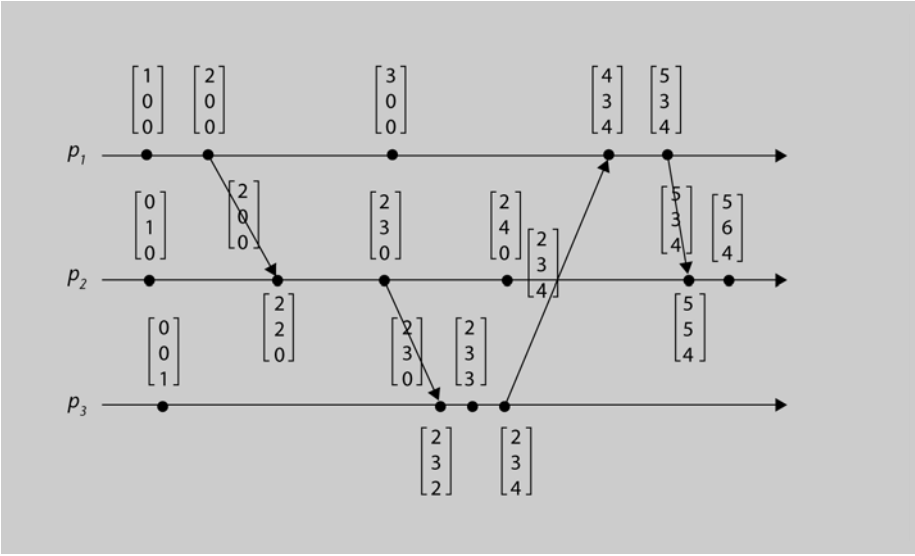


Figura 9. Evolució del vector temps en un sistema de dimensió 3.

En un sistema amb n processos, cada procés P_i manté un vector $V_i[1..n]$ amb la seva vista de tot el sistema segons la informació rebuda a través de missatges: la història causal. Just abans que es produeixi un esdeveniment en un procés, el seu propi component s'incrementa:

$$V_i[i] = V_i[i] + 1.$$

Quan s'envia un missatge, el vector viatja amb ell: (m, $V[\,]$).

Quan es rep un missatge, primer es fonen el vector que ve en el missatge amb el vector local i pren el valor més alt de cada component:

$$\text{Per a } k = 1..n : V_i[k] = màx(V_i[k], V[k])$$

Després s'incrementa el component propi i finalment es lliura el missatge.

Els rellotges vectorials es poden comparar entre ells, element a element. Així:

$V_h = V_k$ si en cada element s'esdevé que $Vh[x] \leq Vk[x]$
$V_h < V_k$ si en cada element s'esdevé que $Vh[x] \leq Vk[x]$ i algun $Vk[x] < Vk[x]$
$Vh \parallel Vk$ si ni $(V_h < V_k)$ ni $(V_k < V_h)$

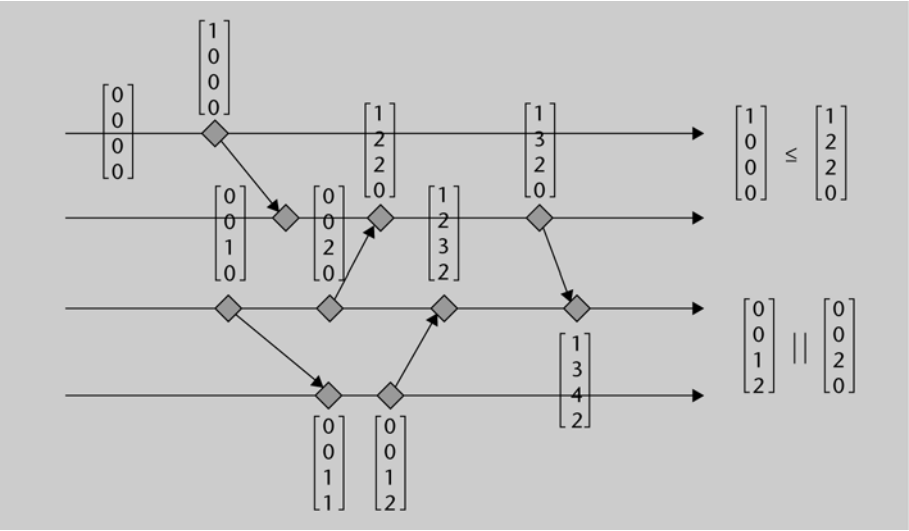


Figura 10. Rellotges vectorials: les línies de punts indiquen la frontera de la història causal.

Comparar vectors és com comparar la història causal (el que sabem de la resta), per tant la relació $V_h \leq V_k$ entre vectors és equivalent a la relació $e_h \rightarrow e_k$ entre esdeveniments i també $V_h \parallel V_k$ entre vectors és equivalent a la relació $e_h \parallel e_k$. És a dir, que la propietat rellevant dels rellotges vectorials, que no tenen els rellotges de Lamport, és que comparant vectors podem saber si dos esdeveniments estan relacionats causalment o són concurrents.

A "Logical time: capturing causality in distributed systems" Raynal i Singhal es plantegen els rellotges matricials: vectors de vectors, que funcionen de manera similar. El rellotge matricial permet conèixer el vector de cada procés i, per tant, el que coneix cada procés de tot el sistema. Per tant, permet determinar quins esdeveniments ja són coneguts o rebuts per tots els processos, esdeveniments que no tornaran a circular o a reclamar-se i es poden oblidar.

Bibliografia complementària

M. Raynal; M. Singhal (1996, febrer). "Logical time: capturing causality in distributed systems". *Computer* (vol. 29, núm. 2, pàg. 49-56).

3. Exclusió mútua

Els processos distribuïts sovint han de coordinar-se. Si aquesta coordinació implica compartir recursos, caldrà algun mecanisme per a evitar interferències i assegurar la consistència en l'accés a aquests recursos. En aquest apartat veurem alguns dels algorismes distribuïts més populars.

Lectura recomanada

En l'article següent trobareu una comparació entre diferents algorismes per a aconseguir exclusió mútua:

P. Saxena; J. Rai (2003, maig). "A survey of permission-based distributed mutual exclusion algorithms". *Computer Standards and Interfaces* (vol. 25, núm. 2, pàg. 159-181).

3.1. Algorisme centralitzat

La manera més senzilla d'aconseguir exclusió mútua és que un procés actuï de coordinador i doni permisos d'accés a una secció crítica. Per a accedir a una secció crítica, un procés envia un missatge de petició al coordinador i espera una resposta d'aquest. Si en aquest moment no hi ha cap procés accedint a la secció crítica, el coordinador respon immediatament i el procés pot entrar-hi. En cas que hi hagi un procés a la secció crítica, el coordinador no contesta i encua la petició. Quan el procés que està a la secció crítica en surt, s'envia un missatge al coordinador informant-lo. D'aquesta manera el coordinador ja pot donar accés a la secció crítica al procés següent.

Si la cua de processos esperant no està buida, el coordinador escull l'entrada que fa més temps que està a la cua, la treu i respon a aquest procés (recordem que el procés havia fet una petició i estava bloquejat esperant resposta).

En aquesta solució, el coordinador és un punt centralitzat de fallada. Si aquest falla, tot el sistema deixa de funcionar. Un procés bloquejat en la petició d'accés a la secció crítica no és capaç de distingir entre una fallada del coordinador i estar esperant que aquest li doni accés a la secció crítica. A més a més, en un entorn de gran escala, un únic coordinador pot ser un coll d'ampolla. Tot i això, els beneficis provinents de la simplicitat del mecanisme sovint pesen més que aquests possibles inconvenients.

3.2. Algorisme descentralitzat

Tenir un únic coordinador és una aproximació molt pobre. Una manera de gestionar l'accés a un recurs de manera descentralitzada és usar un algorisme basat en votacions que s'executi en un sistema basat en una DHT. La idea és la següent: se suposa que cada recurs està reproduït *n* vegades. Cada reproducció té el seu coordinador que controla l'accés de processos concurrents.

Lectura recomanada

S. D. Lin; Q. Lian; M. Chen; Z. Zhang (2004). "A Practical Ditributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems". Proc. Third International Workshop on Peer-to-Peer Sytems. *Lecture Notes in Computer Sciences* (vol. 3279, pàg. 11-21).

Quan un procés vol accedir a un recurs només li cal aconseguir que una majoria *m* de coordinadors hi estigui d'acord. Aquesta majoria *m* ha de ser supe-

rior a *n*/2, és a dir, necessita que més de la meitat de reproduccions votin a favor que hi pugui accedir. Cadascun dels coordinadors, en rebre la petició d'accés al recurs, contesta al procés que fa la petició amb un vot perquè hi pugui accedir o amb una negació d'accés si ja ha donat permís a algun altre procés per accedir al recurs.

Aquesta solució és menys vulnerable a fallades que la centralitzada, ja que suporta fallades puntuals de coordinadors. El problema es presenta quan el procés coordinador es recupera de la fallada. Durant la fallada, el procés haurà perdut la informació associada als accessos que havia concedit i en recuperarse pot donar de manera incorrecta accés al recurs a un altre procés. En l'article esmentat han arribat a la conclusió que si els processos que fallen es recuperen ràpidament la probabilitat que *k* processos entre els *m* coordinadors que havien participat en una votació reiniciïn és molt petita i, en qualsevol cas, molt menor que la disponibilitat del recurs a protegir.

La implementació que proposen es basa en una DHT on cada recurs està reproduït *n* vegades. Si el recurs està identificat de manera única per *nom* i suposem que cada reproducció del recurs s'identifica per *nom-i* on *i* és la i-èsima reproducció del recurs, llavors cada reproducció del recurs es pot identificar de manera única usant una funció resum o *hash*. D'aquesta manera, donat un nom de recurs, cada procés pot generar les *n* claus, i cercar el procés responsable de la reproducció.

Si es denega l'accés al recurs (és a dir, la petició obté menys de *m* vots), el procés que fa la petició s'ha d'esperar un temps aleatori i tornar a intentar-ho. En situacions en què molts processos volen accedir a un mateix recurs, aquest sistema perd rendiment molt ràpidament, ja que cap procés pot aconseguir suficients vots per a accedir al recurs i es deixa el recurs sense utilitzar. En l'article esmentat a l'inici de l'apartat, els autors proposen una solució per aquest problema.

3.3. Algorisme basat en un anell

Una manera senzilla de gestionar l'exclusió mútua entre un conjunt de processos sense que cap d'ells faci una funció especial o sense requerir cap procés addicional és organitzar els processos com si formessin un anell. La idea que hi ha al darrere d'aquesta solució és que es pot accedir a la secció crítica quan es disposa d'un testimoni, que és el que dóna el dret d'accés. Aquest testimoni és un missatge que es va passant d'un procés a un altre segons un ordre establert, que es construeix tenint un anell amb tantes posicions com processos s'hagin de coordinar. Cada procés ocupa una posició en aquest anell virtual i coneix quin és el següent procés seguint aquest ordre. No cal que la topologia d'aquest anell tingui relació amb la topologia d'interconnexió dels processos que en formen part.

Si un procés no ha d'accedir a la secció crítica quan rep el testimoni, immediatament passa el testimoni al següent procés segons l'ordre de l'anell. Un procés que necessiti el testimoni s'espera fins que el rebi i, un cop el té, el reté fins que acabi les operacions que ha de realitzar. Quan vol sortir de la secció crítica, envia el testimoni al seu veí.

Un dels problemes d'aquest algorisme és si es perd el testimoni. De fet, és difícil detectar que s'ha perdut el testimoni, ja que la quantitat de temps que passa des que es veu el testimoni fins que es torna a veure és totalment imprevisible. Que faci molt de temps que no es veu el testimoni no vol dir que aquest s'hagi perdut, ja que pot ser que algun dels processos l'estigui usant.

Un segon problema es pot presentar si falla un procés, però aquest és més senzill de solucionar. Es pot demanar a un procés que quan rebi el testimoni en confirmi la seva recepció. Es detectarà que un procés ha fallat quan intenti donar-li el testimoni i aquest falli. En aquest moment es pot eliminar del grup el procés mort i el procés que té el testimoni l'envia al següent procés en l'ordre de l'anell, o al següent, si cal. Per suposat, caldrà que tots els processos refacin la configuració de l'anell.

Un altre inconvenient d'aquest algorisme és que consumeix amplada de banda contínuament, excepte quan algú està dins d'una secció crítica.

3.4. Algorisme distribuït

Quan un procés *Pi* vol entrar a una secció crítica, genera una nova marca de temps Ts_{Pi}, i envia un missatge de petició (*Pi*, Ts_{Pi}) a tota la resta de processos del sistema.

Lectura recomanada

G. Ricart; A. K. Agrawala (1981). "An Optimal Algorithm for Mutual Exclusion in Computer Network Computing". *Communications of the ACM* (vol. 24, núm. 1, pàg. 9-17).

Quan un procés *Pj* rep un missatge de petició, pot contestar immediatament o endarrerir l'enviament de la resposta fins més endavant.

Quan el procés *Pi* rep el missatge de resposta de la resta de processos del sistema, pot entrar a la secció crítica.

Després de sortir de la secció crítica, el procés envia un missatge de resposta a tots els que tenen sol·licituds endarrerides.

La decisió de si el procés *Pj* respon immediatament a un missatge de sol·licitud (*Pi*, Ts_{Pi}) o endarrereix la resposta es basa en tres factors:

- Si *Pj* està en una secció crítica, llavors endarrereix la resposta a *Pi*.
- Si *Pj* no vol entrar a una secció crítica, envia immediatament una resposta a Pi.
- Si *Pj* vol entrar a una secció crítica però encara no hi ha entrat, compara la marca de temps de la seva sol·licitud (Ts_{Pj}) amb la marca de temps Ts_{Pi}.

a) Si la marca de temps de la seva sol·licitud és major que Ts_{Pi}, llavors envia immediatament un missatge de resposta a Pi (Pi ho havia demanat abans). Si Ts_{Pj} és igual a Ts_{Pi}, llavors la petició s'ordena segons els identificadors dels processos. Això garanteix que hi hagi un ordre total.

b) Altrament, endarrereix la resposta.

D'aquest comportament es dedueix el següent:

- El sistema està lliure de bloquejos indefinits.

- Cap procés estarà esperant entrar a la secció crítica infinitament. L'ordenació per marques de temps assegura que els processos accedeixen a la secció crítica d'acord amb un ordre.

- El nombre de missatges per a entrar a la secció crítica és de $2(N – 1)$, on N és el nombre de processos. Són $N – 1$ missatges per a fer la petició i $N – 1$ per a les respostes. Si es fes servir una primitiva de *multicast*, el nombre de missatges seria N.

Alguns inconvenients d'aquest algorisme són que força a utilitzar una primitiva de comunicació en grups o força que cada procés conegui la identitat de la resta de processos, cosa que fa més complex afegir o treure processos. A més a més, quan un procés falla el sistema no funciona, cosa que obliga a utilitzar algun sistema de monitorització. Això fa que aquest algorisme funcioni bé en grups petits i estables de processos que cooperen entre si.

Sapigueu que s'han proposat variacions de l'algorisme que en milloren el rendiment, però no les tractarem perquè s'escapen una mica dels objectius de l'assignatura.

