Arquitectura d'aplicacions web

Leandro Navarro Moldes

P07/M2006/02842

UOC
www.uoc.edu

Índex

Introducció

En aquest mòdul didàctic es parlarà de les maneres d'organitzar aplicacions web i de com fer que puguin funcionar tot i estar subjectes al comportament caòtic i imprevisible d'Internet.

Primer es caracteritza la demanda d'aquests serveis i com mesurar-la en una situació real. Després es descriuen les formes de construir i l'evolució dels serveis web (CGI, *servlets*, servidors d'aplicacions i servidors web) analitzant els casos de diversos servidors web; per acabar parlant de formes distribuïdes de servei: servidors intermediaris *proxy-cache*, xarxes de distribució de continguts, aplicacions orientades a serveis i computació sota demanda.

La manera d'adquirir els coneixements passa per fer 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 les característiques de la demanda que ha de satisfer un servidor web.

2. Conèixer les diverses maneres d'organitzar una aplicació web i els models que existeixen, segons els diversos criteris.

3. Conèixer les característiques i el funcionament de cada model.

4. Poder triar la millor opció en cada situació i valorar les implicacions del muntatge que cal fer.

1. Característiques de la demanda de pàgines web

El trànsit de web és el responsable d'un bon percentatge del trànsit d'Internet. Aquesta tendència ha anat creixent gradualment des que va aparèixer el web (protocol HTTP) i avui en dia el trànsit HTTP predomina respecte de la resta de protocols, i hi ha una gran població d'usuaris "navegants" que poden generar una quantitat immensa de peticions si el contingut és interessant. L'organització d'un servei web connectat a Internet reclama de tenir en compte les característiques de la demanda que pugui haver d'atendre.

Esculls de corall i trànsit a Internet

L'organització CAIDA (www.caida.org) es dedica a l'anàlisi del trànsit a Internet i ha desenvolupat una eina anomenada *Coral Reef* que pren traces del trànsit d'un enllaç. Amb aquesta, el 1998 van fer un estudi de trànsit per protocols al nucli de la xarxa del proveïdor MCI.

L'article que ho descriu es va presentar en la conferència Inet 98, titulat: "The nature of the beast: recent traffic measurements from an Internet backbone".

Les gràfiques adjuntes s'han obtingut d'aquestes mesures.

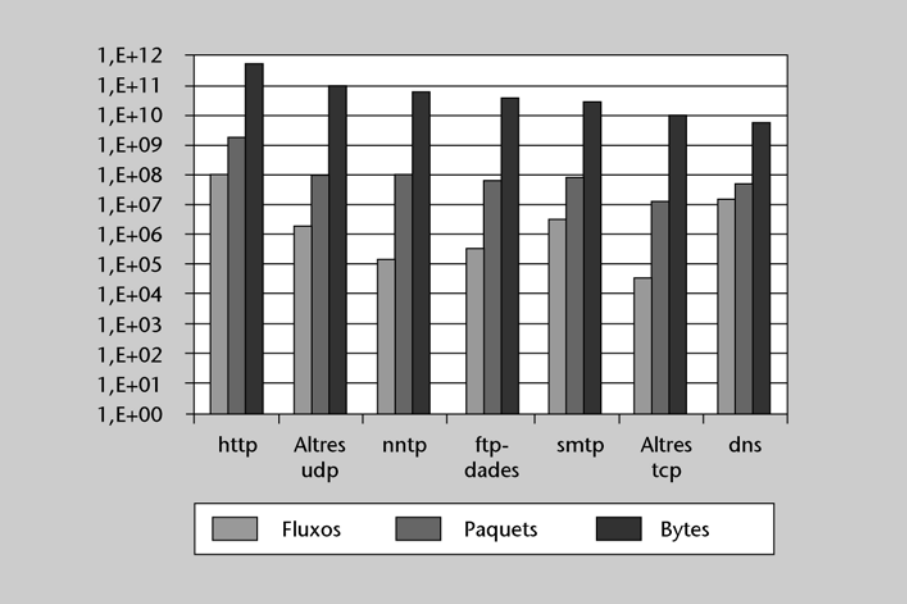

Figura 1. Volum de trànsit en escala logarítmica de fluxos, paquets i bytes intercanviats durant vint-i-quatre hores en un enllaç del nucli de la xarxa d'MCI/Worlcom (1998), organitzat per protocol.

