Conceptes de sistemes distribuïts

Leandro Navarro Moldes
Joan Manuel Marquès i Puig
Pedro A. García López

P07/11068/00081

UOC
www.uoc.edu

Índex

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 consisteix a pensar un model de programació que permeti de 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.

Finalment, es veuen les taules de *hash* distribuïdes (DHT), que són un mecanisme distribuït que permet la localització eficient de dades a través d'un índex descentralitzat i uniformement repartit entre tots els nodes del sistema. Els nodes es connecten entre si seguint una estructura predeterminada com ara un anell, un hipercub o un arbre. Aquestes estructures asseguren que cada node necessita poques connexions (log N), que la xarxa té un diàmetre petit i que es pot fer arribar informació d'un node a un altre en pocs salts (log N).

La forma d'adquirir els coneixements passa per a realitzar els petits experiments que s'ofereixen en l'apartat d'activitats i en la web de l'assignatura, i que ajuden tant a concretar les idees centrals, com a tenir experiències pròpies i personals dels fenòmens, tècniques i eines que es descriuen.

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 els models del sistema 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 que ofereix el model de programació que es presenta per a organitzar la distribució d'un sistema.

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

6. Entendre què poden aportar les taules de *hash* distribuïdes al disseny de sistemes distribuïts de gran escala.

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.

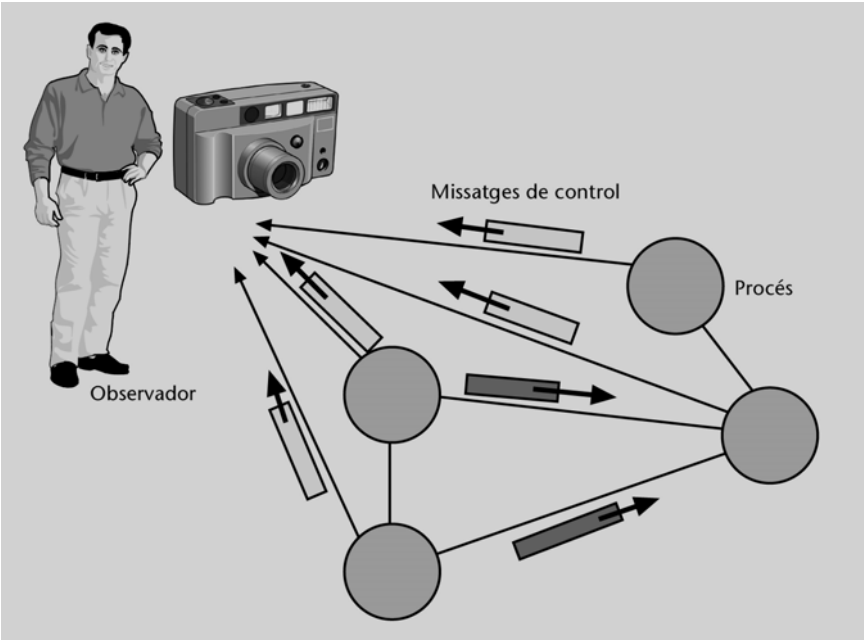


Figura 1. L'observació d'un missatge es fa com un procés més del sistema distribuït

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.

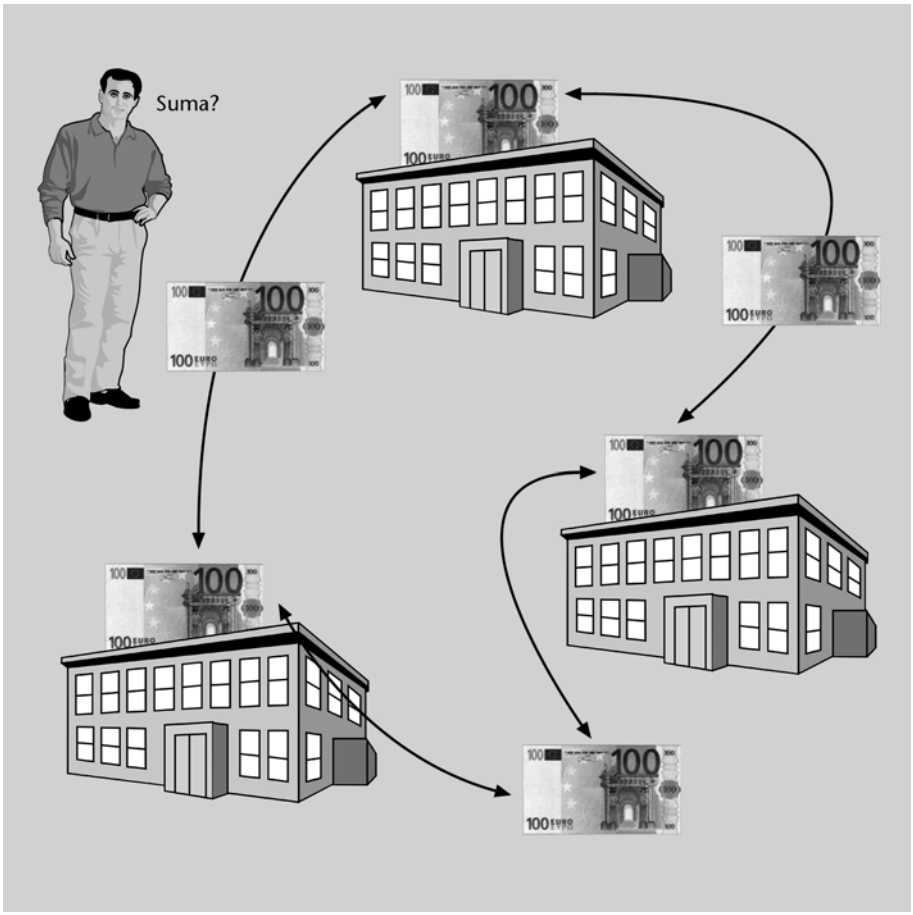


Figura 2. Comptar el valor global del nombre de bitllets en circulació

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).

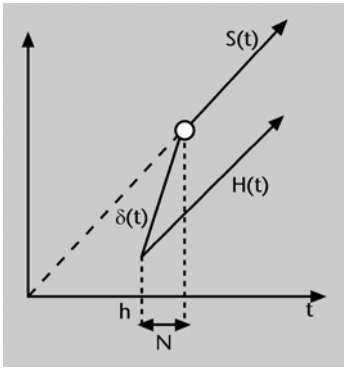


Figura 4. Ajust del rellotge progressiu en N passos

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):

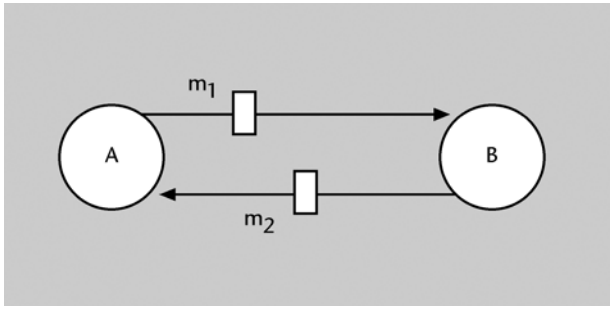


Figura 5. Intercanvi de missatges per a demanar marca de temps a B

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}.

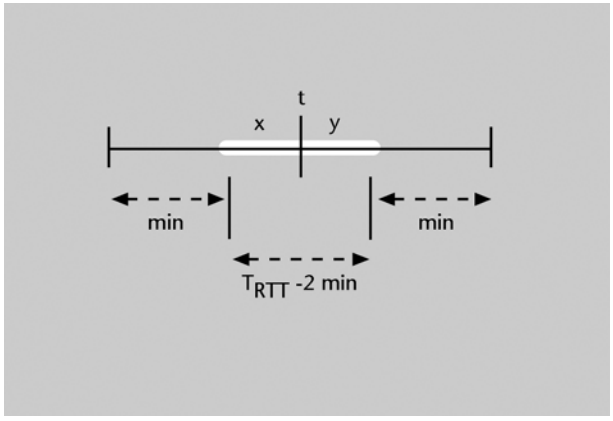


Figura 6. Marges de variació de la mesura de temps segons el retard

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.