Per acabar, val la pena dir que aquest algorisme és més lent, més complicat, més costós i menys robust que la solució centralitzada. Tot i això, està bé veure que es pot construir una solució distribuïda.

3.5. Comparació d'algorismes

La taula següent presenta una comparació entre els algorismes d'exclusió mútua vistos.

Algorisme	Missatges per a entrar/sortir	Retard abans d'entrar (en temps de missatges)	Problemes
Centralitzat	3	2	Fallada coordinador
Descentralitzat	3mk, k = 1, 2, ...	2mk, k = 1, 2, ...	Inanició, baixa eficiència
En anell	1 a	0 a n – 1	Pèrdua del testimoni, fallada procés
Distribuït	2(n – 1)	2(n – 1)	Fallada qualsevol procés

En el cas de l'algorisme centralitzat, necessita tres missatges per a entrar i sortir: petició d'entrada, autorització d'entrar i alliberament per a sortir. El descentralitzat necessita tres missatges per a cadascun dels coordinadors. La k correspon al nombre d'intents que ha de fer fins que aconsegueix entrar. En l'algorisme en anell, si els processos volen entrar constantment a la secció crítica, cada missatge correspondrà a una entrada i sortida. Però si no hi ha ningú interessat a entrar a la secció crítica, el testimoni pot estar circulant per l'anell durant hores, cosa que fa que el nombre de missatges per entrar a la secció crítica sigui il·limitat. Finalment, l'algorisme distribuït necessita $n - 1$ missatges per a la petició d'entrada a la secció crítica i $n - 1$ missatges per a les respostes.

El retard (en nombre de missatges necessaris) des que un procés intenta accedir a una secció crítica fins que ho aconsegueix és de 2 per al cas de l'algorisme centralitzat; $2mk$ per al descentralitzat; $2(n - 1)$ per al distribuït; i un nombre de missatges entre 0 (el testimoni tot just ha arribat) i $n - 1$ (el testimoni tot just acaba de marxar) en el cas de l'algorisme d'anell.

Finalment, tots els algorismes presentats, excepte el descentralitzat, pateixen molt quan hi ha fallades. En l'avaluació respecte a la tolerància a fallades, cal considerar què passa quan els missatges es perden i què passa quan un procés falla. Cal afegir mesures especials i complexitat addicional per tal d'evitar que una fallada faci aturar tot el sistema. Si les fallades ocorren rarament aquests mecanismes poden anar bé, sinó caldrà tenir en compte mecanismes que permetin als processos coordinar-se en presència de fallades. L'algorisme descentralitzat és menys sensible a fallades, però a canvi pot tenir problemes d'inanició i d'eficiència.

4. Algorismes d'elecció

Molts algorismes distribuïts necessiten que algun dels processos actuï com a coordinador, iniciador o desenvolupi algun rol especial. En general, tant se val quin procés ho faci, però algun ho ha de fer. Tot seguit veurem tres algorismes d'elecció de coordinador.

4.1. L'algorisme de Bully

L'algorisme de Bully serveix perquè un conjunt de processos en triï un que faci de coordinador. Un procés comença l'elecció de coordinador en les situacions següents:

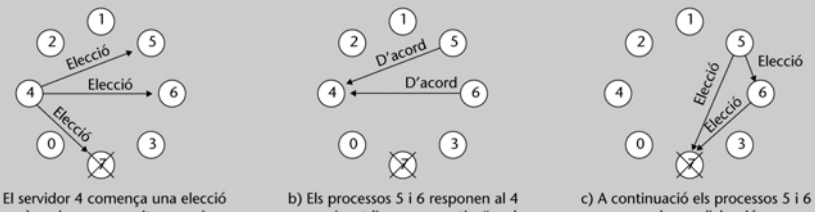
- acaba d'entrar en el sistema;
- el coordinador actual no respon;
- ha rebut un missatge d'elecció de coordinador d'un altre procés.

Lectura recomanada

H. Garcia-Molina (1982, gener). "Election in a Distributed Computing System". *IEEE Trans. Comp.* (vol. 31, núm. 1, pàg. 48-59).

L'elecció consisteix a enviar un missatge d'elecció a la resta de processos amb identificador superior al seu. (Es pot optimitzar deixant d'enviar elecció quan algun dels servidors amb identificador més gran respon.) Si algun servidor respon, l'originador de l'elecció no fa res més.

El coordinador serà el procés que no obtingui resposta als seus missatges d'elecció adreçats a processos amb identificador superior al seu. Aquest procés ha d'avisar els altres processos del seu identificador perquè sàpiguen que ell és el nou coordinador.

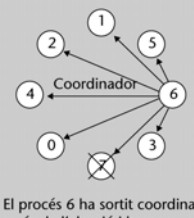


a) El servidor 4 comença una elecció perquè no ha aconseguit comunicar-se amb el procés coordinador (7). Per tant, envia *elecció* a tots els processos amb identificador superior al seu.

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

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

d) El procés 6 indica al 5 que no continuï, perquè ell té un identificador superior.



e) El procés 6 ha sortit coordinador després de l'elecció i ho comunica als altres processos.

4.2. Algorisme anell

Consisteix a usar un anell. Suposem que cada procés coneix el seu successor segons algun ordre donat. Quan algun procés s'adona que el coordinador no està funcionant, envia un missatge *elecció* al seu successor. Aquest missatge conté la informació del seu identificador de procés. Si el successor no respon, el procés que està intentant fer l'enviament, ho intenta amb el successor, o el següent, fins que troba un procés que estigui funcionant. Quan un node rep un missatge d'*elecció*, afegeix el seu identificador a la llista d'identificadors i passa el missatge al seu successor.

Quan el missatge torna a arribar a l'originador del procés d'elecció, s'envia un nou missatge *coordinador* on s'informa tothom de qui és el nou coordinador (per exemple, el membre de la llista amb l'identificador més gran) i qui són els membres del nou anell. Quan aquest missatge acaba la volta, s'elimina de l'anell i tots els processos ja poden tornar a treballar amb el nou coordinador.

L'algorisme funciona encara que hi hagi més d'un procés que iniciï el procés d'elecció; en aquest cas el que passarà és que hi haurà més d'un missatge *elecció*. Quan aquests missatges acabin la volta, arribaran al seu originador i en tots els casos, es convertiran en un missatge *coordinador* amb la mateixa llista de membres i en el mateix ordre. Quan hagin fet tota la volta s'eliminaran de l'anell i tots els processos tindran el mateix coordinador. Aquests missatges de més no fan cap mena de mal a part del fet de consumir una mica d'amplada de banda.

4.3. Elecció en entorns sense fil

Les assumpcions que hem fet en els algorismes anteriors no les podríem fer en un entorn sense fil. Per exemple, suposen que l'enviament de missatges és fiable i que la topologia de la xarxa no canvia. A continuació explicarem un algorisme d'elecció que funciona en una xarxa *ad hoc*. Aquest algorisme pot gestionar nodes que fallen i particions a la xarxa. Una propietat important d'aquesta solució és que escull el millor líder, no un a l'atzar com en els dos anteriors que hem vist.

Xarxa *ad hoc*

Per simplificar, ens hem centrat en el fet que la xarxa és *ad hoc*. No hem tingut en compte que els nodes es poden moure.

Qualsevol node a la xarxa (l'anomenarem *originador*) pot iniciar una elecció de líder enviant un missatge *elecció* als seus veïns immediats (és a dir, els nodes que estan dins del seu rang). Quan un node rep un missatge d'*elecció* per primer cop, designa qui li ha enviat el missatge com a pare i a continuació envia un missatge d'*elecció* a tots els seus veïns immediats excepte al seu pare. Quan un node rep un missatge d'*elecció* d'algú diferent del seu pare, senzillament en confirma la recepció.

Quan un node *R* ha designat un altre node *Q* com a pare, envia el missatge d'*elecció* als seus veïns immediats (excloent-ne *Q*) i espera que arribin les confirmacions abans de confirmar el missatge d'*elecció* rebut del node *Q*.

Els nodes que ja hagin seleccionat un pare, contestaran a *R* molt ràpidament. Si tots els veïns ja tenen un pare, *R* és una fulla i respondrà a *Q* molt ràpidament. En fer-ho, també proporcionarà informació com la durada de la seva bateria i altres capacitats del recurs.

Aquesta informació permetrà a *Q* comparar les capacitats de *R* amb les d'altres nodes, i escollir el millor node perquè sigui el líder. Al seu torn, *Q* havia enviat el missatge d'elecció perquè l'havia rebut del seu pare *P*, que havia fet el mateix. Quan *Q* confirmi el missatge d'*elecció* rebut de *P*, li passarà el millor candidat a ser escollit. D'aquesta manera, l'originador arribarà a saber quin dels nodes és el millor per a ser escollit com a líder. Un cop triat, farà un *broadcast* d'aquesta informació a tots els nodes.

Quan s'iniciïn diverses eleccions, cada node decidirà apuntar-se només a una de les eleccions. Per a aconseguir-ho, cada elecció anirà etiquetada amb un identificador. Els nodes participaran només en l'elecció amb l'identificador més alt i aturaran qualsevol altra participació en altres eleccions.

Lectura recomanada

Podeu trobar els detalls de l'algorisme a:

S. Vasudevan; J. F. Kurose, D. F. Towsley (2004). "Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks". *Proc of the 12th IEEE International Conference Network Protocols. IEEE Computer Society Press* (pàg. 350-360).

5. Tolerància a fallades

Es podria esperar que la fiabilitat d'un sistema distribuït depengués de la fiabilitat dels seus components, però això no acostuma a ser així. Un sistema distribuït està format per components que interactuen. Aquesta interacció pot fer que falli el sistema quan un component falla, o que continuï funcionant sense una degradació excessiva malgrat la fallada d'algun component. D'aquesta manera, es poden construir sistemes amb capacitat de "supervivència".

Una definició de sistema distribuït...

… des d'un punt de vista cínic:

"You know you have one when the crash of a computer you've never heard of stop you from getting work done."

Leslie Lamport

L'objectiu de funcionament és no fallar quan falla un component, sinó funcionar amb components avariats o en manteniment (fallades parcials), de manera que el sistema "aguanti" més que els seus components.

Sobre la llançadora espacial

"El programa de bord s'executa en dues parelles de computadors principals, una parella porta el control mentre els seus càlculs simultanis coincideixin. En cas de desacord, el control passa a l'altra parella. Els quatre computadors principals executen programes idèntics.

Per a prevenir fallades catastròfiques en què les dues parelles cometen un error (per exemple, si el programa té una errada), la llançadora té un cinquè computador que està programat amb un codi diferent per programadors diferents d'una altra empresa, però usant les mateixes especificacions i el mateix compilador (HAL/S)".

Peter Neuman. *Computer Related Risks*. Addison-Wesley, 1995.

Un sistema falla quan no pot complir les seves promeses de servei i és conseqüència d'alguna fallada: una falta, deficiència o error.

Una forma tradicional per tal d'aconseguir tolerància a les fallades és mitjançant un maquinari repetit. Per exemple, la llançadora espacial de la NASA té moltes peces repetides (moltes de quadruplicades). Les decisions es prenen per consens, amb tres n'hi hauria prou per a detectar un error, amb quatre es permeten dos errors durant una missió. El computador principal està 4 vegades + 1 de recolzament = 5.

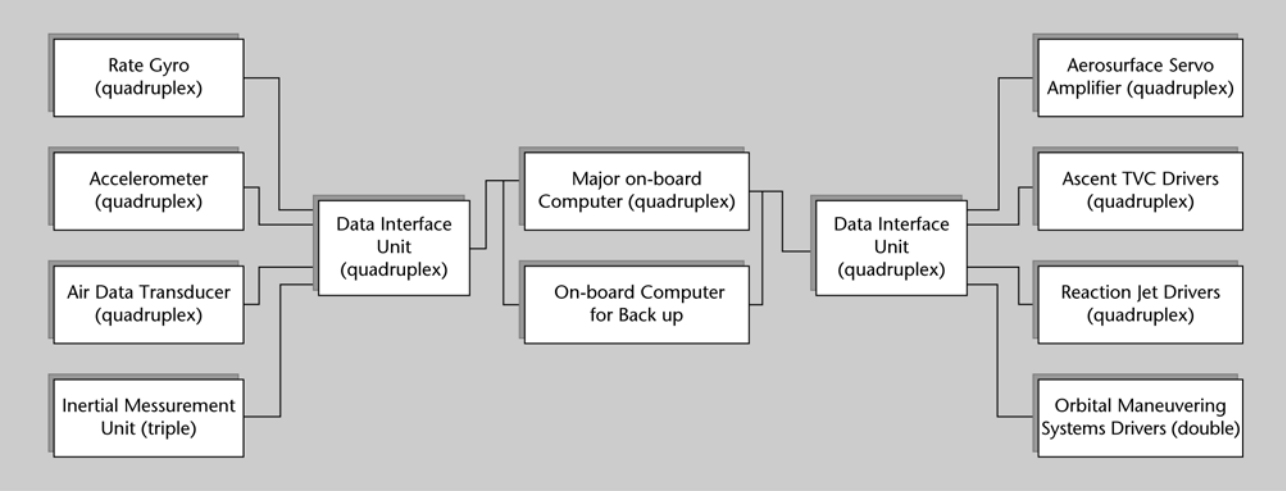


Figura 11. Esquema de la redundància del sistema de vol de la llançadora espacial de la NASA.

Tanmateix, és possible d'obtenir una tolerància a errors menys crítica a partir de maquinari "normal" i una capa de programes que ofereixi un model de programació i alguns serveis essencials per a programar aplicacions distribuïdes tolerants a fallades.

De fet, Internet és una xarxa que tolera fallades en els enllaços a causa de la seva organització redundant, també algunes aplicacions com la DNS i SMTP tenen mecanismes per a oferir la informació i el servei reproduït per a evitar que l'error d'una màquina deixi una organització sense servei.

La tolerància a fallades tracta de com es pot dependre del servei d'un sistema o comptar-hi.

Hi ha quatre aspectes importants que caracteritzen un sistema i el seu context:

- Disponibilitat – el sistema està llest per a l'ús immediat.
- Fiabilitat – el sistema pot funcionar de manera contínua sense fallada.
- Seguretat – si un sistema falla, no passa res greu.
- Mantenibilitat – quan falla un sistema, que es pugui reparar de manera fàcil i ràpida (i idealment, els usuaris no notin la fallada).

Per exemple, un sistema que falla durant 1 mseg cada minut és més disponible que un que falla durant un minut cada any, tanmateix el primer és menys fiable. També cal considerar la seguretat, ja que la fallada d'un servidor d'impressió ens deixa temporalment sense poder imprimir, mentre que la fallada d'un controlador d'una central nuclear o d'una sala d'operacions pot tenir conseqüències catastròfiques.

La fallada d'un sistema es pot descriure en termes de quan falla i de com falla. Respecte a quan passa hi ha tres categories:

- Fallada transitòria – es produeix una vegada i desapareix.
- Fallada intermitent – es produeix, desapareix i torna a aparèixer més tard.
- Fallada permanent – quan apareix ja no cessa.

Respecte a com falla, el comportament pot ser el següent:

- Fallada i parada: un procés funciona bé i de sobte s'atura. És el model de fallada més fàcil de detectar.