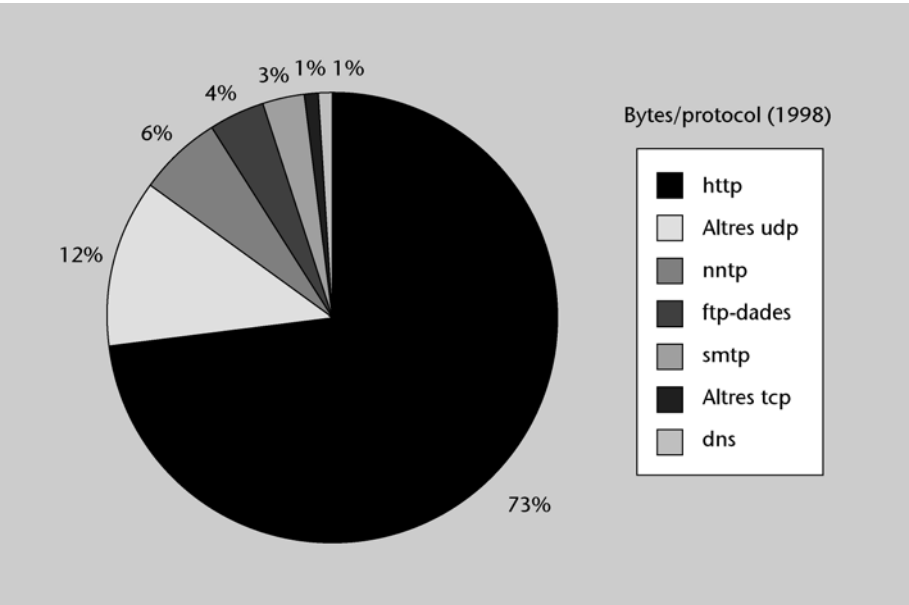

Figura 2. Percentatge de trànsit en bytes de cada protocol respecte al total mesurat. Es pot apreciar millor que en la figura 1, en escala logarítmica, que el percentatge de trànsit web domina la resta (73% del total).

D'altra banda, el web (HTTP) és un servei molt reclamat per tot tipus d'organitzacions per tal de publicar informació, com es pot veure en la tendència de creixement del nombre de servidors web a Internet, que ha estat exponencial tal com mostra la figura 3.

La cronologia d'Internet

Un calendari dels esdeveniments relacionats amb Internet des de 1957 fins avui.

Ho podeu veure en l'adreça següent: http://www.zakon.org/robert/internet/timeline/

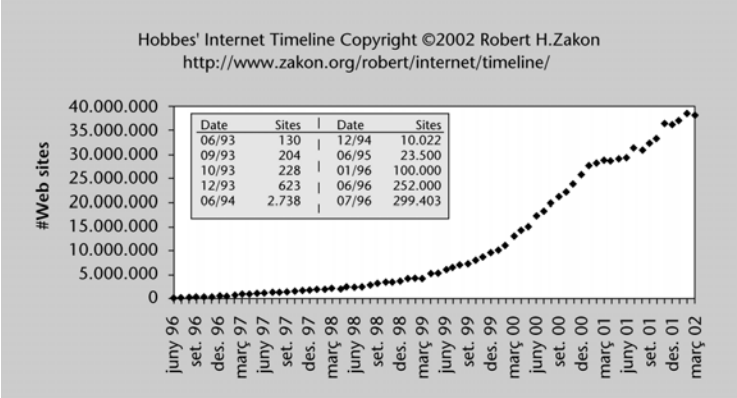

Figura 3. Creixement del nombre de llocs web durant els últims sis anys

La popularitat dels servidors web també és molt variable. Un mateix lloc web pot rebre molt poques visites durant molt de temps i, de sobte, rebre diverses vegades més peticions de les que pot servir: és un trànsit a ràfegues.

Flash crowd

Un conte de ciència-ficció de Larry Niven (1973) va predir que una conseqüència d'un mecanisme de teletransport barat seria que grans multituds es materialitzarien instantàniament als llocs amb notícies interessants. Trenta anys després el terme s'usa a Internet per a descriure els pics de trànsit web quan un determinat lloc web esdevé sobtadament popular i és visitat de manera massiva.

També es coneix com a efecte "slashdot" o efecte "/.", que es dóna quan un lloc web resulta inaccessible per causa de les nombroses visites que rep quan apareix en un article del lloc web de notícies slashdot.org (en castellà barrapunto.com).

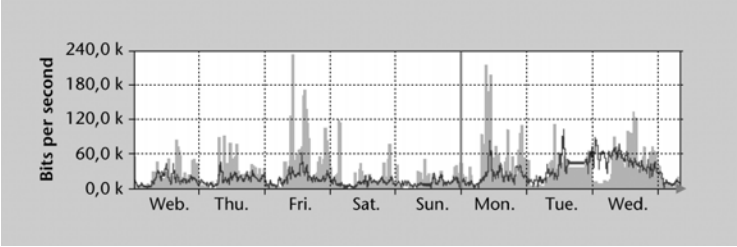

Figura 4. Evolució del trànsit entrant i sortint d'un lloc web típic durant una setmana. Podeu observar la gran variació horària i la reducció de trànsit durant el cap de setmana.