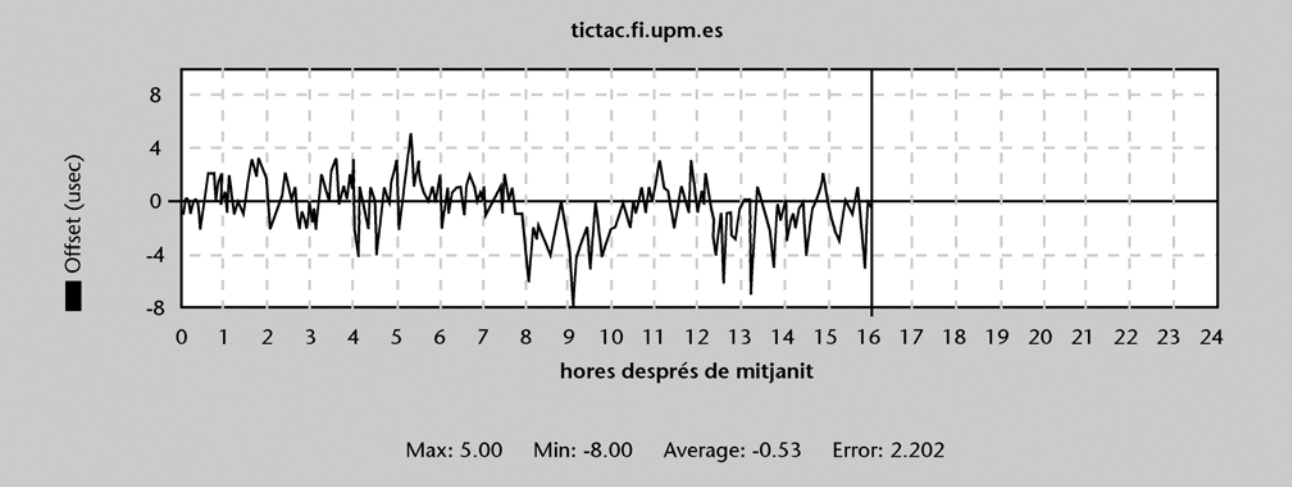
Sincronització de temps

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.



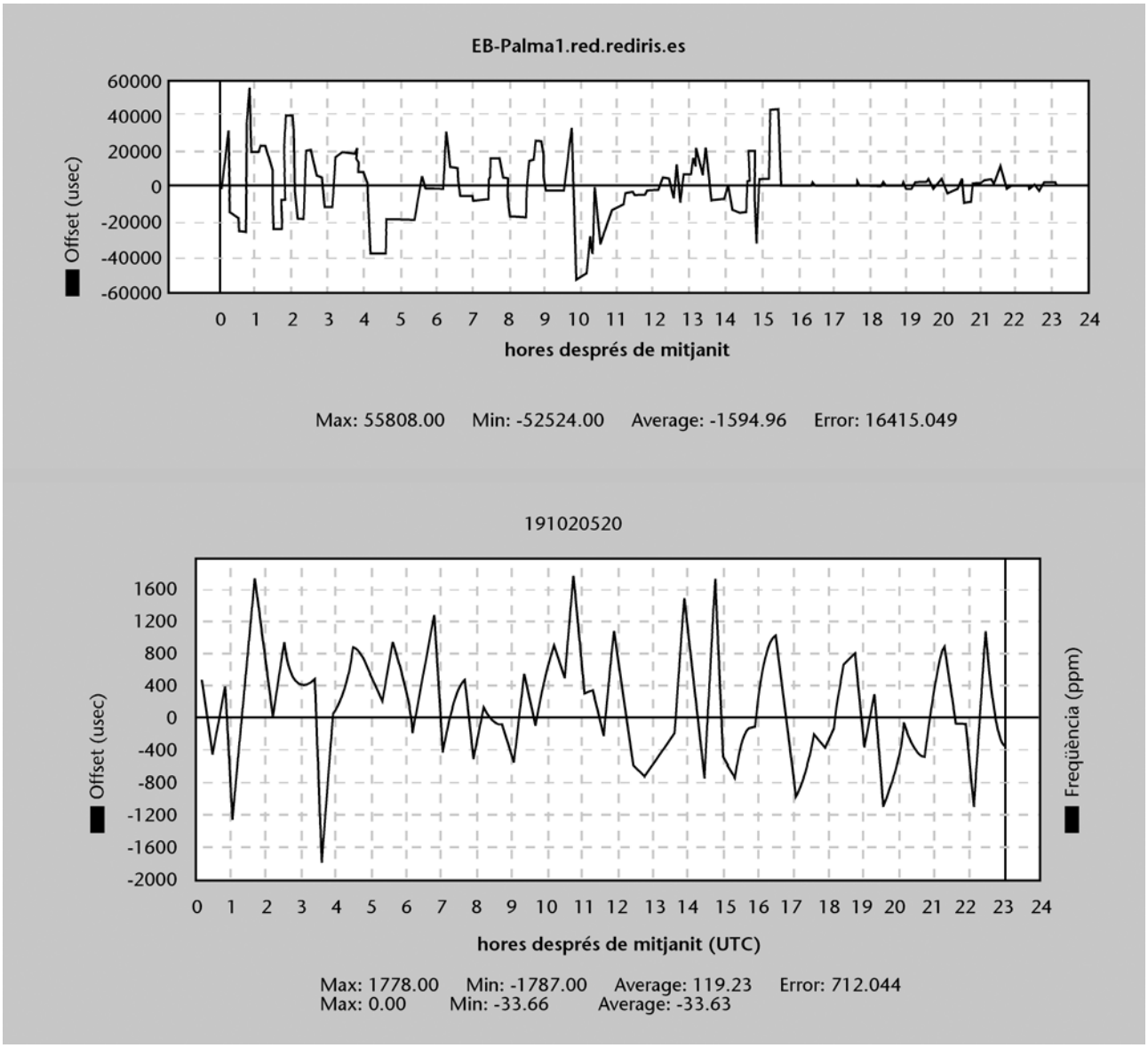


Figura 7. Variació de diversos servidors NTP en microsegons al llarg d'un dia (20/5/2002).
– "stratum 1": tictac.fi.upm.es, Sincronitza amb el GPS
– "stratum 2": eb-palma1.red.rediris.es, sincronitzat amb servidors "stratum 1" per la xarxa.
– "stratum 3": hora.usc.es, sincronitzat per la xarxa amb 3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

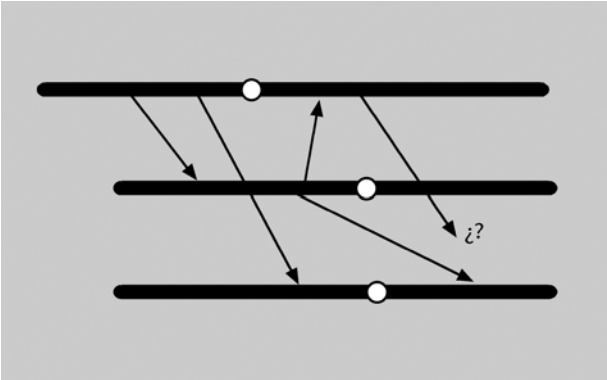


Figura 8. Diagrama d'una computació distribuïda.
Cada línia és un procés que envia o rep missatges en forma de fletxes. La inclinació indica la velocitat amb què viatja el missatge que pot variar per a cada un en funció de la càrrega de la xarxa i les màquines.
El missatge entre ¿? pot ser conseqüència 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.