- Funcionament erroni: un procés funciona bé i de sobte comença a donar alguns resultats erronis. És molt difícil detectar la situació, i en un procés de decisió pot fer que el sistema cometi un error abans de verificar que està funcionant malament.

- Funcionament lent: el procés funciona bé, però comença a anar cada vegada més lent. Es pot deure al fet que el sistema operatiu està paginant, un procés està reintentant alguna operació o la xarxa està congestionada. Pot arribar a alentir la resta del sistema.

Una manera de simplificar els casos és disposar d'una capa de programari que proporcioni el model *fallada-parada*: vestir totes les fallades com si fossin parades. Per tant, es tracta de fer preguntes als processos per verificar que no estan funcionant erròniament, i si s'alenteixen o donen errors se'ls considera aturats i se'ls ignora.

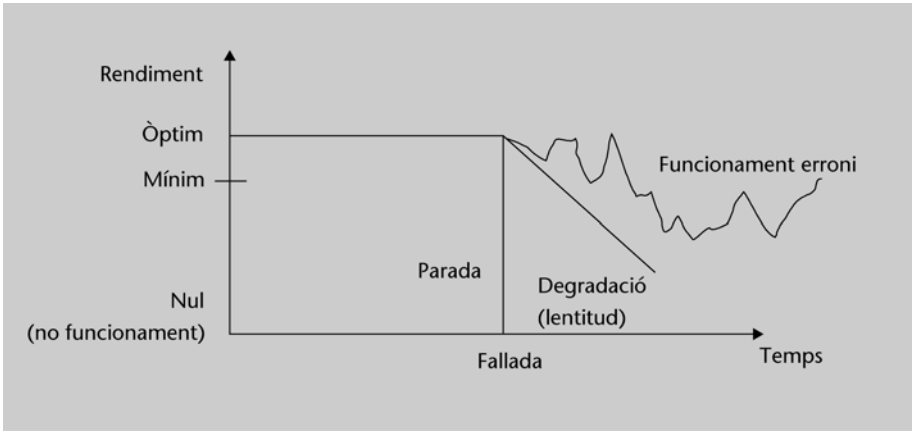


Figura 12. Rendiment en funció del temps. El model "desitjable" és fallada i parada, però no és l'únic i això complica el tractament de les fallades.

En un sistema en què els missatges o els participants poden fallar, per a aconseguir fiabilitat fa falta reproducció: components repetits; però si hi ha components repetits cal que aquests es posin d'acord per a realitzar una acció o compartir informació. Però aquest acord o consens no és possible si els missatges es poden perdre o els processos poden fallar.

Aquests problemes se solen explicar amb l'exemple de diversos generals amb els seus exèrcits en el camp de batalla. És el problema dels dos generals i el problema dels generals bizantins.

1) Impossibilitat de consens amb comunicació no fiable.

El problema dels dos generals explica que dos generals A i A' només poden vèncer si acorden atacar alhora l'exèrcit B. Per arribar a un acord han d'intercanviar missatges. Si els missatges es poden perdre pel camí (han de travessar l'exèrcit enemic) és impossible posar-se d'acord: un missatge amb una proposta d'hora d'atac s'ha de confirmar, però també s'ha de confirmar la confirmació. Així que un exèrcit mai no està segur que l'altre atacarà alhora.

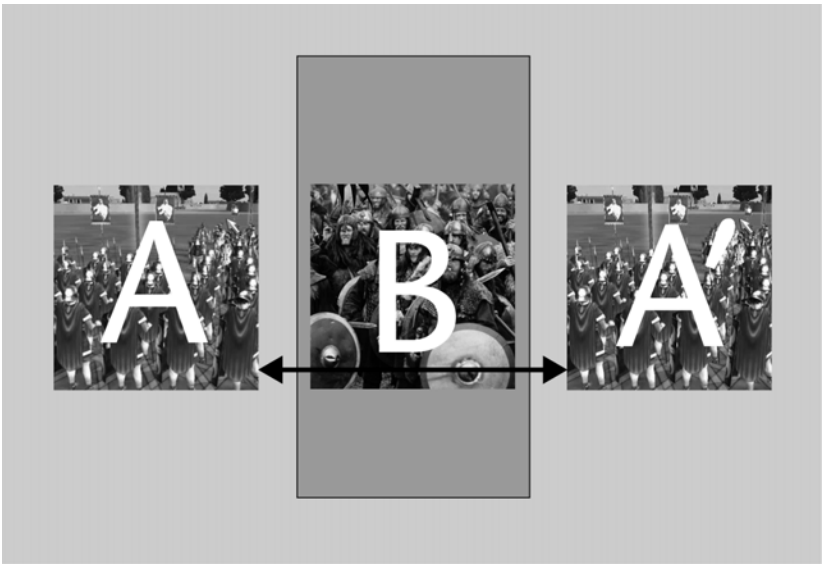


Figura 13. L'exèrcit B només pot ser derrotat si els exèrcits A i A' sumen les seves forces i ataquen alhora. Per a això han d'enviar un missatger que ha de travessar el campament contrari i pot fallar en el camí. Per exemple, A envia un missatger indicant l'hora de l'atac m (A, A', 12.30 h), però A' haurà de confirmar si vol que A estigui segur. Tanmateix, aquest procés de confirmació es repeteix indefinidament. És el problema d'assolir consens quan la comunicació no és fiable.

2) Quin és el nombre total de participants necessari per a arribar a un acord quan diversos participants poden fallar de manera arbitrària.

El problema dels generals bizantins descriu aquesta situació. Un general bizantí representa un procés o component que pot fallar de manera arbitrària: en el pitjor cas podem suposar que és maliciós i vol confondre la resta de generals. El nom ve de les traïcions entre els generals de l'imperi de Bizanci (l'actual Turquia i part dels Balcans).

Bibliografia complementària

Lectures sobre tolerància de fallades bizantines:

L. Lamport; R. Shostak; M. Pease (1982). "Byzantine Fault Tolerance". *ACM Transactions on Programming Languages and Systems*.

L. Lamport; R. Shostak; M. Pease (1982, juliol). "The Byzantine Generals Problem". *ACM Transactions on Programming Languages and Systems* (vol. 4, núm. 3, pàg. 382-401).

L'algorisme té tres fases:

a) Els generals anuncien el seu nombre de soldats (el seu identificador en aquest cas) en un missatge als altres membres del grup.

b) Cada general construeix un vector amb les dades rebudes en (a) i el seu propi nombre que envia als altres generals.

c) Cada general pot determinar qui és el traïdor mirant les majories a les columnes.

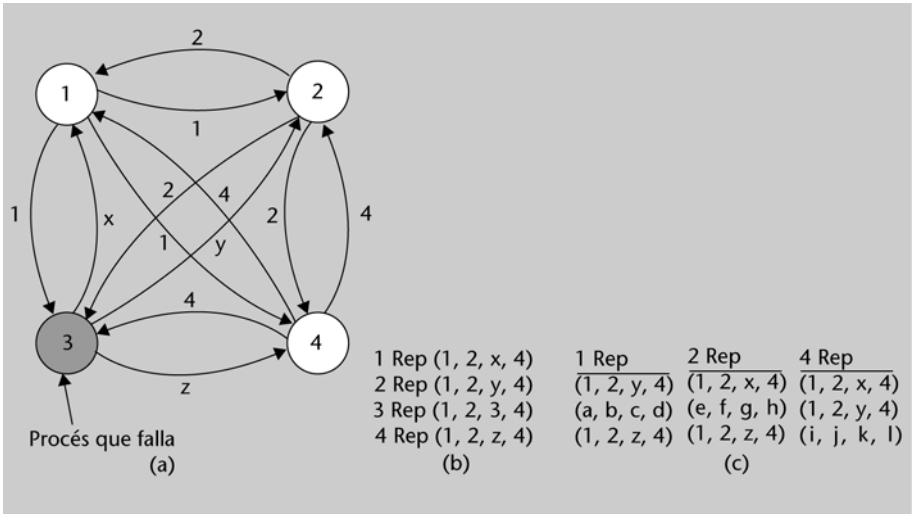


Figura 14. El problema dels generals bizantins amb tres generals lleials i un traïdor. Amb dos generals lleials no seria possible detectar el traïdor.

Seguint els detalls de l'algorisme, s'acaba concloent que perquè l'algorisme detecti els processos que tenen fallades arbitràries, més de dos terços dels processos han de funcionar correctament. És a dir, si hi ha M processos que fallen, calen 2 M + 1 processos correctes per a poder arribar a un consens.

5.1. Comunicació fiable en grup

Si per a aconseguir tolerància a fallades cal reproducció (diversos components), qualsevol comunicació s'ha de dirigir a aquest grup de components. La comunicació en grup (també anomenat *groupcast* o *multicast*) fiable és una idea simple però molt difícil de realitzar.

Un aspecte és com es poden enviar missatges a un grup de manera fiable per una xarxa. A les xarxes locals es pot aprofitar la capacitat del maquinari per a la tramesa de missatges alhora a diversos destinataris. Tanmateix, assegurar la fiabilitat en el lliurament dels missatges a tots els destinataris i al control de flux quan els destinataris són heterogenis és un problema. Quan els destinataris són a Internet, el problema resideix a mantenir un graf de connexions punt a punt entre les diferents subxarxes que participen en la comunicació i gestionar la diversitat de xarxes, terminals i problemes de congestió a la xarxa.

Un altre aspecte és com s'ha d'organitzar el grup de servidors. En el model primari-secundari (*primary-backup*) les peticions van al servidor primari i el primari les passa de manera síncrona als secundaris. Quan hi ha fallades o recuperacions, es decideix qui farà de primari.

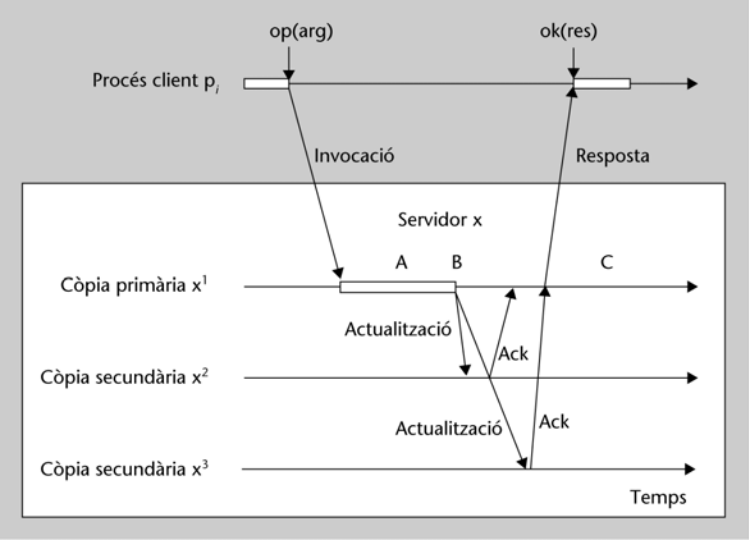


Figura 15. Model primari-secundari: el client es comunica només amb el primari. El primari es comunica al seu torn amb els secundaris.

En el model de reproducció activa, el client ha d'enviar el seu missatge a totes les còpies i aquestes simplement han de respondre.

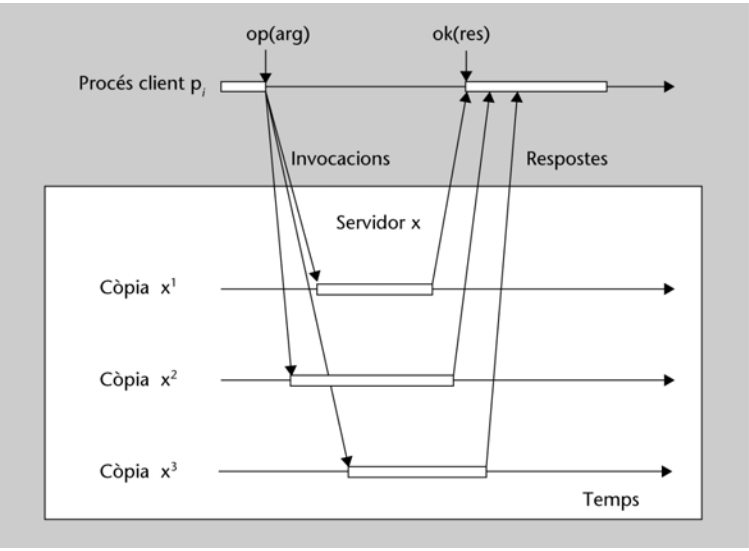


Figura 16. Model de reroducció activa: el client es comunica directament amb totes les còpies.

Bibliografia complementària

Mecanismes de replicació de programari per a tolerància a fallades:

R. Guerraoui; A. Schiper (1997). "Software-Based Replication for Fault Tolerance". *Computer* (vol. 30, núm. 4, pàg. 68-74). [http://dx.doi.org/10.1109/2.585156]

Els problemes sorgeixen quan es produeixen canvis d'estat (operacions d'escriptura), fallades o arriben al servidor operacions des de diversos clients. El lliurament de missatges al grup de servidors pot tenir diverses propietats.

S'ha tractat aquí la reproducció síncrona. Més endavant es descriuen els mecanismes de replicació asíncrons o optimistes.

Sobre els mecanismes de reproducció asíncrons o optimistes, vegeu el subapartat 7.5, "Reproducció optimista", d'aquest mòdul didàctic.

5.2. Lliurament de missatges

Un mecanisme de lliurament dels missatges és essencial per a garantir certes propietats del sistema, que es respectin les relacions d'ordre (per exemple, causalitat), que es garanteixi el lliurament dels missatges, i que es respecti el temps de lliurament acordat per a un missatge. Fa una funció similar a un protocol de transport com a TCP: oferir a les aplicacions un model de funcionament senzill i fiable.

Es tracta, doncs, de poder determinar:

- *Fiabilitat*: determina quins processos poden rebre una còpia del missatge.
- *Ordre*: en quin ordre arriben els missatges.
- *Latència*: durant quant de temps es pot estendre el lliurament d'un missatge.

Fiabilitat en el lliurament: cada participant haurà de guardar internament informació de l'estat i una còpia dels missatges, i també rebre confirmacions de lliurament per a oferir les garanties següents:

- *Atòmica*: a tots els membres d'un grup o a cap.
- *Fiable*: a tots els membres en funcionament (els que fallin no rebran el missatge, si falla l'emissor no hi ha garantia de lliurament).
- *Quòrum*: a una fracció del grup. Si falla l'emissor, no hi ha garantia de lliurament.
- *Intent* (*best effort*): a cada membre del grup, però cap no garanteix haver rebut el missatge.

Ordenació de missatges (lliurament): cada receptor tindrà una cua de lliurament que reordenarà els missatges segons les restriccions d'ordre seleccionades:

- *Total, causal*: en el mateix ordre en cadascun, respectant les relacions causals.
- *Total, no causal*: en el mateix ordre, sense tenir en compte relacions causals.
- *Causal*: en ordre respecte a potencials relacions causals.
- *FIFO*: en ordre des de cada un, però els missatges provinents d'altres poden arribar en qualsevol ordre.
- *Desordenat*: en qualsevol ordre (en l'ordre d'arribada)

Latència en el lliurament de missatges: els processos que es comuniquen hauran de tenir en compte la latència seleccionada per determinar si un missatge s'ha lliurat o ha fallat.