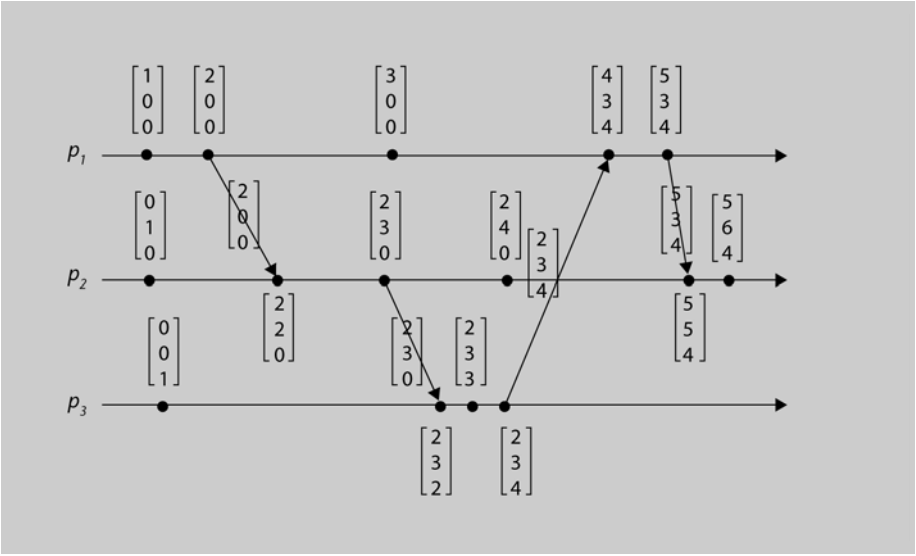


Figura 9. Evolució del vector temps en un sistema de dimensió 3.

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 $(V_k < V_h)$

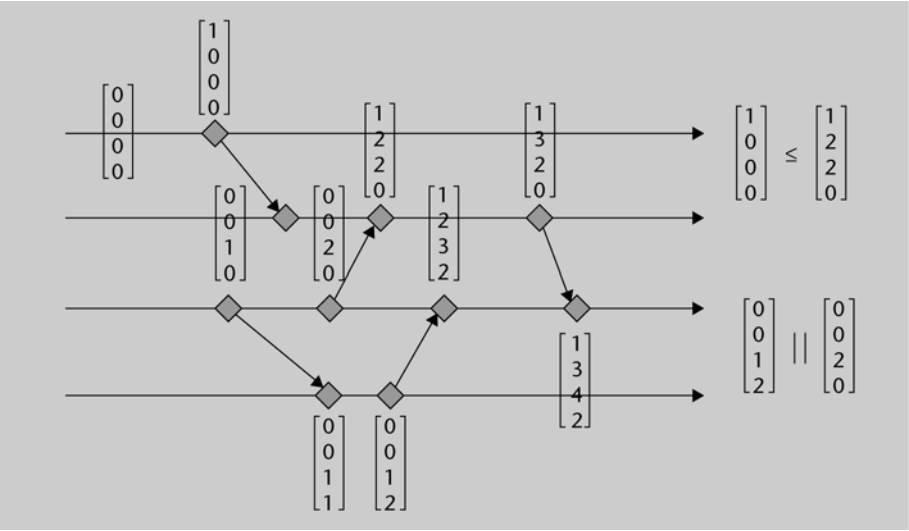


Figura 10. Rellotges vectorials: les línies de punts indiquen la frontera de la història causal.

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. Tolerància a fallades

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

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".

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.

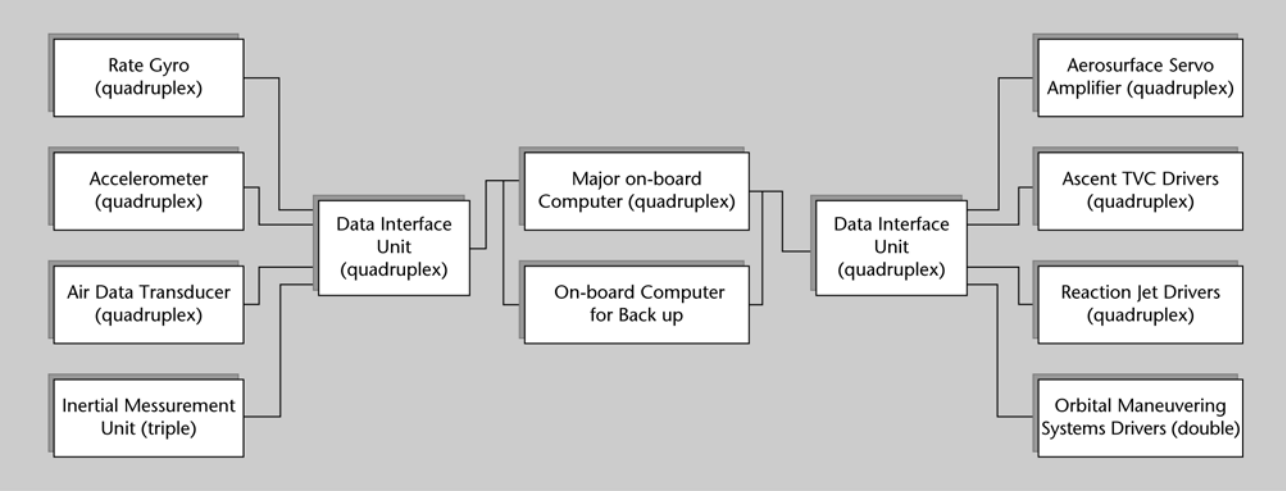


Figura 11. Esquema de la redundància del sistema de vol de la llançadora espacial de la NASA.

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.

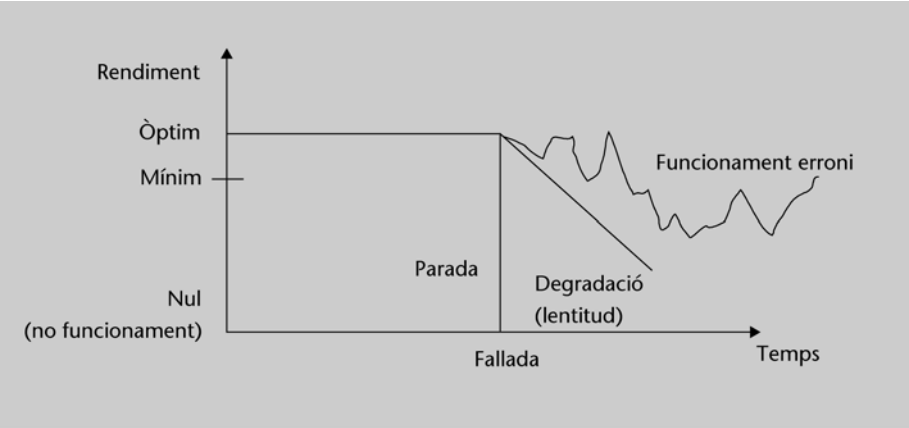


Figura 12. Rendiment en funció del temps. El model "desitjable" és fallada i parada, però no és l'únic i això complica el tractament de les fallades.

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.

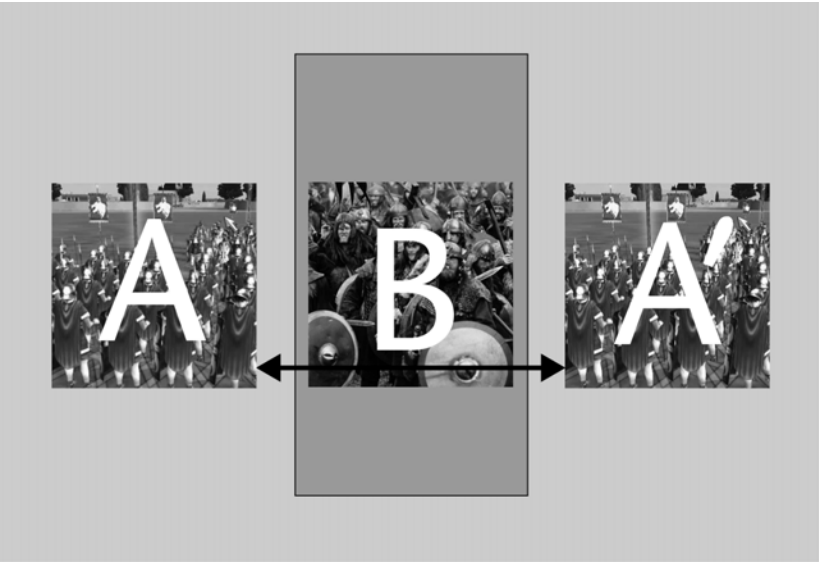


Figura 12. L'exèrcit B només pot ser derrotat si els exèrcits A i A' sumen les seves forces i ataquen alhora. Per a això han d'enviar un missatger que ha de travessar el campament contrari i pot fallar en el camí. Per exemple, A envia un missatger indicant l'hora de l'atac m (A, A', 12.30 h), però A' 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.

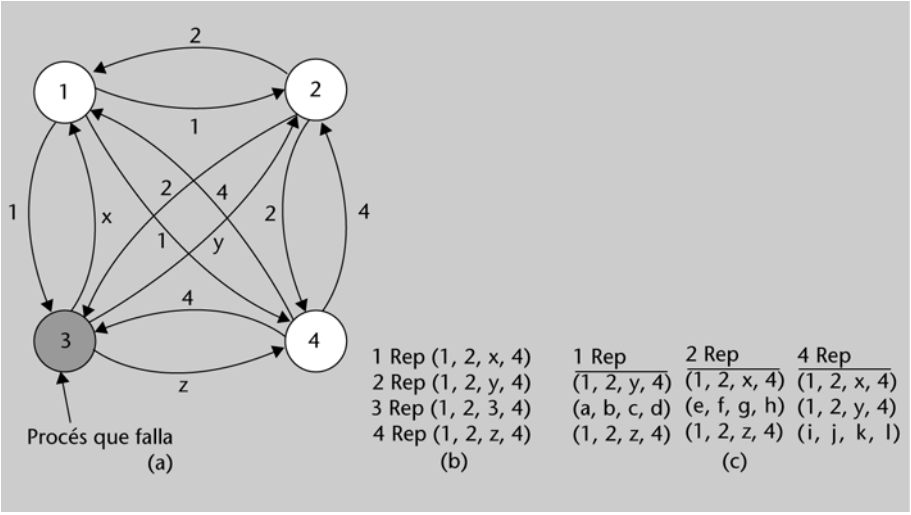


Figura 13. El problema dels generals bizantins amb tres generals lleials i un traïdor. Amb dos generals lleials no seria possible detectar el traïdor.

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.

3.1. 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.

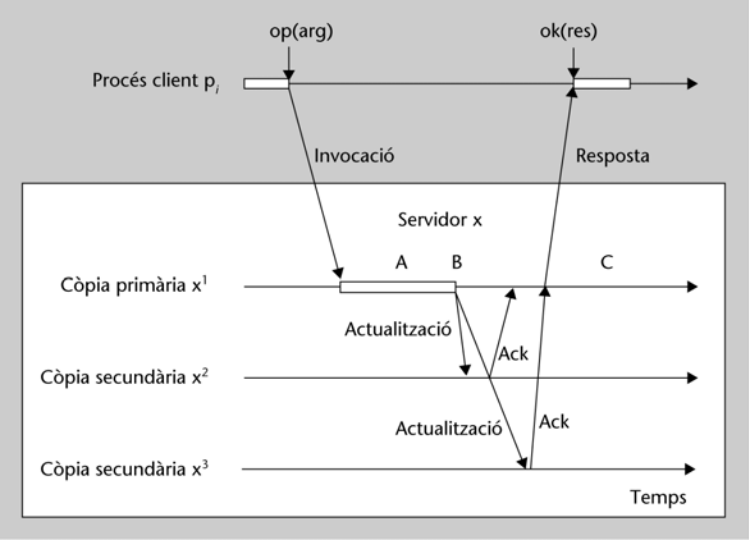


Figura 14. Model primari-secundari: el client es comunica només amb el primari. El primari es comunica al seu torn amb els secundaris.

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

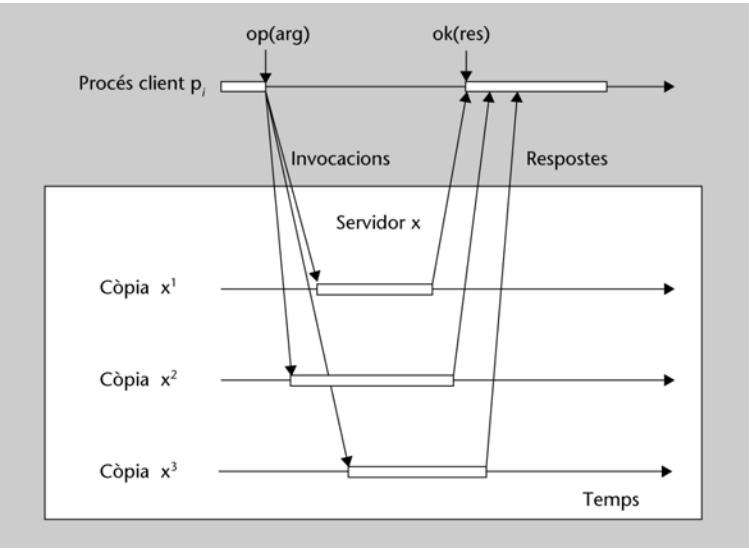


Figura 15. Model de reroducció activa: el client es comunica directament amb totes les còpies.

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 4.3, "Reproducció optimista", d'aquest mòdul didàctic.

3.2. 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:

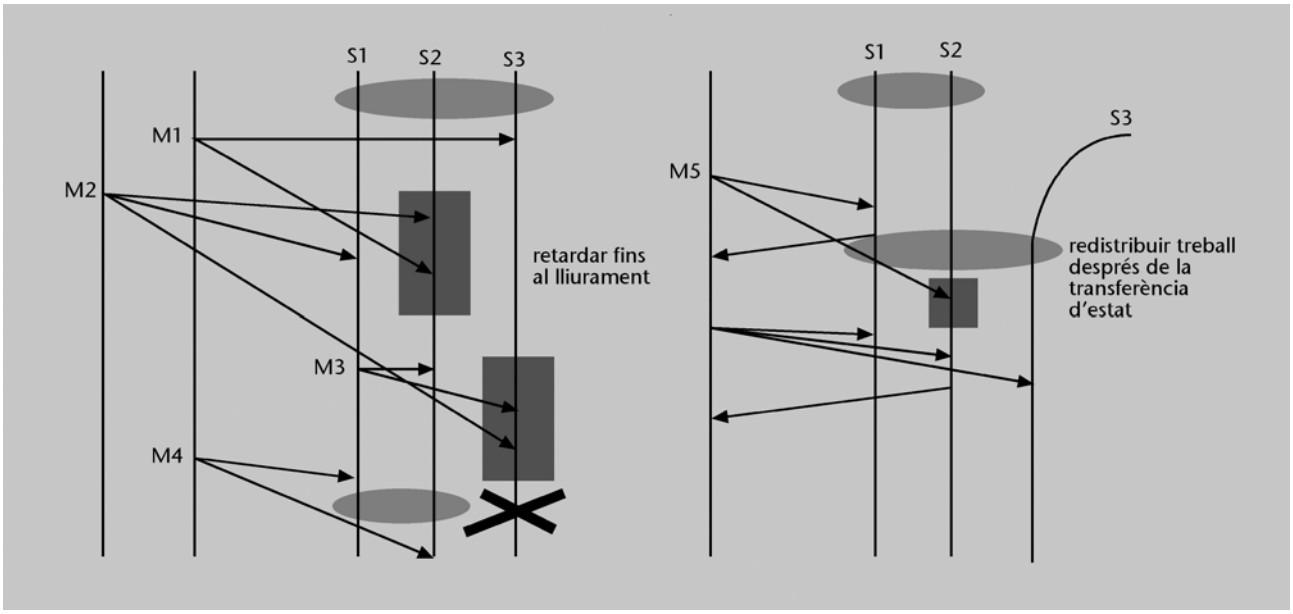


Figura 15. Problemes de lliurament de missatges a un grup de tres processos servidors

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.