- *Síncrona*: comença immediatament i es completa en temps limitat.
- *Interactiva*: comença immediatament, però pot necessitar un temps finit però no limitat per al seu lliurament complet.

– *Limitat*: els missatges poden ser enviats a la cua o retardats, però el lliurament es fa en un temps límit.
– *Eventual*: els missatges poden ser enviats a la cua o retardats, i el lliurament pot reclamar un temps finit però no limitat per al seu lliurament complet.

Es poden produir diversos problemes de lliurament de missatges durant la comunicació amb un grup de processos que ofereixen un servei tolerant a fallades com el següent:

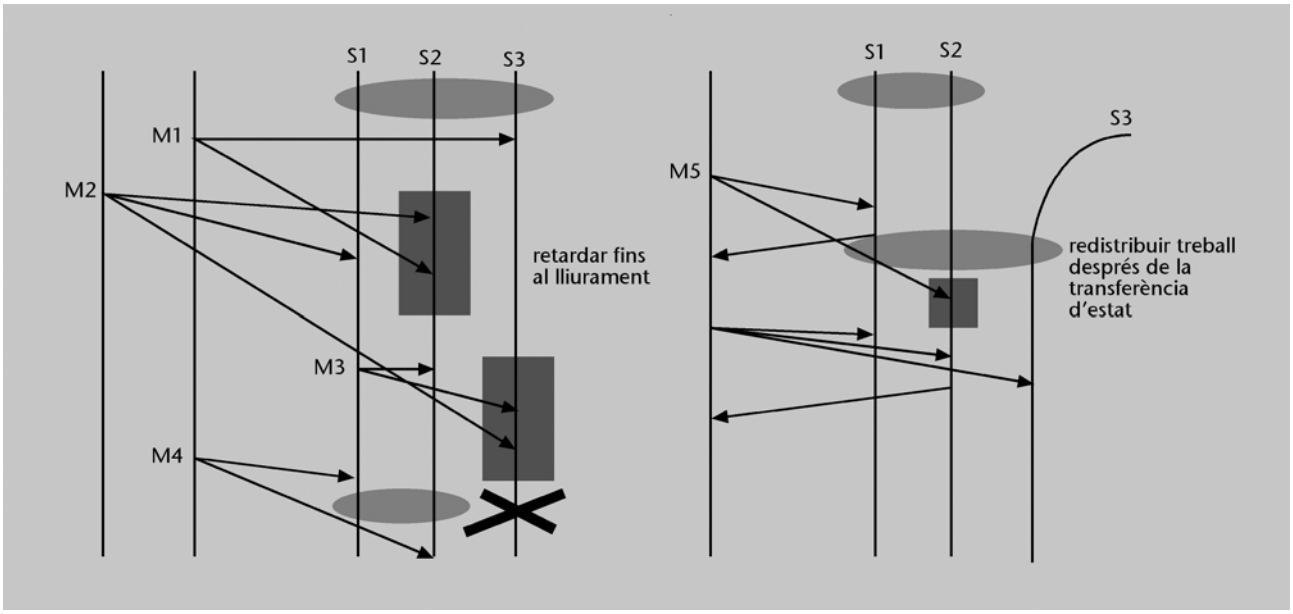


Figura 17. Problemes de lliurament de missatges a un grup de tres processos servidors

Problemes d'ordre total: el lliurament d'M1 i M2, que són missatges concurrents, es fa d'una manera diferent en S2 que en S1 i S3 (M1, M2; M2, M1; M1, M2). Si cal un lliurament amb ordre total, en S2 s'haurà de retardar el lliurament d'M2 fins que no arribi M1.

Problemes d'ordre causal: en S3 el lliurament d'M3 s'ha de retardar fins que arribi M2, ja que M2 (precedeix) → M3.

Problemes de fiabilitat del lliurament: mentre M4 s'està enviant, S3 falla. Si es desitja un lliurament fiable, es pot enviar a S1 i S2, però si es desitja un lliurament atòmic, llavors s'ha de cancel·lar el lliurament a S1 i S2, ja que S3 ha fallat i no rebrà M4.

Un problema subtil però important seria el que es dóna mentre es lliura M5: S1 rep el missatge M5 i, per tant, el seu estat canvia. S'incorpora un nou servidor i S2 s'encarrega de transferir a S3 la informació d'estat que tenia. S3 comença a fer servei després de la tramesa d'M5 i, per tant, no rep M5. S2 el rep just després d'haver transferit el seu estat a S3. A partir d'aleshores, S1 i S2 tenen un estat diferent d'S3 que no ha vist M5. Per a evitar el problema, sembla que durant l'operació de transferència d'estat, el sistema s'hauria d'aturar.

En l'exemple no s'han tingut en compte els límits de latència de lliurament dels missatges que podrien haver fet que algun missatge retardat s'eliminés del sistema.

5.3. Transaccions en presència de fallades

Una operació sobre diversos processos pot necessitar certes garanties perquè sigui equivalent al lliurament d'un sol procés sense error: envia exactament una sola vegada i, si falla l'emissor, un altre pren el seu lloc, no hi pot haver *pipelining* o concurrència.

El protocol pot requerir dues o tres rondes de comunicació:

– En la primera ronda, l'emissor proposa l'operació i recull l'acord de tots els receptors. Si un no la pot portar a terme, respon negativament i l'operació es cancel·la.

– En la segona ronda, l'emissor confirma que l'operació es pot portar a terme (o notifica a tots que l'operació s'ha cancel·lat).

– Opcionalment, en la segona ronda, els receptors poden confirmar que cada un ha pogut portar a terme l'operació i aleshores hi ha una tercera ronda en què l'emissor comunica a tots els receptors que l'operació ha anat bé i que ho poden deixar estar.

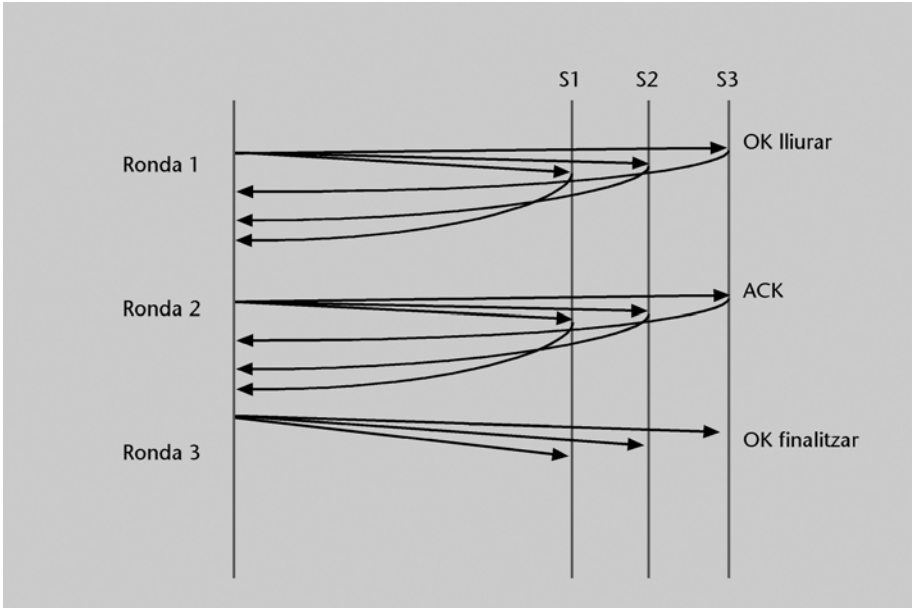


Figura 18. Transacció en tres fases

Podem observar que el lliurament d'un sol missatge a tres destinataris amb garanties transaccionals (amb fiabilitat atòmica, com si anés a només un) pot generar molt més dels tres missatges que inicialment podríem imaginar.

6. Consens

Quan es parla de consens en sistemes distribuïts es fa referència a un conjunt de processos que han de posar-se d'acord en un valor una vegada un o més d'un d'aquests processos han proposat quin hauria de ser aquest valor.

Ja hem vist anteriorment en l'apartat 5 que els diferents components de la llançadora espacial han de posar-se d'acord abans de realitzar certes accions crítiques. També hem vist el problema dels generals bizantins, en què dos exèrcits han de decidir de manera consistent si ataquen o es retiren. El mateix passaria en el cas de transaccions per a transferir fons d'un compte a un altre: els ordinadors implicats han de posar-se d'acord de manera consistent a fer les respectives operacions de dèbit i crèdit. En exclusió mútua, els processos han de posar-se d'acord en quin procés pot entrar a la secció crítica. En elecció, els processos es posen d'acord en quin procés escullen. En *multicast* amb ordre total, els processos es posen d'acord en l'ordre de lliurament dels missatges.

Existeixen protocols fets a mida per a aquests tipus d'acords. Ja n'hem vist alguns i en veurem més en la resta d'apartats d'aquest mòdul. El que veurem en aquest apartat és com es caracteritzen tant el problema com les solucions i veurem un algorisme de consens molt utilitzat.

Suposem que tenim una col·lecció de processos p_i (i = 1, 2, ... N) que es comuniquen per pas de missatges. Ens interessa arribar a consens encara que hi hagi fallades. Assumim que la comunicació és fiable, però que els processos poden fallar.

Per a arribar a un consens, cada procés p_i comença a l'estat de *no-decisió* i *proposa* un valor v_i. Els processos es comuniquen els uns amb els altres intercanviant valors. A continuació, cada procés fixa un valor en una *variable de decisió* d_i. En fer-ho entra a l'estat *decidit*, en el qual ja no es podrà canviar d_i.

Cadascuna de les execucions d'un algorisme de consens hauria de satisfer les següents condicions:

- **Acabament**: tard o d'hora cada procés correcte acaba assignant un valor a la seva variable de decisió.
- **Acord**: el valor decidit per tots els processos correctes és el mateix.
- **Integritat**: si tots els processos correctes han proposat el mateix valor, llavors qualsevol procés correcte en l'estat de *decisió* ha escollit aquest valor.

Per a entendre com aquesta formulació es tradueix a un algorisme, considerem un sistema en el qual els processos no poden fallar. En aquest cas és molt sen-

zill resoldre el consens. Només cal agrupar els processos en un grup i fer que cada procés enviï de manera fiable les seves propostes de valor a la resta de membres del grup. Cada procés espera fins que ha rebut tots els valors de tots els altres processos i es queda amb el que hagin proposat la majoria (o no se'n decideix cap si no hi ha majoria). Aquesta funció *majoria* és la mateixa per a tots els processos. La fiabilitat de l'enviament garanteix l'acabament. L'acord i la integritat es garanteixen per la definició de la funció *majoria* i la propietat d'integritat de l'enviament fiable. Tots els processos reben el mateix conjunt de valors i tots els processos apliquen la mateixa funció a aquests valors i, consegüentment, han de posar-se d'acord.

Si els processos poden fallar, la cosa es complica. Cal detectar les fallades i no és clar que una execució de l'algorisme de consens pugui acabar.

Si els processos poden fallar de manera *bizantina*, els processos que fallen poden comunicar valors aleatoris als altres. Encara que això pugui semblar poc probable, no és una cosa que no pugui passar si el procés té una errada que el fa fallar d'aquesta manera. Per complicar-ho més, aquesta errada podria no ser fortuïta, és a dir, el resultat d'un comportament malintencionat o malèvol.

Un cop introduït el problema del consens, veurem un dels algorismes de consens més populars.

6.1. Algorisme de Paxos

L'algorisme de Paxos és un algorisme tolerant a fallades que permet arribar a consensos en sistemes distribuïts. Funciona en el model de pas de missatges amb asincronia i amb menys de $n/2$ fallades (però no amb fallades bizantines). L'algorisme de Paxos garanteix que s'arribarà a un acord i garanteix l'acabament si hi ha un temps suficientment llarg sense que cap procés reinicïi el protocol.

Història de l'algorisme de Paxos

L'algorisme de Paxos està considerat un dels algorismes més eficients per a aconseguir consens en un sistema de pas de missatges on es poden detectar fallades de processos. Però en un primer moment això no va ser així...

L'algorisme de Paxos va ser descrit per Lamport per primera vegada el 1990 en un informe tècnic que va ser considerat per molts com una broma. Si voleu saber la descripció de la història que en fa Lamport, podeu visitar la pàgina http://research.microsoft.com/users/lamport/pubs/pubs.html#lamport-paxos.

Els processos es classifiquen en *proposadors*, *acceptadors* i *aprenents* (un mateix procés pot tenir tots tres rols). La idea és que un proposador intenta ratificar un valor proposat a força de recollir acceptacions d'una majoria d'acceptadors, i els aprenents observen aquesta ratificació. Els aprenents, llavors, intenten comprovar que se n'ha fet la ratificació.

La intuïció darrera del funcionament de l'algorisme és que qualsevol proposador pot reiniciar el protocol generant una nova proposta (i així tractar els bloquejos), i que hi ha un procés per a alliberar els acceptadors dels seus vots anteriors si es pot provar que els vots anteriors eren per a un valor que properament no obtindrà una majoria.

Per a garantir que el sistema progressa* cal seleccionar un proposador (líder) que sigui qui generi les propostes. Cada procés tindrà en tot moment una estimació de qui és el líder. Quan es vulgui realitzar una operació, aquesta s'enviarà a qui sigui el líder en cada moment. El líder seqüencia les operacions i posa en marxa un algorisme de consens per a arribar a acords.

*** Que el sistema progressa vol dir que va passant per les diferents fases i tard o d'hora acaba.**

L'algorisme de consens s'estructura en dues fases: *prepara* i *accepta*. El líder contacta amb una majoria en cadascuna de les fases. El protocol permet que en moments puntuals hi hagi més d'un líder concurrentment. Encara que més d'un d'aquests líders generi una votació, en tot moment es pot distingir entre els valors proposats pels diferents líders.

Per a organitzar el procés de votacions, s'assigna un número de proposta diferent a cada proposta. La manera més senzilla de generar aquests números de proposta és que siguin parelles formades per una marca de temps (n) i l'identificador de l'originador (p). Un parell $<n1,p1>$ és més gran que un altre parell $<n2,p2>$ si $((n1 > n2)$ o $((n1 = n2)$ i $(p1 > p2))$. El líder escull números de proposta de manera local, única i monòtonament creixent. És a dir, si el darrer número de proposta és $<n,q>$ llavors escull $<n + 1,q>$.

Marca de temps (*timestamp*, en anglès)

És un número que creix monòtonament. Inicialment val 1 i cada vegada que es vol una nova marca de temps s'incrementa el seu valor en 1. La combinació de la marca de temps i l'identificador del procés farà que el número de proposta sigui únic.

L'algorisme funciona de la manera següent:

Fase 1

a) Un proposador selecciona un nou número de proposta n i envia una petició *prepara*(n) a tots els acceptadors.

b) Si un acceptador rep una petició de *prepara* amb un número n més gran que qualsevol altra petició de *prepara* que hagi respost fins aquell moment, llavors contesta a la petició amb una promesa de no acceptar cap altra proposta amb número inferior a n i amb el número de proposta més gran (si n'hi ha algun) que ha acceptat.

Fase 2

a) Si el proposador rep una resposta a la seva petició de *prepara*(n) d'una majoria d'acceptadors (la meitat més 1), llavors envia una petició d'*accepta*(n,v) a cadascun dels acceptadors, on v és el valor del número de proposta més gran entre totes les respostes rebudes, o és el nou valor a acceptar.

b) Si un acceptador rep una petició d'*accepta* per un número de proposta n, l'accepta tret que ja hagi respost a una petició de *prepara* per un número més gran que n.

L'acceptació és un fenomen local. Calen missatges addicionals per a detectar quines, si és que n'hi ha hagut alguna, de les propostes han estat acceptades per una majoria d'acceptadors.

Progrés

Seria fàcil construir un escenari en què hi hagi dos processos generant una seqüència de propostes amb números que incrementin de manera monòtona, però que cap d'aquests números s'acaba escollint mai. El proposador p completa la fase 1 per un número de proposta n_1. Un altre proposador q completa la fase 1 per un número de proposta $n_2 > n_1$. Es rebutja la petició d'*accepta* de la fase 2 del proposador p pel número n_1 perquè tots els acceptadors han promès no acceptar cap nova proposta amb un número menor que n_2. D'aquesta manera, el proposador p comença i completa la fase 1 per una nova proposta amb número $n_3 > n_2$, cosa que causant que es rebutgi la segona petició d'*accepta* de la fase 2 del proposador q. I així per sempre més.

Per tal de garantir el progrés, cal escollir un proposador "distingit" que sigui l'únic que intenti generar propostes. Si el proposador distingit es pot comunicar amb una majoria suficient d'acceptadors, i si fa servir un número de proposta més gran que qualsevol número usat anteriorment, llavors tindrà èxit en generar una proposta que sigui acceptada. En cas que el proposador aprengui que hi ha números de proposta majors que el que està proposant, només ha d'abandonar la proposta que estigui fent i anar intentant números més grans fins que arribi a un número de proposta suficientment gran.

De tot això es conclou que si hi ha un nombre suficient de processos que funcionen correctament, escollint un líder, el sistema funcionarà.

Bibliografia sobre l'algorisme de Paxos

Per a més informació de l'algorisme de Paxos vegeu:

L. Lamport (1998). "The part-time parliament". *ACM Transactions on Computer Systems* (vol. 16, núm. 2, pàg. 133-169).

Article original de Paxos. Pot semblar una mica llarg, però està molt ben explicat. Exposa el funcionament de l'algorisme situant-lo a l'illa de Paxos a l'antiga Grècia. Aquesta contextualització pot despistar en un primer moment, però a mesura que es va llegint l'article ajuda a entendre els problemes i com els soluciona.

L. Lamport (2001). "Paxos made simple". *SIGACT News* (32(4), pàg. 18-25).

Versió reduïda de l'article sense la contextualització a l'illa de Paxos.

7. Conceptes bàsics de reproducció

La reproducció de dades consisteix a mantenir diferents còpies d'objectes de dades en diferents magatzems de dades. Un objecte és la unitat mínima de reproducció en un sistema. Per exemple, un objecte pot ser un fitxer XML, un registre d'una base de dades o una taula d'una base de dades.

La reproducció de dades és molt important en els sistemes distribuïts per dos motius principals:

a) millora la disponibilitat pel fet d'eliminar punts únics de fallada (ja que es pot accedir als objectes en diferents ubicacions);

b) millora el rendiment del sistema perquè els objectes es poden ubicar més propers als usuaris que hi han d'accedir i perquè el mateix objecte el pot servir més d'un magatzem.

Un efecte secundari molt interessant d'aquestes dues propietats és que contribueix a millorar l'escalabilitat del sistema perquè en pot suportar el creixement tot mantenint uns temps de resposta acceptables. La contrapartida és que la gestió de les diferents reproduccions d'un objecte és complexa.

Hi ha moltes tècniques per a gestionar la reproducció. Aquestes es poden classificar de diferents maneres. En aquest mòdul ens fixarem en dos paràmetres per a presentar dues possibles classificacions:

1) Quina reproducció es modifica.
2) Quan es propaguen les modificacions a la resta de reproduccions.

Segons el primer paràmetre, els protocols de reproducció es poden classificar en màster únic (en anglès *single-master)* i màster múltiple (en anglès *multi-master)*. Segons el segon paràmetre, en síncrones (*eager*) o asíncrones (*lazy*).

7.1. Màster únic enfront de màster múltiple

En el cas del màster únic hi ha una còpia principal de cada objecte, que anomenarem *primària*. Quan hi ha una modificació, aquesta s'aplica primer a la còpia primària. Després es propaga a la resta de còpies (que són les secundàries). En aquest model es pot llegir qualsevol reproducció d'un objecte, però només es pot modificar la primària. Un exemple d'aquest tipus de sistemes és el DNS.

En l'aproximació màster múltiple hi ha diversos magatzems que contenen una còpia primària d'un mateix objecte. Totes aquestes còpies es poden actu-

alitzar concurrentment. En aquest cas es pot accedir per lectura a qualsevol de les còpies i per modificació a qualsevol de les primàries. En són exemples el CVS o els sistemes que fan servir les PDA per a sincronitzar les dades.

cvs

Concurrent version system és un sistema de control de versions que permet als usuaris editar fitxers de manera col·laborativa. També permet obtenir versions antigues. És un sistema molt popular pel desenvolupament del programari.

L'aproximació màster múltiple redueix els colls d'ampolla i els punts de fallada i incrementa la disponibilitat. Per contra, per tal de garantir la consistència de les dades calen mecanismes de coordinació i reconciliació entre les diferents reproduccions d'aquesta.

7.2. Sistemes síncrons enfront de sistemes asíncrons

7.2.1. Síncrons

El sistemes de propagació **síncrons** apliquen les actualitzacions a totes les reproduccions d'un objecte com a part de l'operació de modificació. Això fa que quan l'operació d'actualització acaba, totes les reproduccions tenen el mateix estat.

Aquests mecanismes es poden implementar utilitzant algorismes com reserva en dues fases –en què quan es vol fer una actualització, primer es bloquegen totes les reproduccions, després s'actualitzen i, finalment, s'alliberen–, o basats en marques de temps. Confirmació en dues fases proporciona, a més, atomicitat (o bé totes les transaccions acaben o bé no se n'aplica cap).

Amb aquest tipus de tècniques s'aconsegueix que, tot i que hi hagi diverses còpies d'un mateix objecte, l'usuari percebi que el comportament és com si només n'hi hagués una. Aquest criteri de consistència es coneix com a *one-copy seriability*.

Els protocols síncrons més senzills d'implementar utilitzen màster únic. També n'hi ha màster múltiple, per exemple, ROWA (*read-one/write-all*), en el que es pot llegir qualsevol còpia, però perquè l'operació d'escriptura es completi cal que s'actualitzin totes les còpies. Té el problema que no és tolerant a fallades perquè si una de les còpies no està disponible, l'escriptura s'ha d'avortar; ROWAA (*read-one/write-all available*) aborda aquesta limitació actualitzant només les còpies disponibles. Altres protocols utilitzen quòrums. En aquests una operació d'escriptura té èxit sempre que hi hagi un nombre determinat de còpies (quòrum) que executen l'operació.

També hi ha protocols que aprofiten els avantatges dels sistemes de comunicació en grup per a evitar alguns problemes de rendiment.

Sobre els avantatges dels sistemes de comunicació en grup, vegeu l'apartat 5.1, "Comunicació fiable en grup", d'aquest mòdul didàctic.

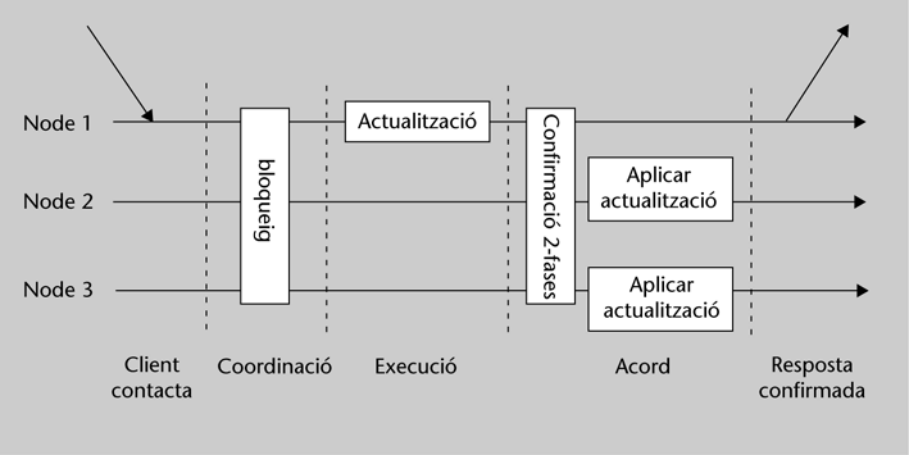


Figura 19. Exemple protocol síncron

La figura 19 mostra les diferents etapes que segueix un sistema distribuït síncron que utilitza un algorisme de confirmació en dues fases per a actualitzar un objecte. L'operació d'actualització s'inicia en el node 1. Com a part d'aquesta es bloquegen totes les reproduccions de la dada. Seguidament es fa la modificació en el node 1 i, utilitzant un algorisme de confirmació en dues fases, l'actualització es propaga a la resta de nodes. Finalment, l'operació d'actualització acaba. En aquest moment totes les reproduccions de l'objecte tenen el mateix valor. És important destacar que l'operació no retorna fins que totes les reproduccions han aplicat l'actualització.

El gran avantatge dels protocols síncrons és que eviten la divergència entre reproduccions d'una mateixa dada. El gran inconvenient és que qualsevol escriptura ha d'actualitzar moltes o totes les reproduccions abans de finalitzar. Això és un inconvenient per a sistemes els nodes dels quals siguin dinàmics –sistemes d'igual a igual o sistemes que permeten el treball en desconnectat– o en entorns de gran abast –on a causa de la latència és costós en temps actualitzar totes les reproduccions. A més a més, té grans limitacions d'escalabilitat degudes al temps necessari per a actualitzar totes les reproduccions.

7.2.2. Asíncrons

En els sistemes **asíncrons** no cal que s'actualitzin totes les còpies d'un objecte com a part de l'operació que inicia l'actualització. En aquest tipus de sistemes, l'operació d'actualització acaba tan aviat com pot i, posteriorment, el canvi es fa arribar a la resta de reproduccions.

Els sistemes asíncrons poden ser optimistes o no optimistes. Els primers fan la suposició que hi haurà poques actualitzacions que puguin ser conflictives. Així, els sistemes optimistes propaguen les actualitzacions en *background*. En cas que hi hagi algun conflicte, aquest es resol un cop ha ocorregut. Això fa que es requereixin mecanismes per a detectar i resoldre conflictes, així com mecanismes per a estabilitzar definitivament les dades. Els sistemes no optimistes, d'altra banda, consideren que és probable que hi pugui haver conflictes. Això fa que implementin mecanismes de propagació que eviten actualitzacions que puguin provocar conflictes. En aquest mòdul ens centrarem en els sistemes asíncrons optimistes.

Conflicte

Conjunt d'actualitzacions originades a diferents nodes que conjuntament violen la consistència del sistema. Per exemple, quan hi ha dues actualitzacions concurrents sobre una mateixa dada.

7.3. Models de reproducció síncrons

7.3.1. Reproducció passiva*

*** També anomenat *primari-secundari o primary-backup*, en anglès.**

És un model per a oferir reproducció tolerant a fallades caracteritzades pel fet que en tot moment hi ha un servidor de reproduccions primari i un o més de secundaris –*backups* o esclaus. En el model pur, els clients es comuniquen no-

més amb el primari. El primari executa les operacions i envia còpies de la dada actualitzada als secundaris. Si el primari falla, s'escull un dels secundaris perquè passi a ser el primari i aquest continua a partir del punt on el primari ho havia deixat.

Quan un client demana que s'executi una operació ocorre la seqüència d'esdeveniments següent:

1) El client fa la petició al primari.

2) El primari agafa cada petició de manera atòmica, en l'ordre en què les rep.

3) El primari executa la petició i emmagatzema la resposta.

4) Si la petició és una actualització, llavors el primari envia l'estat actualitzat, la resposta i l'identificador únic a tots els secundaris. Els secundaris envien un acusament de recepció.

5) El primari contesta al client.

Aquest model té l'inconvenient que introdueix sobrecàrrega. Proporcionar una vista síncrona de les dades implica haver d'actualitzar els secundaris abans de respondre, i si falla el primari la latència encara augmenta més mentre el grup es posa d'acord en l'estat del sistema i escull un nou primari.

Una variant del model és que els clients puguin enviar peticions als secundaris, i així alleugerir càrrega al primari.

7.3.2. Reproducció activa

Els models de reproducció activa acostumen a enviar les operacions d'actualització a totes les reproduccions (màster múltiple), i no només a una com fa el model primari-secundari. Cada reproducció, en rebre l'operació d'actualització, l'executa i respon. També es pot fer enviant la modificació (en lloc de l'operació).

Un problema que té la reproducció activa és que les operacions s'han d'executar a tot arreu en el mateix ordre, i això fa que calgui un mecanisme de *multicast* amb ordenació total.* Aquests mecanismes, però, tenen l'inconvenient que no escalen bé per sistemes distribuïts de gran escala. Una altra manera d'aconseguir aquesta ordenació total és utilitzant un coordinador, també anomenat **seqüenciador**. El client envia primer de tot l'operació al seqüenciador, que li assignarà un número únic seqüencial i, posteriorment, s'envia l'operació a totes les reproduccions. Les operacions s'executen en l'ordre del número de seqüència. Aquesta solució amb seqüenciadors s'assembla molt al model primari-secundari.

*** Això ho hem vist en l'apartat 5.1. "Comunicació fiable en grup".**

L'ús de seqüenciadors no soluciona els problemes d'escalabilitat. Hi ha propostes que aborden aquest problema, però estan fora de l'abast d'aquesta assignatura.

7.3.3. Basats en quòrums

En el model basat en quòrums l'operació a realitzar s'envia a una majoria de reproduccions, en lloc d'enviar-la a totes les reproduccions, com fa la reproducció activa. Més concretament, no cal que l'operació s'executi en totes les reproduccions sinó que només es demana que un subgrup del grup de reproduccions completi amb èxit l'execució de l'operació. La resta de reproduccions tindran una còpia antiga de la dada.

Es poden utilitzar números de versió o marques de temps per a saber quines dades estan desactualitzades. Si es fan servir números de versió, l'estat inicial de la dada és la primera versió, i després de cada canvi tenim una nova versió. Cada còpia d'una dada té un número de versió, però només les versions que estan actualitzades tenen el número actual de versió, mentre que les versions que no estan actualitzades tenen números de versió anteriors. Les operacions només s'han de fer sobre les còpies amb el número de versió actual.

Explicarem el funcionament dels sistemes basats en quòrum a partir d'un sistema de fitxers que va desenvolupar Gifford. En aquest sistema de fitxers, que té N còpies d'un fitxer, per a poder llegir un fitxer cal obtenir un quòrum de lectura, o sigui, un conjunt qualsevol de N_L servidors o superior. De la mateixa manera, per a modificar un fitxer cal un quòrum d'escriptura de com a mínim N_E servidors.