3.3. 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.

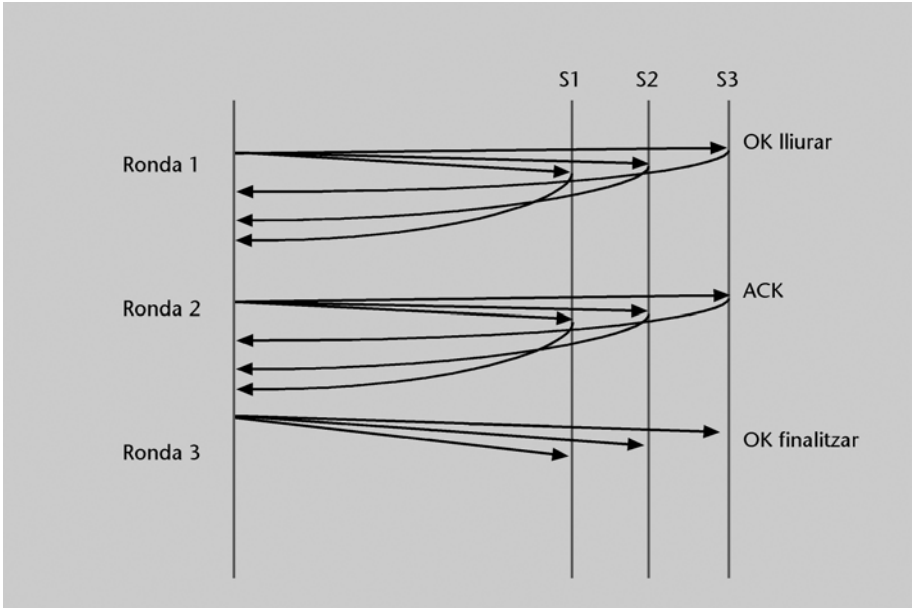


Figura 16. Transacció en tres fases

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.

4. Conceptes bàsics de reproducció en sistemes distribuïts

Bibliografia recomanada

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).

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 al 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 *single-master* i *multi-master*. Segons el segon paràmetre, en síncrones (*eager*) o asíncrones (*lazy*).

4.1. *Single* enfront de *multiple masters*

En el cas del *single-master* hi ha una còpia principal de cada objecte, que s'anomena *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 *multi-master* hi ha diversos magatzems que contenen una còpia primària d'un mateix objecte. Totes aquestes còpies es poden actualitzar

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.

L'aproximació *multi-master* 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.

4.2. Sistemes síncrons enfront de sistemes asíncrons

4.2.1. 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 *two-phase locking* –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. *Two-phase commit* 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 *single-master*. També n'hi ha *multi-master*, 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 3.1, "Comunicació fiable en grup", d'aquest mòdul didàctic.

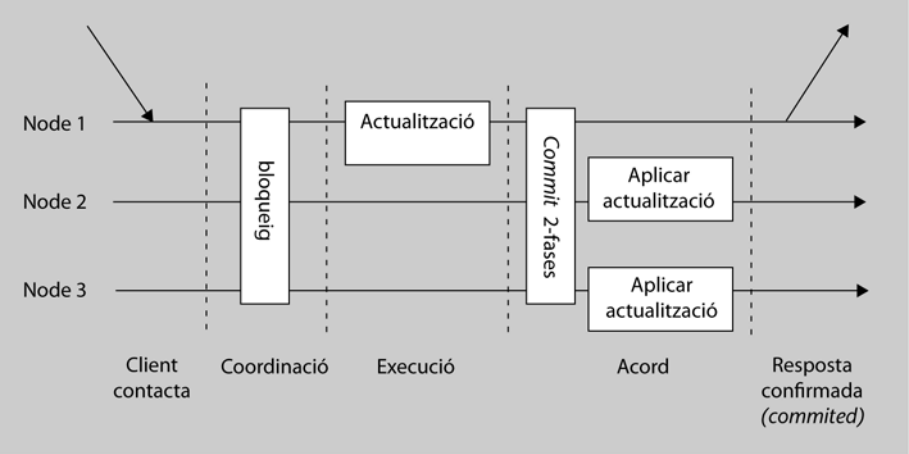


Figura 17. Exemple protocol síncron

La figura 17 mostra les diferents etapes que segueix un sistema distribuït síncron que utilitza un algorisme de *commit* de 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 *commit* de 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.

4.2.2. 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.

4.3. 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.

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

4.3.1. 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.

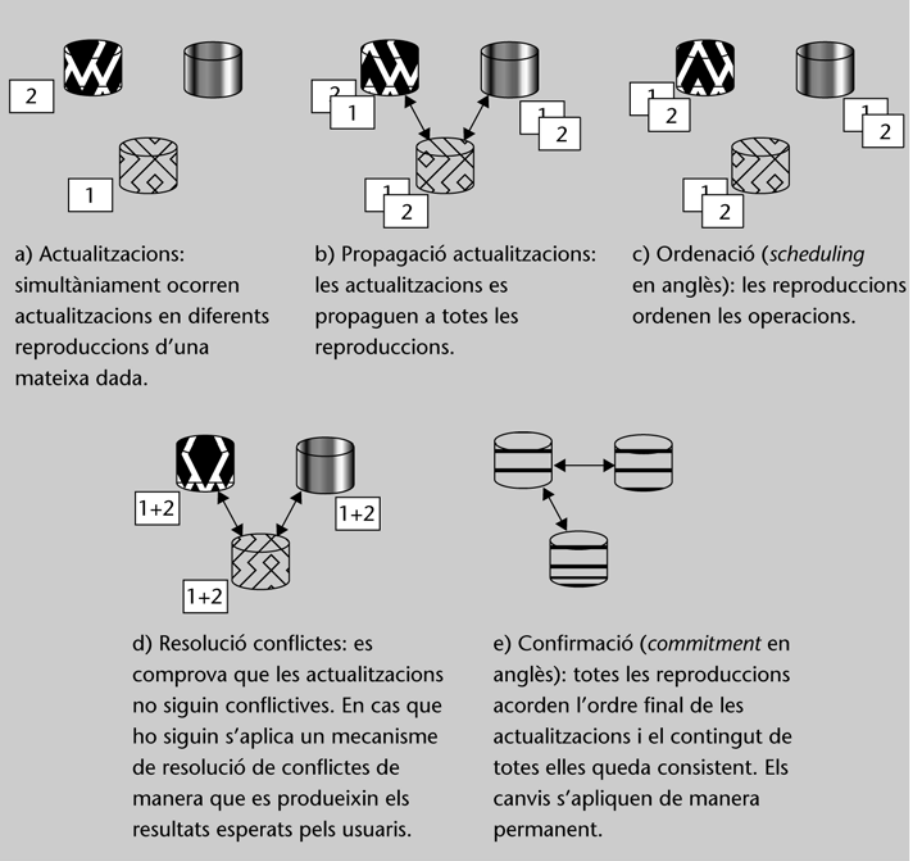


Figura 18

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 al final

d'aquest apartat. 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 actualitzacions. 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 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.

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.

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.

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.