Lectura recomanada

D. K. Gifford (1979). "Weighted Voting for Replicated Data". *Proceedings of the seventh ACM symposium on Operating systems principles* (pàg. 150-162).

Els valors N_L i N_E estan subjectes a les restriccions següents:

1) $N_L + N_E > N$

2) $N_E > N/2$

La primera restricció es fa servir per a evitar conflictes lectura-escritpura, la segona evita conflictes escriptura-escriptura. Un fitxer només es pot llegir o escriure un cop un nombre suficient de servidors han acceptat participar-hi.

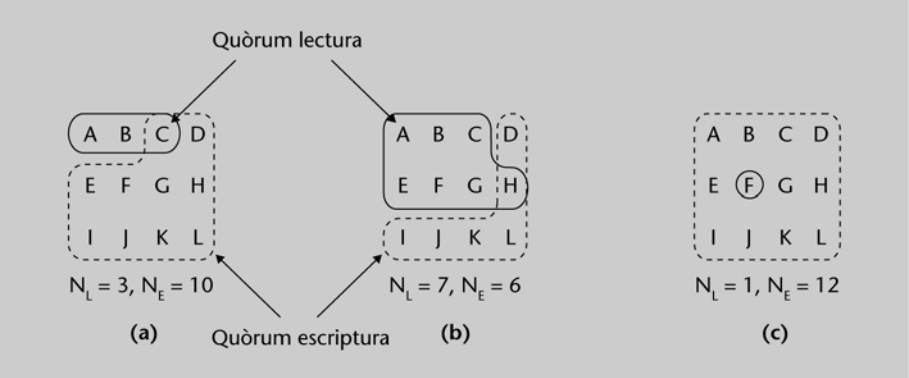


Figura 20

L'exemple de la figura 20 ens ajudarà a entendre el funcionament de l'algorisme. En la figura *a* el quòrum d'escriptura és 10. La darrera escriptura s'ha fet a les reproduccions de la *C* a la *L*. Qualsevol lectura haurà d'accedir a alguna d'aquestes deu còpies ja que el quòrum de lectura és de tres.

En la figura *b* hi poden ocórrer conflictes escriptura-escriptura perquè N_E és menor o igual que $N/2$. Podem tenir el cas en què un client esculli {*A*,*B*,*C*,*E*,*F*,*G*} com a conjunt d'escriptura i un altre client esculli {*D*,*H*,*I*,*J*,*K*,*L*}.

Finalment, la figura *c* és interessant perquè fixa el quòrum de lectura a 1. En aquest cas es pot llegir qualsevol reproducció a canvi que les actualitzacions s'hagin de fer a totes les còpies. Aquest esquema s'acostuma a anomenar ***read-one write-all*** **(ROWA)**.

7.4. Algorismes per reproducció síncrona

7.4.1. Reserva en dues fases*

*** *two-phase locking* o 2PL, en anglès.**

La reserva en dues fases és un mecanisme senzill per a garantir seriabilitat i *one-copy seriability*.

Seriabilitat: propietat que fa que el resultat d'executar una operació sigui el mateix que si el resultat s'hagués executat de manera seqüencial (sense superposicions degudes a la concurrència).

One-copy seriability: propietat que fa que un usuari percebi el comportament d'un conjunt de còpies d'una dada reproduïda com si només n'hi hagués una.

El protocol de reserva en dues fases gestiona les reserves (*locks*, en anglès) durant l'execució de la transacció en les dues fases següents:

1) S'adquireixen totes les reserves i no se n'allibera cap.

2) S'alliberen les reserves i no se n'adquireix cap.

Es garanteix la seriabilitat per a execucions que segueixin aquest ordre en la gestió de les reserves.

7.4.2. Confirmació distribuïda*

*** *distributed commit*, en anglès.**

Els algorismes de confirmació distribuïda són útils per a situacions en les quals interessa garantir que tots els processos d'un grup executen una operació o que cap d'ells l'executa. En el cas de *multicast* fiable, l'operació a executar seria el lliurament del missatge. En el cas de transaccions distribuïdes, l'operació seria la realització de la transacció.

Generalment, les operacions de confirmació distribuïda es basen que hi hagi un coordinador que notifica a la resta de processos que ja poden fer (o no fer) l'operació en qüestió (en local). Clarament, ja es veu que si un dels processos no pot fer l'operació, no hi ha manera de notificar-ho al coordinador. Per

aquest motiu calen uns mecanismes més sofisticats per a poder fer aquests confirmacions distribuïdes. A continuació presentem la confirmació en dues fases i la confirmació en tres fases.

7.4.3. Confirmació en dues fases*

* *two-phase locking* o 2PL, en anglès.

És un algorisme molt popular per a fer la confirmació distribuïda. Es basa a tenir un coordinador i la resta de processos. Si suposem que no hi ha fallades, el funcionament de l'algorisme seria el següent:

1. Fase petició de confirmació

- El coordinador envia una **petició de confirmació** a la resta de participants.
- El coordinador espera fins que rep un missatge de cadascun de la resta de participants.
- Quan un participant rep un missatge de **petició de confirmació** contesta al coordinador un missatge indicant si **està d'acord a fer la confirmació** local o **d'avortament** si no la pot fer.

2. Fase de confirmació

a) Èxit

Si el coordinador ha rebut un missatge de tots els participants en la fase de petició de confirmació indicant que estan **d'acord a fer la confirmació**:

- El coordinador envia un missatge de **confirmació** a tots els participants.
- Cada participant completa la transacció, i allibera tots els recursos i reserves adquirides durant la transacció.
- Cada participant envia un missatge d'acusament de recepció (*acknowledgement*, en anglès) al coordinador.
- El coordinador acaba la transacció un cop ha rebut tots els acusaments de recepció.

b) Fracàs

Si algun dels participants contesta un missatge d'**avortament** durant la fase de petició de confirmació:

- El coordinador envia un missatge a tots els participants perquè desfacin les operacions que hagin pogut fer o alliberin els recursos o reserves que tinguin ocupades.
- Cada participant desfà les possibles operacions que ja hagi fet i allibera els recursos i reserves que tingui ocupades.
- Cada participant envia un missatge d'acusament de recepció al coordinador.

- El coordinador acaba la transacció un cop ha rebut tots els missatges d'acusament de recepció.

El gran desavantatge del protocol de confirmació en dues fases és el fet que és un protocol que bloqueja. Un participant es bloquejarà mentre esperi un missatge. Això vol dir que altres processos que estiguin competint per la reserva d'un recurs que té ocupat el procés bloquejat hauran d'esperar que aquest alliberi la reserva. Un procés continuarà esperant fins i tot quan la resta de processos han fallat. Si el coordinador falla de manera permanent, alguns participants no resoldran mai la transacció. Això fa que els recursos estiguin ocupats per sempre més. L'algorisme es pot bloquejar indefinidament en els casos següents: quan un participant envia un missatge d'acusament de recepció al coordinador fa que es quedi bloquejat fins que rep un missatge de confirmació o de desfer les operacions realitzades. Si el coordinador falla i no es torna a recuperar, els participants es quedaran bloquejats fins que el coordinador es recuperi o, en el pitjor dels casos, indefinidament ja que aquest no pot decidir pel seu compte avortar o confirmar. L'únic que es podria intentar és esbrinar quin missatge havia enviat el coordinador.

Per un altre costat, el coordinador es quedarà bloquejat un cop hagi fet la petició de confirmació fins que tots els participants li contestin. Si un participant està indefinidament aturat, el coordinador decidirà avortar quan salti el temporitzador del node que està aturat. Aquesta decisió també és un desavantatge del protocol perquè està esbiaixat cap a l'avortament en lloc d'estar-ho cap a l'acabament.

A continuació, teniu un esquema del codi que executen tant el coordinador com els participants en què s'han tingut en compte fallades del coordinador i del participants.

a) Accions coordinador

```
escriu COMENÇAMENT al log local
envia a tots els participants PETICIÓ_CONFIRMACIÓ
mentre no s'han rebut tots els vots
    espera arribada vot
    si timeout
        escriu AVORTAMENT_GLOBAL al log local
        envia a tots participants  AVORTAMENT_GLOBAL
        exit
    enregistra resposta
si tots participants han enviat D'ACORD_CONFIRMACIÓ
    i el coordinador vota CONFIRMAR
        escriu CONFIRMACIÓ_GLOBAL al log local
        envia a tots els participants CONFIRMACIÓ_GLOBAL
si no
        escriu AVORTAMENT_GLOBAL al log local
        envia a tots els participants  AVORTAMENT_GLOBAL
```

b) Accions cada participant

```
escriu INICI al log local
espera PETICIÓ_CONFIRMACIÓ del coordinador
si timeout
    escriu AVORTAMENT al log local
    exit
si el participant vota CONFIRMACIÓ
    escriu D'ACORD_CONFIRMACIÓ al log local
    envia D'ACORD_CONFIRMACIÓ al coordinador
    espera la DECISIÓ del coordinador
    si timeout
        envia a la resta de participants PETICIÓ_DECISIÓ
        espera fins que es rep DECISIÓ // continua bloquejat
        escriu DECISIÓ al log local
    si  DECISIÓ == CONFIRMACIÓ_GLOBAL
        escriu CONFIRMACIÓ_GLOBAL al log local
    si no si DECISIÓ ==  AVORTAMENT_GLOBAL
        escriu AVORTAMENT_GLOBAL al log local
si no
        escriu AVORTAMENT al log local
        envia AVORTAMENT al coordinador
```

c) Thread que gestiona peticions de decisió (és un thread diferent. L'executen els participants)

```
mentre cert
    espera que arribi alguna PETICIÓ_DECISIÓ
    llegeix l'ESTAT més recent desat al log local
    si ESTAT == CONFIRMACIÓ_GLOBAL
        envia CONFIRMACIÓ_GLOBAL
    si no si ESTAT == INICI
            o ESTAT == AVORTAMENT_GLOBAL
        envia AVORTAMENT_GLOBAL
    si no
        no fer res // el participant continua bloquejat
```

Pot passar que un participant necessiti quedar-se bloquejat fins que un coordinador es recupera d'una fallada. Aquesta situació es pot donar quan tots els participants han rebut i processat una PETICIÓ_CONFIRMACIÓ del coordinador i, en paral·lel, el coordinador falla. En aquest cas els participants no poden resoldre de manera cooperativa quina decisió prendre. Hi ha diverses solucions per a evitar el bloqueig. Una solució, descrita per Babaoglu i Toueg, és (usant primitives de *multicast*): immediatament després de rebre un missatge, el receptor fa *multicast* del missatge a tots els altres processos. Aquesta aproximació permet que un participant arribi a prendre una decisió encara que el coordinador no s'hagi recuperat. Una altra solució és el protocol de confirmació en tres fases que comentarem a continuació.

Lectura recomanada

O. Babaoglu; S. Toueg (1993). "Non-blocking atomic commitment". A: S. Mullender (ed.). *Distributed Systems* (2a. ed., pàg. 147-168). Addison-Wesley.

7.4.4. Confirmació en tres fases*

* ***Three-phase commit*** **o 3PC, en anglès.**

Un problema del protocol de confirmació en dues fases és que quan el coordinador falla, els participants poden no ser capaços d'arribar a una decisió final. Això pot fer que els participants es quedin bloquejats fins que el coordinador es recuperi. Encara que el protocol de confirmació en tres fases sigui no bloquejant i sigui molt referenciat en la literatura, no es fa servir gaire a la pràctica, ja que les condicions en les quals el protocol de confirmació en dues fases es bloqueja ocorren rarament. Més concretament, el protocol de confirmació en tres fases fixa un llindar superior en la quantitat de temps necessari abans que una transacció o bé confirmi o bé avorti. Aquesta propietat assegura que si una transacció intenta confirmar via el protocol de confirmació en tres fases i agafa alguna reserva d'un recurs, alliberarà la reserva després d'expirar un temporitzador associat.

Coordinador

1) El coordinador rep una petició de transacció. Si hi ha alguna fallada en aquest punt, el coordinador avorta la transacció (és a dir, quan es recuperi, considerarà la transacció avortada). Altrament, el coordinador envia un missatge d'inici de transacció als participants i es posa en estat d'espera.

2) Si hi ha una fallada, expira el temporitzador, o si el coordinador rep un missatge d'algun participant dient que no està en disposició d'iniciar la tran-

sacció durant la fase d'espera, el coordinador avorta la transacció i envia un missatge d'avortament a tots els participants. Altrament, el coordinador rebrà missatges d'acceptació de l'inici de la transacció dels participants dins de la finestra de temps, i enviarà missatges de confirmació a tots els participants. A continuació, es posa en estat preparat.

3) Si el coordinador falla en l'estat de preparat, es mourà a l'estat de confirmació. En canvi, si el temporitzador del coordinador expira mentre està esperant la confirmació d'un dels participants, avortarà la transacció. En el cas que es reben tots els acusaments de recepció, el coordinador també canvia a l'estat de confirmació.

Participant

1) El participant rep un missatge d'inici de transacció del coordinador. Si el participant hi està d'acord, contesta amb un missatge indicant que està en disposició d'iniciar la transacció i es canvia a l'estat preparat. Altrament, envia un missatge indicant que no està en disposició d'iniciar la transacció i avorta. Si hi ha una fallada, es mou a l'estat d'avortar.

2) En l'estat de preparat, si el participant rep un missatge d'avortament del coordinador, falla, o si expira el temps d'espera per a una confirmació, avorta. Si el participant rep un missatge de confirmació, contesta un missatge d'acusament de recepció i confirma.

El principal desavantatge d'aquest algorisme és que no es pot recuperar d'una fallada de partició de la xarxa. És a dir, si els nodes se separen en dues meitats iguals, cada meitat continuarà pel seu compte.

7.5. Reproducció optimista

La reproducció optimista fa referència a un conjunt de tècniques per a compartir dades de manera eficient en un entorn de gran abast o mòbil. La característica principal que diferencia la reproducció optimista d'altres enfocaments és que les operacions d'actualització es fan sobre una reproducció qualsevol de la dada a actualitzar. Un cop feta aquesta, l'iniciador de l'actualització ja dóna la dada per actualitzada. Posteriorment l'actualització es propagarà en *background* a la resta de reproduccions de la dada. Aquest funcionament es basa en l'assumpció "optimista" que només ocorreran problemes molt esporàdicament. El seu avantatge principal és que augmenta la disponibilitat i el rendiment del sistema.

Lectura associada

Aquest apartat només ha pretès presentar alguns conceptes bàsics de la reproducció optimista. Per a conèixer les tècniques utilitzades per a implementar aquest comportament us recomanem que consulteu:

Y. Saito; M. Shapiro (2005, març). "Optimistic replication". *ACM Computing Surveys* (vol. 37, núm. 1, pàg. 42-81).

Aquestes tècniques són d'ús corrent tant a Internet com en la computació mòbil perquè Internet continua essent lenta i no fiable. A més, està creixent molt l'ús de dispositius mòbils amb connectivitat intermitent (per exemple, ordinadors portàtils o PDA). Un altre factor que hi ha contribuït és que algunes acti-

vitats humanes s'adapten molt bé a la compartició optimista, per exemple, en el desenvolupament cooperatiu de programari.