4.3.2. 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 *single-master* 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 *multi-master* 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 en les seves còpies locals dels fitxers. Quan un usuari acaba, fa *commit* de la seva còpia local cap al magatzem. Si cap altre usuari no ha modificat el fitxer, el *commit* acaba immediatament. Si no, si les modificacions afecten línies diferents, les combina de manera automàtica i fa *commit* 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 *multi-master* 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.

5. Taules de *hash* distribuïdes (*distributed hash tables*)

Les taules de *hash* distribuïdes (DHT) són un tipus de sistemes distribuïts que permeten la localització eficient de dades per mitjà d'un índex descentralitzat i uniformement repartit entre els nodes del sistema. Se'ls denomina DHT perquè cada node és anàleg a una cel·la d'una taula *hash* que permet emmagatzemament i recuperació d'informació (PUT (K,V), GET(K)) de manera eficient en un entorn distribuït.

Les DHT sorgeixen en l'àmbit de la investigació en sistemes *peer-to-peer* com una evolució dels models d'índex centralitzat de Napster i totalment descentralitzat per inundació com Gnutella.

5.1. Història

En entorns acadèmics d'investigació sobre sistemes distribuïts al final dels noranta s'intenta resoldre el problema de la localització de recursos descentralitzada que resolgui els colls d'ampolla i problemes d'escalabilitat d'un índex centralitzat. Un problema clau és aconseguir que tots els nodes siguin alhora encaminadors d'informació i que es produeixi un repartiment balancejat de la càrrega entre tots els participants. En aquest context, els sistemes descentralitzats no estructurats com Gnutella es basen en sistemes d'encaminament per inundació que mitjançant algorismes de replicació adequats permeten la localització de les dades. S'anomenen *no estructurats* perquè les connexions entre els nodes són més o menys aleatòries sense que hi hagi així una estructura global en el sistema.

Tanmateix, aquest tipus de sistemes generen molt de trànsit innecessari i no asseguren que puguem localitzar un recurs encara que existeixi a la xarxa. Un primer pas per a solucionar aquestes limitacions és el donat per Freenet situant els índexs en zones més o menys conegudes de la xarxa. D'aquesta manera, un node pot encaminar cap a la zona on se suposa que es troba l'índex que està buscant. Aquest és un primer pas cap al denominat *greedy routing* (*routing* egoista o ansiós) pel qual cada node pot prendre una decisió local de cap a on encaminar partint del que es busca. Això suposa un gran avenç davant el model de Gnutella en el qual cada node no sap quina de les seves connexions ha d'utilitzar per a encaminar i per això ha de propagar en totes direccions.

Així, al començament del segle XXI, es proposen les primeres xarxes *peer-to-peer* estructurades com Chord, CAN, Pastry o Tapestry entre d'altres. En aquestes xarxes, els nodes es connecten entre ells seguint una estructura predeterminada com un anell, un hipercub o un arbre. Les estructures establertes asseguren que cada node necessita poques connexions (log N), que la xarxa té un diàmetre petit i que es pot encaminar informació d'un node a un altre en pocs salts (log N). A més, s'utilitza el *greedy routing* per la qual cosa cada node sap per on és més òptim enviar la informació.

Un descobriment clau que propicia l'aparició d'aquestes xarxes estructurades en el concepte de *hashing* consistent (*consistent hashing*). Malgrat que no va ser pensat en principi per a entorns descentralitzats *peer-to-peer*, aquesta contribució va servir d'inspiració per a resoldre el problema del repartiment balancejat dels recursos (*load balancing*) entre els nodes de la xarxa. Així, el *hashing* consistent utilitza una funció determinística que assigna identificadors en un rang de manera uniforme, assegurant així el repartiment equitatiu de la càrrega.

Chord: http://pdos.csail.mit.edu/chord/
Ion Stoica; Robert Morris; David Karger; M. Frans Kaashoek; Hari Balakrishnan (2001, agost). *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. ACM SIGCOMM 2001, San Diego, CA, pàg. 149-160.

CAN
Sylvia Ratnasamy; Paul Francis; Mark Handley; Richard Karp; Scott Shenker. *A Scalable Content-Addressable Network*. In Proceedings of ACM SIGCOMM 2001.

Pastry: http://research.microsoft.com/~antr/Pastry/
A. Rowstron; P. Druschel (2001, novembre). "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pàg. 329-350.

Chord

Estudiarem el cas de Chord com a exemple paradigmàtic i clàssic de les xarxes *peer-to-peer* estructurades o també denominades *taules de hash distribuïdes* (DHT). Com veiem en la figura, en Chord els nodes formen un anell amb identificadors compresos entre [0 i 2^M] mòdul 2^M essent M el nombre de bits de l'identificador. En aquest exemple, M és 6 per la qual cosa els identificadors estan compresos entre el 0 i el 63.

L'identificador de cada node s'obté utilitzant una funció de *hashing* **consistent** que assegura que els identificadors estaran uniformement repartits. Un exemple és la funció SHA (*secure hash algorithm*), que genera un identificador de 160 bits que després es poden truncar per manejar identificadors més reduïts (la mida sol estar entre 32 i 160 bits, cosa que permetrà d'aquesta manera un gran nombre de nodes).

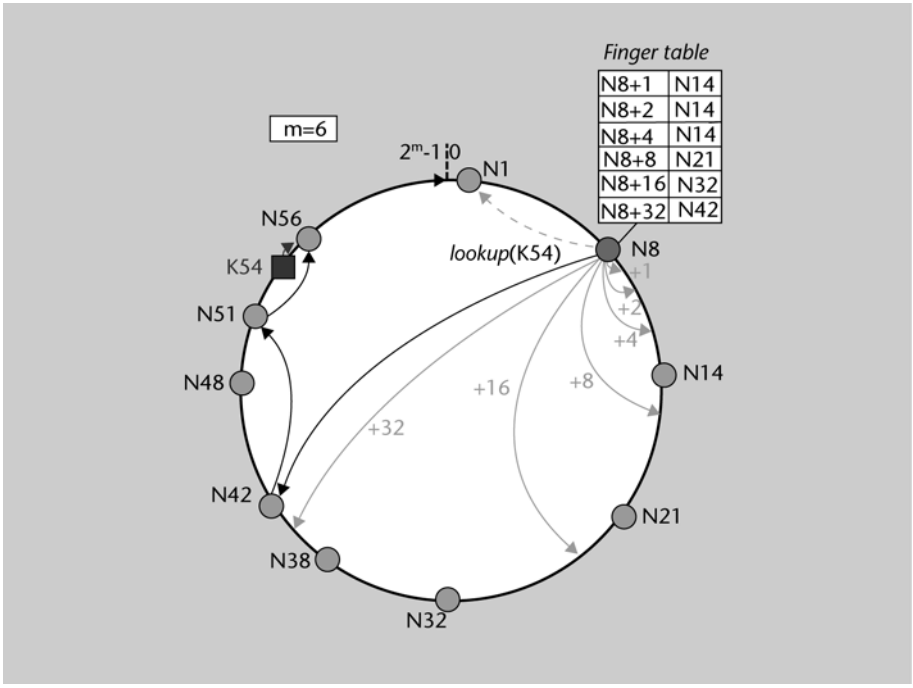


Figura 19

Quan un node entra a la xarxa i obté un identificador, s'ha de situar a la zona de l'anell adequada entre el predecessor i el successor del seu identificador. És el que es coneix com a *procés d'entrada a la xarxa* (*join*, *bootstrapping*) i ho pot fer a través de qualsevol node. Per exemple, si en aquesta xarxa entra el node 4, se situarà entre el node N1 i l'N8.

En Chord, cada node és responsable de les claus o recursos situats entre el seu predecessor i ell mateix (inclòs). D'aquesta manera, l'N8 és responsable de les claus 2, 3, 4, 5, 6, 7 i 8. Quan s'insereix un recurs a la xarxa, s'aplica la funció de *hashing* consistent per a obtenir la seva clau i així s'assegura que els recursos estan uniformement repartits. En calcular el *hashing*, per exemple del nom del fitxer, (SHA(ASD.doc)) obtenim una clau que ens diu, a més, quin node de la xarxa n'és responsable. D'aquesta manera, en inserir la clau 17 (PUT(17)) s'emmagatzemarà en el successor de 17, que és el node N21.

Si cada node només tingués els enllaços de successor, l'encaminador seria *O(N)*. És a dir, en el pitjor dels casos si som en el node zero i només podem anar cap a la dreta, per a arribar al node 63 haurem de passar per tots els nodes de l'anell. Per a millorar això, Chord té una taula d'enllaços a altres nodes denominada *finger table*. Aquesta taula té log N enllaços essent $N = 2^M$, per la qual cosa en el nostre exemple la mida és 6. En la *finger table*, cada node té connexions logarítmiques a altres nodes seguint el patró: $n + (2^i$ mòdul $2^M)$, on n és l'identificador del node, i i correspon a cada entrada de la *finger table* (exemple, 0...6). Com veiem en l'exemple, el node N8 hauria de tenir connexions als nodes 9, 10, 12, 16, 24 i 40 si la xarxa fos completa. En no estar-ho, es connectarà als nodes successors d'aquests identificadors que si es troben a la xarxa.

L'encaminament en Chord és ansiós (*greedy routing*), en el sentit de les agulles del rellotge (*clock-wise*) i se sol denominar *encaminament basat en claus* (*key based routing*, KBR). Quan des d'un node s'intenta localitzar un recurs amb una clau concreta, el node busca en la seva *finger table* la connexió que més l'apropa a la seva destinació. Per exemple, si en l'N8 es busca la clau 54 (GET(54)), s'enviarà la sol·licitud al node N42, que és el que més s'aproxima a la clau 54. Aquest, al seu torn, l'enviarà al 51, que finalment preguntarà al 56, que és el responsable de la clau 54. Com podem observar, en molt pocs salts trobem qualsevol recurs de manera eficient i així s'eviten les inundacions del model Gnutella.

5.2. Discussió

Els avantatges clau de Chord i altres sistemes similars són la descentralització, l'escalabilitat i el balanceig de la càrrega entre els participants.

> Les DHT permeten construir sistemes completament descentralitzats en els quals tots els nodes són iguals entre ells i altament escalables a milions de participants. A més, l'avantatge essencial d'aquests sistemes és que permeten localitzar informació amb un ordre de salts logarítmic (O(logN)) i mantenint un petit nombre de connexions a altres nodes (taules d'encaminament d'ordre constant O(1) o logarítmic (O(logN)).

Els principals problemes són el manteniment del sistema, l'entrada i sortida de nodes constant (*churn*), la proximitat en xarxa i la limitació a recerques exactes (*exact match*).

En Chord, per exemple, és necessari mantenir en cada node una llista de successors de mida logarítmica per a assegurar que no es trenca l'anell. A més, és necessari un protocol de manteniment d'enllaços que els actualitzi correctament a mesura que entren o surten nodes. Això implica un cost que limita la seva aplicació per a sistemes molt dinàmics i encara s'està estudiant en cercles d'investigació acadèmics.

A més, en aquestes xarxes descentralitzades és essencial tenir en compte la proximitat en xarxa o les latències entre nodes. Si un DHT necessita pocs salts, però aquests passen per nodes molt llunyans en termes de latència (Espanya-Japó-Austràlia-França), el sistema serà molt ineficient. Per això, molts DHT han intentat organitzar-se partint d'informació de proximitat com CAN, Pastry o CORAL per a millorar l'encaminament. Tanmateix, això continua essent un problema obert que es continua investigant en cercles científics.

D'altra banda, les DHT ofereixen un sistema de recerca exacta GET(K) que els fan menys flexibles que altres sistemes amb recerques obertes o més comple-

xes. A més, quan un node se'n va, les seves claus han de migrar al node successor que ara és el responsable d'aquestes claus. Per això s'han de replicar aquestes claus per a fer el sistema tolerant a fallades i assegurar que les claus no es perden. Diversos treballs d'investigació continuen estudiant avui en dia com s'han d'adaptar o construir aplicacions sobre els DHT que permetin recerques més avançades.

De qualsevol manera, les DHT han suposat una autèntica revolució en entorns descentralitzats i ja se'ls considera la tercera generació de sistemes *peer-to-peer* després de Napster i Gnutella. Gràcies a les considerables qualitats de les DHT s'han desenvolupat molts serveis distribuïts que es construeixen sobre aquestes.

Exemples de serveis...

... serien els sistemes de fitxers distribuïts, els sistemes de compartició d'arxius P2P, l'emmagatzemament cooperatiu en web, serveis de *multicast* o *anycast* a nivell d'aplicació, o fins i tot serveis de noms i de DNS descentralitzats.

Entre les aplicacions existents avui en dia que es basen en DHT destaquem les següents:

- **eMule KAD:** el programa de descàrrega de fitxers P2P eMule utilitza una DHT denominada Kademlia (KAD) com a evolució del sistema. KAD millora la localització de fonts a la xarxa i la fa menys fràgil a atacs sobre els servidors centrals.
- **BitTorrent/Azureus:** els clients de bitTorrent utilitzen una implementació del DHT Kademlia per a aconseguir la descentralització del component *tracker*. Això els fa menys vulnerables a la caiguda d'aquest.
- **OpenDHT:** és un DHT (Tapestry) públicament accessible que ofereix les funcionalitats de PUT i GET a través d'interfícies Sun RPC i XMLRPC. Diverses aplicacions s'han basat en OpenDHT com middleware base. http://opendht.org/

En conclusió, avui en dia les DHT es consideren un substrat distribuït essencial sobre el qual construir serveis descentralitzats. La seva integració amb mecanismes de proximitat en xarxa d'àmbit Internet permetran aplicacions eficients en el context de la computació Grid de gran escala o en xarxes de distribució de continguts (CDN). Finalment, cal destacar que les DHT no són la solució ideal a tots els problemes en sistemes distribuïts descentralitzats (*silver bullet*). Per exemple, en entorns mòbils amb una gran dinamicitat i inestabilitat dels nodes, els DHT no són la solució adequada i sí que ho són, en canvi, els sistemes desestructurats o híbrids amb menys cost de manteniment de la xarxa.

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.

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.

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

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.

Per acabar el mòdul, hem introduït uns mecanismes les taules de *hash* distribuïdes (DHT), que són un mecanisme distribuït que permet la localització eficient de dades a través d'un índex descentralitzat i uniformement repartit entre tots els nodes del sistema. Els nodes es connecten entre ells seguint una estructura predeterminada com ara un anell, un hipercub o un arbre. Aquestes estructures asseguren que cada node necessita poques connexions (log N), que la xarxa té un diàmetre petit i que es pot fer arribar informació d'un node a un altre en pocs salts (log N).

També hem vist quins són els seus principals problemes: mantenir el sistema quan hi ha entrades i sortides constants de nodes i la limitació a les cerques exactes. A més, les implementacions que hi ha avui en dia d'aquest tipus de mecanismes no tenen en compte la proximitat entre els nodes.

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 la 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).