7.5.1. Passos seguits per un sistema optimista per a arribar a un estat consistent

En la figura següent es presenten els passos que segueix un sistema que funcioni utilitzant reproducció optimista per a arribar a un estat consistent.

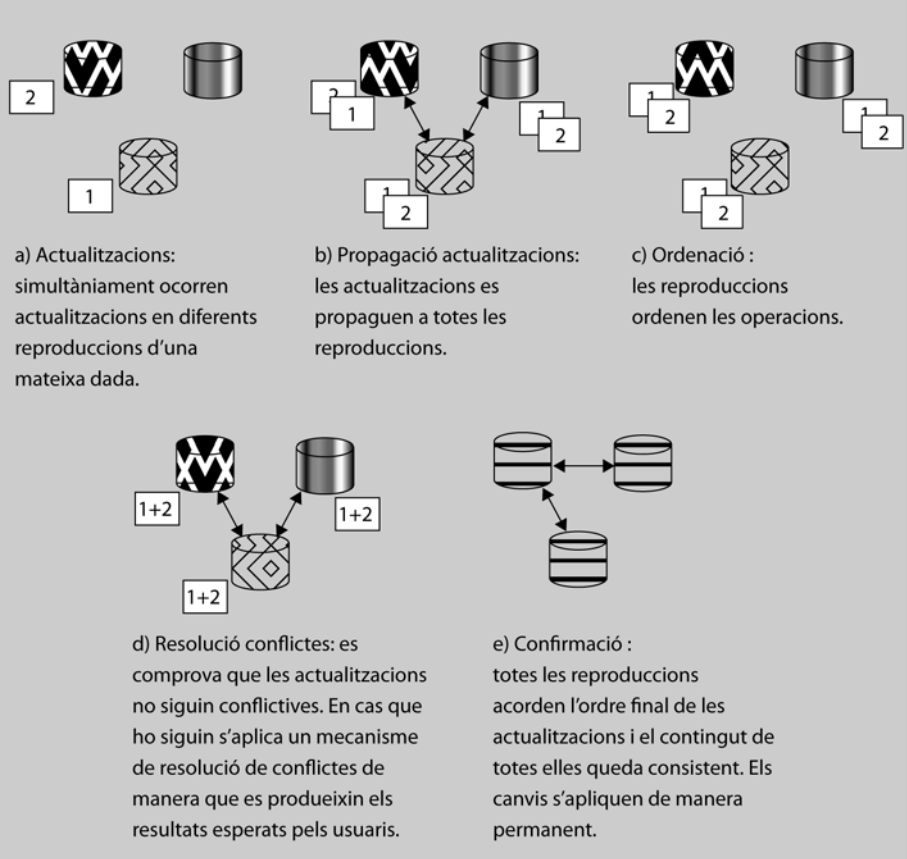


Figura 20

En aquest mòdul no entrarem en els detalls de cadascun d'aquests passos –si algú hi està interessat, pot consultar la referència que es proporciona a l'inici de l'apartat anterior. Tot i això, hi ha uns aspectes que considerem interessant de comentar:

Les operacions d'actualització del pas *a* poden ser operacions en què es canvia tot l'estat de l'objecte (*transferència d'estat*) o operacions en què s'especifica quina transformació cal fer a l'objecte (*transferència d'operació*). La transferència d'estat fa que la construcció del sistema sigui més senzilla perquè la propagació de l'actualització només implica la transferència de tot el contingut a les reproduccions no actualitzades. El DNS és un exemple de sistema que fa servir transferència d'estat. La transferència d'operacions és més complexa perquè cal que totes les reproduccions es posin d'acord en l'ordre d'execució de les ac-

tualitzacions. D'altra banda, pot anar bé quan els objectes són molt grans. A més, permet una resolució de conflictes més flexible. La transferència d'operacions també comporta que cada node hagi de mantenir un log d'operacions per si cal desfer i reordenar l'execució de les operacions. El CVS fa servir transferència d'operacions.

La propagació epidèmica...

... consisteix en el següent: quan dos nodes es comuniquen, intercanvien les seves operacions locals, així com les operacions que han rebut d'altres nodes.

La propagació d'operacions (pas *b*) s'acostuma a fer utilitzant tècniques epidèmiques. D'aquesta manera tots els nodes acaben rebent les actualitzacions encara que no es puguin comunicar directament.

Cada node pot rebre les operacions en un ordre diferent. En el pas *c* cada node intenta reconstruir un ordre d'execució que produeixi un resultat equivalent al de la resta de nodes i que encaixi amb els resultats intuïtius que espera l'usuari. Així, en un primer moment, una operació es considera com a proposta d'execució. Un node pot haver de reordenar o transformar operacions de manera repetida mentre no s'acabi decidint, entre tots els nodes, quin és l'ordre final de les operacions.

Les polítiques d'ordenació es poden classificar en sintàctiques i semàntiques. Els mètodes sintàctics ordenen les operacions només segons la informació de quan, on i qui va ser l'originador. Una manera molt usada per a aconseguir-ho són les marques de temps. Els mètodes semàntics intenten treure partit de les propietats semàntiques de l'aplicació, com ara la commutativitat o la idempotència de les operacions, per a reduir els conflictes i reduir la freqüència en què cal desfer operacions. Els mètodes semàntics només s'usen en sistemes de transferència d'operacions, ja que en sistemes que usen la transferència d'estat seria molt complex fer-ho.

Els mètodes sintàctics són més senzills, però poden detectar falsos conflictes. Els mètodes semàntics no sempre són fàcils de construir perquè cal tenir en compte les relacions semàntiques entre les operacions. Això acostuma a ser complex i les solucions no són generals.

Els usuaris d'un sistema optimista poden fer actualitzacions d'una dada sense saber que hi ha altres usuaris actualitzant el mateix objecte. A més a més, pot passar que els usuaris no estiguin disponibles quan es detecti que hi ha hagut el conflicte. Alguns sistemes no fan cap tractament de conflictes i senzillament es queden amb una de les dades i descarten la resta. Una manera millor de fer-ho és disposar de mecanismes de detecció i resolució de conflictes, que és el pas *d*. Entre aquests n'hi ha que senzillament avisen els usuaris que hi ha hagut un conflicte i aquests el resolen de manera manual. Altres sistemes implementen sistemes automàtics de resolució de conflictes. Aquests resoledors de conflictes tenen l'inconvenient que acostumen a ser complexos de construir. A més, són molt dependents de cada aplicació. La detecció i resolució de conflictes acostuma a ser la part més complexa dels sistemes de reproducció optimista.

Per acabar, el pas *e* correspon al mecanisme utilitzat pels nodes per a posar-se d'acord en l'ordre de les actualitzacions i en els resultats de la resolució de conflictes de manera que les actualitzacions es puguin aplicar de manera definitiva als objectes sense por que hi hagi reordenacions futures.

7.5.2. Alguns exemples de sistemes optimistes

El DNS, el CVS i les Palm (PDA) són tres sistemes molt populars que utilitzen reproducció optimista.

El DNS és un sistema màster únic que utilitza transferència d'estat. Els noms d'una zona estan gestionats per un node principal, que és qui té la còpia autoritzada de la base de dades de la zona i uns nodes secundaris que copien la base de dades del principal. Tant el servidor principal de la zona com els secundaris poden contestar les consultes dels clients i servidors remots. Les actualitzacions de la base de dades, en canvi, es fan únicament en el node principal. Periòdicament els nodes secundaris consulten el principal per si hi ha hagut canvis a la base de dades. Els servidors DNS recents suporten un comportament proactiu del servidor principal.

El CVS és un sistema màster múltiple amb transferència d'operacions que centralitza la comunicació a través d'un magatzem únic en una topologia en estrella. Hi ha un magatzem central que conté la còpia autoritzada dels fitxers així com els canvis que han ocorregut en el passat. Per a fer una actualització, un usuari crea una còpia local dels fitxers i els edita. Hi pot haver diversos usuaris treballant de manera concurrent, cadascun en les seves còpies locals dels fitxers. Quan un usuari acaba, confirma la seva còpia local al magatzem. Si cap altre usuari no ha modificat el fitxer, la confirmació acaba immediatament. Sinó, si les modificacions afecten línies diferents, les combina de manera automàtica i es confirma de la versió fusionada (aquesta resolució automàtica pot no ser correcta des del punt de vista semàntic, per exemple, pot ser que un fitxer fusionat així no compili). En un altre cas, s'informa a l'usuari que hi ha hagut un conflicte i aquest l'ha de resoldre.

Les agendes electròniques o PDA són exemples de sistemes màster múltiple amb transferència d'estat. Els usuaris les fan servir per gestionar la seva informació personal, com ara l'agenda de reunions o de contactes. De tant en tant sincronitzen la PDA amb l'ordinador i les dades viatgen bidireccionalment. Si una mateixa dada s'ha modificat en els dos costats, el conflicte es resol usant un resoledor específic de l'aplicació o manualment per l'usuari.

Altres sistemes optimistes a Internet són el *caching* de WWW, els *mirroing* d'FTP i els sistemes de notícies Usenet-news.

Resum

En aquest mòdul s'han descrit els problemes que presenta i els avantatges que ofereix un sistema distribuït, format per processos i/o màquines que es comuniquen per missatges que viatgen per una xarxa, com Internet.

Els sistemes distribuïts tenen comportaments difícils d'observar: no es poden fer "fotos instantànies" de l'estat del sistema i qualsevol observador tindrà una imatge diferent, depenent de la ubicació i del que triguin els missatges a arribar a través de la xarxa.

Si hi hagués un temps comú per a tots els processos, una marca de temps ens donaria la informació de què va passar i en quin ordre exacte. Com que això no és possible es presenten:

1) algorismes de sincronització de rellotges per a l'intercanvi de missatges, com l'algorisme de Christian, NTP o Berkeley;

2) algorismes per a abstreure's del pas del temps i retenir només la seqüència d'esdeveniments i les seves relacions: rellotges lògics, relació $\rightarrow$ (precedència o causalitat potencial), concurrència $||$.

La relació $\mathbf{e} \rightarrow \mathbf{e'}$ indica que l'esdeveniment **e** ha precedit **e'**, i que **e** és a la història causal de **e'**: **e** podria haver estat causa de **e'**. L'alternativa és la relació $||$ tal que si $\mathbf{e} || \mathbf{e'}$ podem dir que no estan relacionats per una relació de precedència. Els rellotges Lamport i les seves extensions vectorials o matricials permeten de caracteritzar l'essencial d'una execució en un sistema distribuït.

L'exclusió mútua evita interferències i assegura la consistència en l'accés a recursos quan hi ha més d'un procés que vol accedir a un recurs.

Els algorismes d'elecció permeten escollir un coordinador o procés que desenvolupi un rol especial. Hi ha molts algorismes que necessiten un procés que tingui aquest paper.

A còpia de repetir components, es poden construir sistemes distribuïts amb més rendiment i tolerància a fallades que cada component separadament i amb un cost més baix que una única màquina.

Les fallades es poden caracteritzar en termes de quan falla (transitori, intermitent, permanent) o de com falla (fallada-parada, erroni, lent). Per a simplificar,

és necessari introduir mecanismes que facin aparèixer totes les fallades com del tipus fallada-parada.

La reproducció és la solució però també és un problema: les còpies s'han de coordinar entre elles, han d'arribar a un consens. El problema dels dos exèrcits il·lustra la impossibilitat de consens quan la comunicació no és fiable. El problema dels generals bizantins permet determinar el nombre de components necessaris per a suportar cert nombre de fallades arbitràries als components.

Els components repetits necessiten mecanismes de comunicació a grup o *groupcast*. Hi ha dos models principals de comunicació en grups: primari-secundari i replicació activa.

El lliurament de missatges es pot caracteritzar per la seva fiabilitat (a qui es lliura), l'ordre (ordre relatiu a altres missatges) i la latència (durant quant de temps es pot estendre el lliurament dels missatges).

Per a lliurar un sol missatge de manera fiable a diversos destinataris, poden ser necessaris diverses interaccions i missatges entre tots: transaccions. Per tant, certes garanties tenen clarament un cost en temps, càrrega de les màquines i trànsit a la xarxa.

Un cop vistos aquests problemes relacionats amb la reproducció, hem presentat els conceptes bàsics d'una manera més general, per acabar fent èmfasi en la reproducció optimista.

Per consens es fa referència a un conjunt de processos que han de posar-se d'acord en un valor un cop un o més d'un d'aquests processos han proposat quin hauria de ser aquest valor.

Els sistemes de reproducció es poden classificar de moltes maneres, nosaltres ho hem fet segons dos paràmetres: segons quina reproducció es modifica (màster únic o màster múltiple) i segons quan es propaguen les reproduccions (síncrons a asíncrons).

S'han presentat tres models per reproducció síncrona i tres algorismes populars que s'utilitzen en aquests models.

Atès que les tècniques de reproducció optimista s'usen molt a Internet i la computació mòbil, hem presentat els passos que segueixen aquest tipus de sistemes per a arribar a un estat consistent i hem comentat tres exemples de sistemes que usen aquestes tècniques: CVS, DNS i les PDA.

Activitats

1. El cost d'un sistema únic de capacitat N vegades superior a la d'un sol PC acostuma a ser més car que N PC cooperant en un sistema distribuït. Mireu en el Web els preus d'un PC per fer de servidor web i calculeu a partir de quin valor de N és més barat fer un servidor web distribuït. També es podria mirar per al grau de fiabilitat (el temps de mitjana entre fallades) encara que aquesta és una dada més difícil d'aconseguir.
Per exemple, un servidor web de capacitat N·C es pot construir fàcilment utilitzant diversos PC de capacitat C fent que els servidors es reparteixin la càrrega entre ells i fent que el servei de noms DNS resolgui el nom del lloc a una adreça IP diferent cada vegada o torni una llista d'adreces (Round Robin DNS).

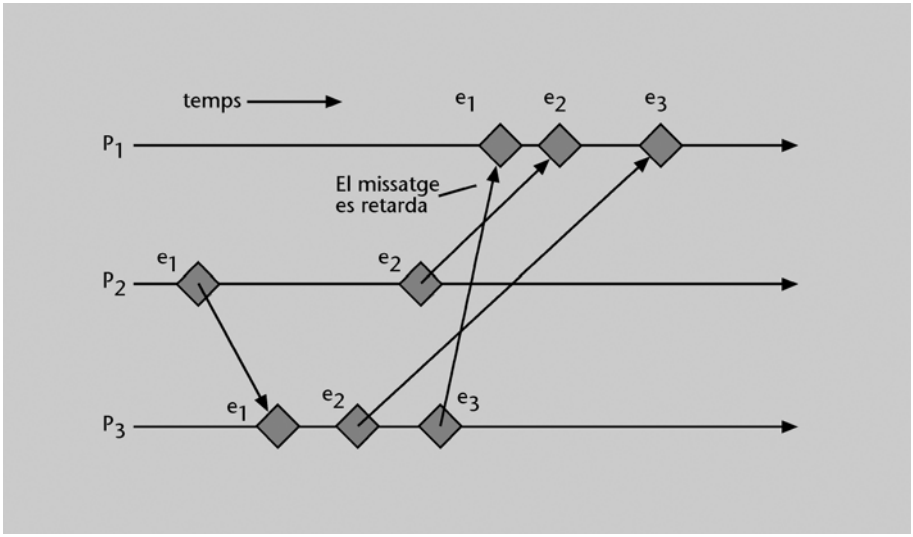
2. Sincronitzeu el rellotge del meu PC usant algun programa per a sincronitzar rellotges. Anoteu les variacions diàries de temps que es puguin mesurar i compareu-les amb l'hora d'un rellotge de polsera durant diversos dies.
Es poden usar programes com Dimension 4:
http://www.thinkman.com/dimension4/ per a Windows,
http://www.ua.es/es/servicios/si/ntp/configuracion.html,
i usar un servidor de temps com ntp.upc.es o d'altres.
Vegeu més informació sobre el projecte pilot de sincronització de temps a la xarxa acadèmica espanyola RedIRIS: http://www.rediris.es/gt/iris-ntp/ i http://www.rediris.es/gt/iris-ntp/drafts/

3. Visiteu diversos llocs web, esbrineu-ne l'hora i compareu si està o no està ben ajustada: podríem arribar a obtenir un contingut aparentment futur o un contingut actual que sembli vell a causa de desajustos en el rellotge.