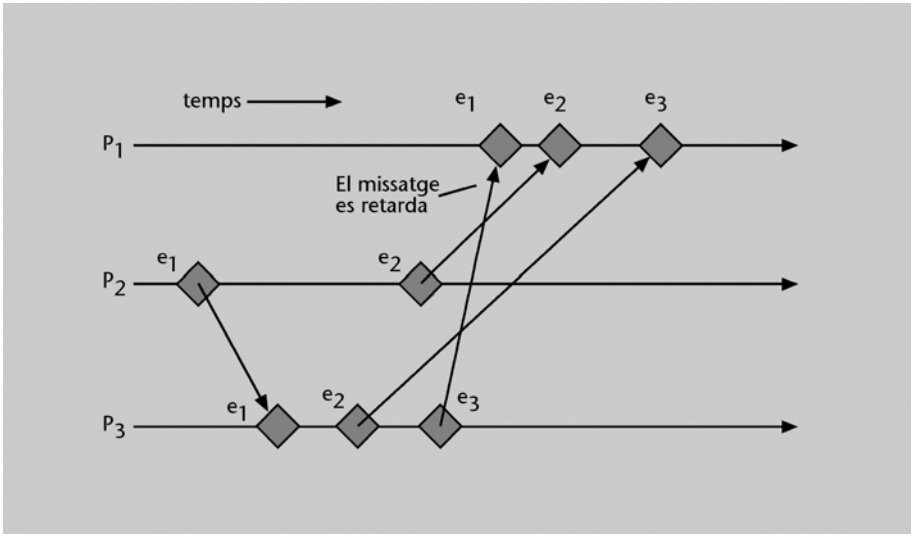
2. Sincronitzeu el rellotge del meu PC usant algun programa per a sincronitzar rellotges. Anoteu les variacions diàries de temps que es puguin mesurar i compareu-les amb l'hora d'un rellotge de polsera durant diversos dies.
Es poden usar programes com Dimension 4:
http://www.thinkman.com/dimension4/ 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.

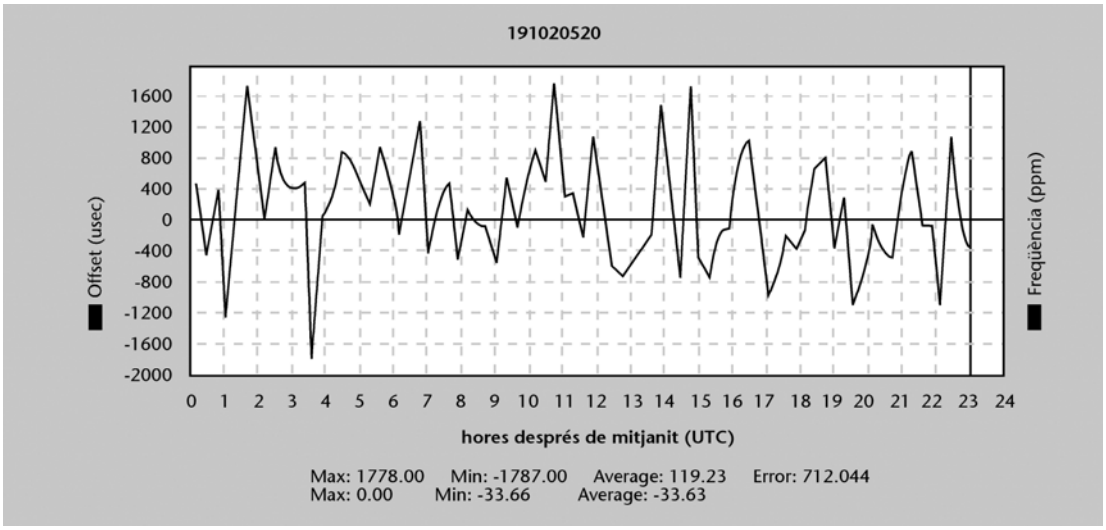
6. Feu una taula amb les diferències fonamentals entre servidors repetits organitzats com a primari-secundari i reproducció activa.

7. Per a la xarxa Chord de la figura, feu el següent:
a) Mostreu les taules d'encaminament dels nodes N8, N12, N1 i N56 quan entri l'N12.
b) Mostreu els salts fets quan en el node N56 volem obtenir la clau K13.
c) Mostreu els salts fets quan en el node N56 volem obtenir la clau K37.
d) Mostreu els salts fets quan en el node N56 volem obtenir la clau K21.

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.

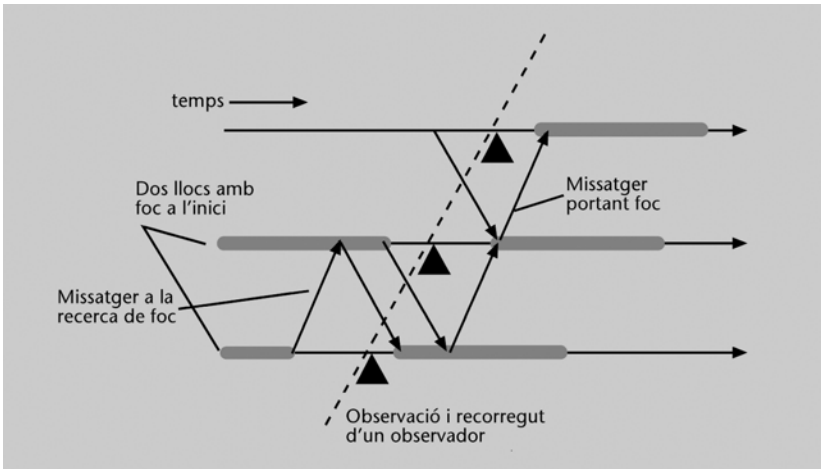


3. Per exemple, el codi següent s'obté visitant la pàgina web d'http://www.apache.org amb el comandament Telnet al port 80 del servidor i demanant informació sobre la pàgina principal mitjançant el comandament HEAD d'HTTP (en negreta el que s'escriu):

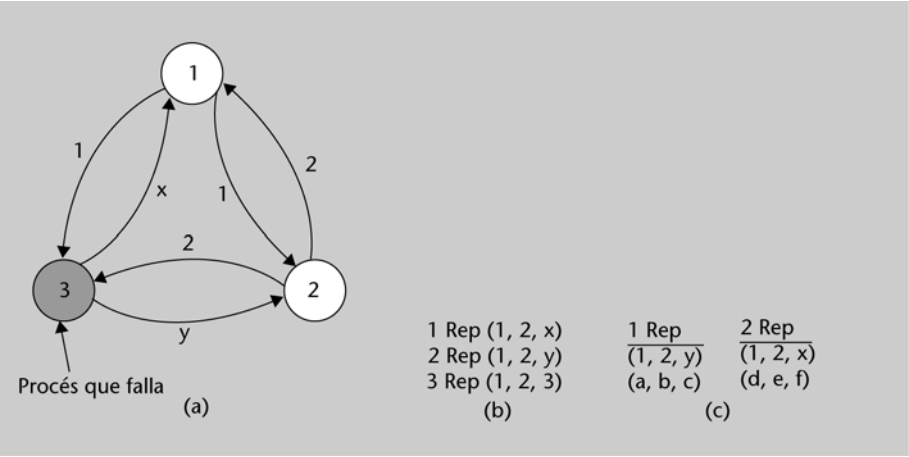
telnet www.apache.org 80
HEAD / HTTP/1.1
Host: www.apache.org

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)

7.

a)
N8+1 12
N8+2 12
N8+4 12
N8+8 21
N8+16 32
N8+32 42

N12+1 14
N12+2 14
N12+4 21
N12+8 21
N12+16 32
N12+32 48

N1+1 8
N1+2 8
N1+4 8
N1+8 12
N1+16 21
N1+32 38

N56+1 1
N56+2 1
N56+4 1
N56+8 1
N56+16 8
N56+32 32

b) 56->8->12->14

c) 56->32->38

d) 56->8->21

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.

master Vegeu **primari**.

multi master *m* Sistema que suporta diversos primaris per objecte.

primari *m* Reproducció que té la capacitat d'acceptar actualitzacions.
en 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 *m* Sistema que suporta un primari per objecte.

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.

Lua, E. K. i altres (2005). "A survey and comparison of peer-to-peer overlay network schemes". *IEEE Communications Surveys&Tutorials* (vol. 7, núm. 2).
És un article en què trobareu un resum de com funcionen les DHT més populars, així com una comparativa entre elles.