Exercicis d'autoavaluació

1. Pensem en una societat primitiva en què els homes no sabien produir foc, quan en un llogarret el foc s'extingia (observació local) s'havia d'enviar un missatger amb una torxa a buscar-ne als llogarrets veïns. Si a cap llogarret veí no hi quedés foc (observació global) haurien d'esperar fins que amb la propera tempesta caigués un llamp que calés foc a un arbre. Dibuixeu la trajectòria de diversos missatgers a la recerca de foc i com podrien arribar a la conclusió errònia que el foc s'havia esgotat als tres llogarrets de la regió.

2. Dibuixeu amb un valor del rellotge de Lamport un sistema distribuït format per tres processos que intercanviïn alguns missatges segons el diagrama següent. Feu una llista dels esdeveniments que tenen relació de precedència i de concurrència. Verifiqueu que si $e \rightarrow e' \Rightarrow L(e) < L(e')$.



3. En un servei format per un grup de servidors repetits, feu una taula comparant com actuaria el model de gestió primari-còpia de seguretat, còpia disponible i votació per als casos: consultes freqüents, modificacions freqüents, un servidor amb fallada-aturada, funcionament erroni i funcionament lent.

4. Segons la fiabilitat/ordre/latència en el lliurament de missatges, classifiqueu els sistemes següents: correu electrònic, web, vídeo i una transacció bancària.

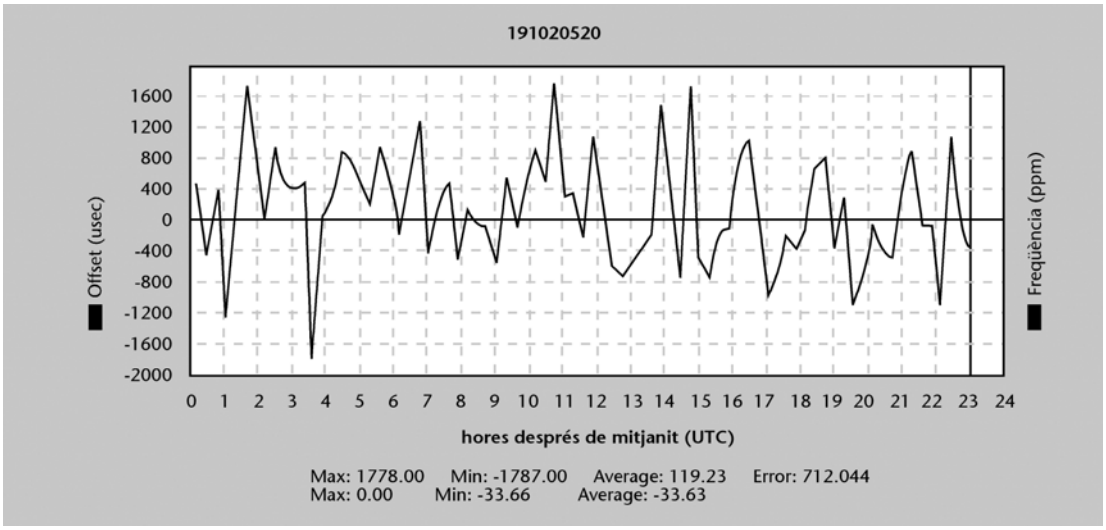
5. Dibuixeu el cas de tres processos amb una fallada bizantina i el cas amb fallades *fail-stop*. Justifiqueu si pot funcionar o no i quants processos podrien fer falta.

6. Feu una taula amb les diferències fonamentals entre servidors repetits organitzats com a primari-secundari i reproducció activa.

Solucionari

1. El valor de N pel qual un sistema distribuït és preferible a un de centralitzat depèn de les característiques del sistema seleccionat i del moment. En general, els preus dels processadors no creixen linealment sinó exponencialment a la velocitat, i també en la capacitat d'emmagatzematge. Valors de N 2 o 3 poden ser habituals.

2. Es tracta de familiaritzar-se amb els conceptes de sincronització de temps per aconseguir de sincronitzar el rellotge del nostre ordinador.
Es tracta també d'esbrinar quin és el marge de variació del nostre rellotge en referència al temps oficial. Depenent del programa seleccionat idealment, es podria arribar a dibuixar una gràfica similar a la que es mostra a continuació, amb aquesta o una altra escala més gran de temps.

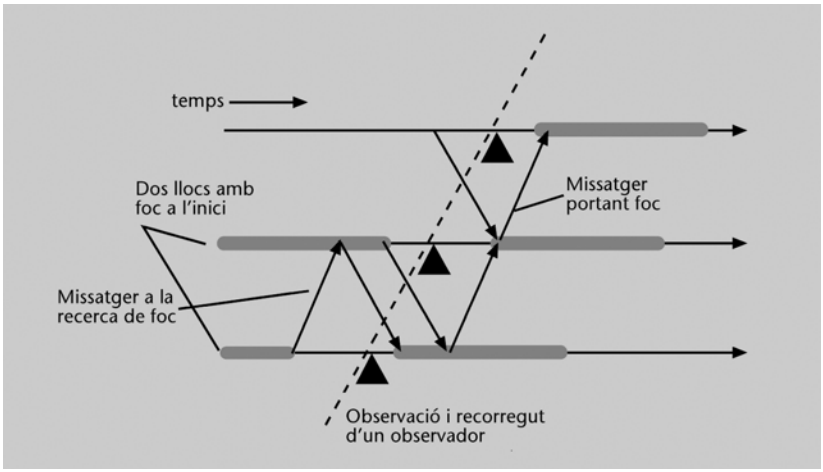


3. Per exemple, el codi següent s'obté visitant la pàgina web d'http://www.apache.org amb el comandament Telnet al port 80 del servidor i demanant informació sobre la pàgina principal mitjançant el comandament HEAD d'HTTP (en negreta el que s'escriu):

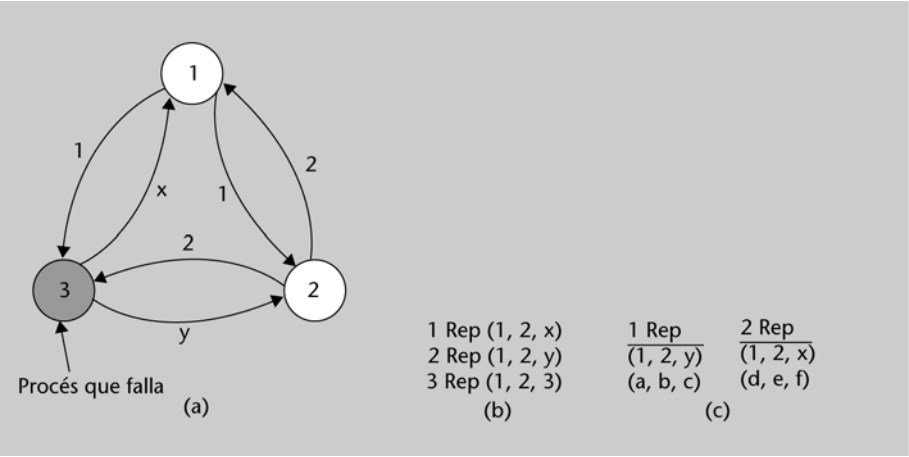
telnet www.apache.org 80
HEAD / HTTP/1.1
Host: www.apache.org

HTTP/1.1 200 OK
Date: Sun, 02 Jan 2005 23:51:01 GMT
Server: Apache/2.0.52 (Unix)
Cache-Control: max-age=86400
Expires: Mon, 03 Jan 2005 23:51:01 GMT
Accept-Ranges: bytes
Content-Length: 11795
Content-Type: text/html

4. En el diagrama següent es pot veure com un observador que viatgés pels llogarrets arribaria a una conclusió equivocada.



5.



Les fases de l'algorisme són *a*, *b* i *c*.

Es pot veure que si el procés 3 és bizantí (*x* i *y* tenen valors arbitraris), els processos 1 i 2 no poden decidir.

En canvi, si el procés 3 falla i s'atura en lloc de donar respostes incorrectes, i si assumim que només hi ha fallades de tipus fallada-parada, el sistema podria funcionar correctament només que funcionés un procés.

6.

Característica	Primari-secundari	Replicació activa
Tramesa missatges al grup	1 únic missatge (com si només hi hagués un servidor)	Missatge al grup (*multicast* o diversos missatges a cada rèplica)
Respostes del grup	1 única resposta (per tant, idèntic a tenir un únic servidor des del punt de vista del client)	Tantes respostes com còpies (el client ha de gestionar això: pot prendre només la primera i descartar la resta, prendre una majoria o esperar que arribin totes)
Organització del grup	Heterogènia (el primari és diferent i cal triar-lo si falla)	Homogènia (tots fan el mateix)
Senzillesa (com si no hi hagués reproducció)	En el client	A cada servidor (còpia)

Glossari

causalitat *f* Relació de causa i efecte entre dos fenòmens. Un esdeveniment que ha ocorregut abans que un altre i pertany a la seva història (esdeveniments relacionats amb l'actual per procés o missatges) mantenen una relació de causalitat potencial. Les situacions no causals són difícils de tractar i un sistema que ordeni el lliurament de missatges per preservar les relacions de causalitat facilita la programació.

commit Vegeu **confirmació**.

confirmació *f* Aplicació d'una operació de manera irreversible.
en commit

conflicte *m* Conjunt d'actualitzacions originades en diferents nodes que conjuntament violen la consistència del sistema.

fallada *f* Un sistema o procés de sobte pot deixar de funcionar correctament. La fallada pot fer que el sistema o procés respongui més lentament, torni respostes errònies o s'aturi.

fiabilitat *f* Probabilitat que una màquina, un aparell, un dispositiu, etc., acompleixi una determinada funció sota certes condicions durant un determinat temps. Les necessitats determinen el nombre de vegades o el temps que pot durar un error en un període de temps.

grup *m* Abstracció necessària per a agrupar diversos processos que cooperen per proporcionar un servei i al qual es poden dirigir missatges. Una capa de programari s'encarrega de repartir els missatges dirigits a un grup entre els processos que en formen part.

màster *m* Reproducció que té la capacitat d'acceptar actualitzacions.

màster múltiple *m* Sistema que suporta diversos màsters per objecte.
en multi-master

multi-master Vegeu **màster múltiple**.

màster únic *m* Sistema que suporta un màster per objecte.
en single-master

propagació epidèmica *f* Forma de propagació en què quan dos nodes es comuniquen intercanvien les seves operacions locals així com les operacions que han rebut d'altres nodes.

rellotge *m* Instrument per a mesurar el temps. En cada computador és necessari un dispositiu electrònic que emeti senyals periòdics per tal de controlar el temps de durada de les operacions. Aquest rellotge té imperfeccions i variacions respecte del patró de referència que fan que la mesura de temps només sigui vàlida dins la màquina. En un sistema distribuït, cada computador tindrà un rellotge amb valors de temps lleugerament diferent.

reproducció asíncrona *f* En els sistemes de reproducció asíncrona, no cal que s'actualitzin totes les còpies d'un objecte com a part de l'operació que inicia l'actualització. Posteriorment l'actualització es fa arribar a la resta de reproduccions.

reproducció síncrona *f* En els sistemes de propagació síncrons, les actualitzacions s'apliquen a totes les còpies de l'objecte com a part de l'operació d'actualització.

sincronia *f* Relacionat amb els processos que es donen en la direcció d'un senyal de rellotge. En un sistema distribuït es pot arribar a obtenir una sincronia virtual: un sistema distribuït asíncron que ofereix propietats equivalents com si cada procés estigués sincronitzat perfectament a un hipotètic rellotge global.

single-master Vegeu **màster únic**.

transferència d'estat *f* Tècnica que propaga les operacions recents tot enviant el valor de l'objecte.

transferència d'operació *f* Tècnica que propaga actualitzacions en forma d'operacions (transformació que cal fer sobre l'objecte).

Bibliografia

Coulouris, G.; Dollimore, J.; Kindberg, T. (2005). *Distributed Systems: Concepts and Design* 4/E. Londres: Addison-Wesley. (trad. cast.: *Sistemas Distribuidos: Conceptos y Diseño*, 3/E. Pearson, 2001).
És un llibre que tracta dels principis i disseny dels sistemes distribuïts, incloent-hi els sistemes operatius distribuïts. En aquest mòdul interessen els capítols 11: "Time and global states"; 12: "Coordination and agreement"; 13: "Transactions and concurrency control"; 14: "Distributed transactions"; 15: "Replication". Es tracta d'un text d'aprofundiment, amb un tractament molt exhaustiu de cada tema.

Tanenbaum, A.; Steen, M. (2007). *Distributed Systems: Principles and Paradigms*, 2/E. Prentice Hall.
Aquest llibre és una bona ajuda per a programadors, desenvolupadors i enginyers per tal d'entendre els principis i paradigmes bàsics dels sistemes distribuïts. Relaciona els conceptes explicats amb aplicacions reals basades en aquests principis. És la segona edició d'un llibre que ha tingut molt d'èxit tant pels aspectes que cobreix com pel tractament que en fa. En aquest mòdul interessen els capítols 6: "Synchronization"; 7: "Consistency and replication"; 8: "Fault tolerance".

Birman, K. (2005). *Reliable Distributed Systems. Technologies, Web Services, and Applications*. Nova York: Springer Verlag.
És un llibre que tracta dels conceptes, principis i aplicacions de les arquitectures i sistemes distribuïts. D'aquest mòdul interessa la part III, "Reliable Distributed Computing", i els capítols 23 –"Clock synchronization and Synchronous Systems"–, 24 –"Transactional Systems"– i 25 –"Peer-toPeer Systems and Probabilistic Protocols"– de la part V. També pot ser interessant veure com aquests aspectes estan implementats en els sistemes i aplicacions que es comenten en l'apartat IV –"Applications of Reliability Tecniques".

Saito, Y.; Shapiro, M. (2005, març). "Optimistic replication". *ACM Computing Surveys* (vol. 37, núm. 1, pàg. 42-81).
És un *survey* amb les tècniques més habituals de reproducció optimista.