Un servidor pot rebre allaus sobtades de trànsit. Per exemple, per les estadístiques del següent servidor web sabem que, després de ser anunciat a la pàgina de notícies slashdot.org va patir un excés de visites tan alt que el servidor es va bloquejar:

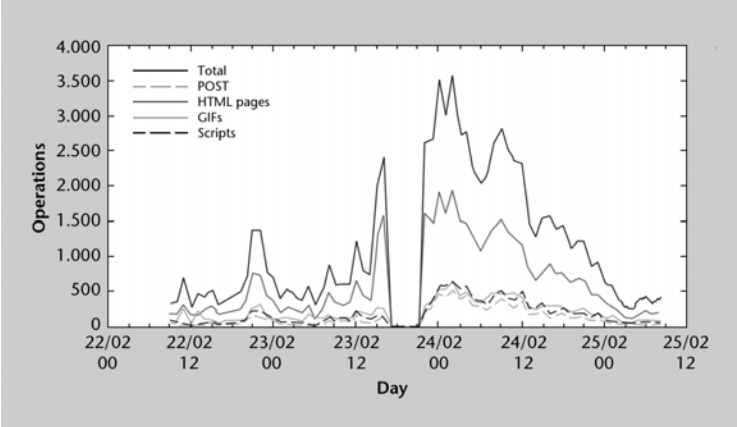

Figura 5. Peticions web per hora, servides per http://counter.li.org durant tres dies. Es pot veure que mentre el nombre habitual d'operacions (peticions web) estava per sota de 500, va pujar ràpidament a unes 2.500, fet que va provocar la fallada del sistema. Després de reconfigurar-lo, va estar suportant durant unes dotze hores al voltant de 3.000 peticions/hora per a baixar posteriorment a valors normals. La història completa és a l'adreça: http://counter.li.org/slashdot/

Un servidor web pot tenir milers de documents i, tanmateix, rebre la majoria de peticions per un únic document. En molts casos, la popularitat relativa entre diversos llocs web o entre diverses pàgines d'un cert lloc web es regeix per la llei de Zipf (George Kingsley Zipf (1902-1950)) que diu:

> La freqüència de succés d'un esdeveniment concret (P) com a funció del rang (i) quan el rang és determinat per la freqüència de succés, és una funció potencial $P_i \sim 1/i^a$, amb l'exponent *a* proper a la unitat.

L'exemple més famós és la freqüència de paraules en anglès. En 423 articles de la revista *Time* (245.412 paraules), *the* és la que més apareix (15.861), *of* en segon lloc (7.239 vegades), *to* en tercer lloc (6.331 vegades), amb la resta formen una llei potencial amb un exponent proper a 1.

Una distribució de popularitat Zipf forma una línia recta quan es dibuixa en una gràfica amb tots dos eixos en escala logarítmica, que resulta més fàcil de veure i comparar que la gràfica en escala lineal, tal com es pot veure en la figura següent:

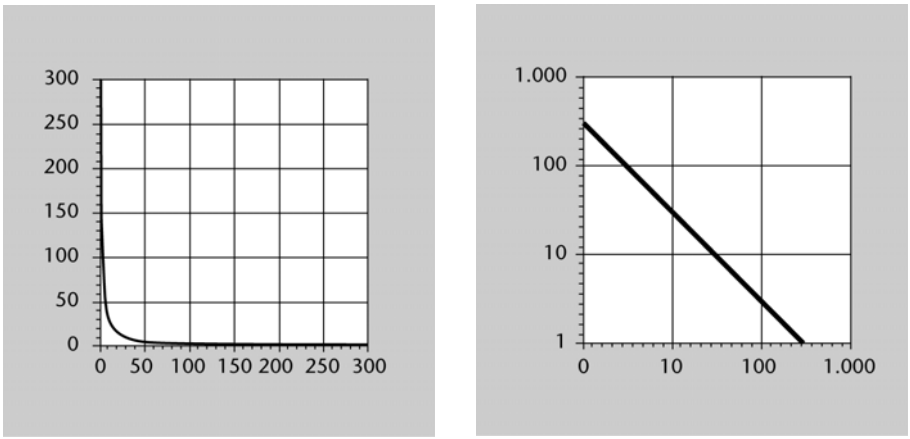

Figura 5. Una distribució de popularitat (casos ordenats per popularitat a l'eix *x*, valor de popularitat a l'eix *y*) que segueix la llei de Zipf a escala lineal queda "enganxada als eixos": molt pocs casos tenen molta popularitat i molts casos en tenen molt poca. Per això se sol representar en escales logarítmiques (gràfica doble logarítmica: els dos eixos en escala logarítmica).

Molts estudis mostren que les visites de pàgines web segueixen una distribució de Zipf. La figura següent mostra les visites a www.sun.com durant un mes del 1996. La pàgina principal va rebre gairebé 1 milió de visites, mentre que la pàgina de la posició 10.000 de popularitat només va rebre una visita aquell mes. La gràfica de visites segueix la corba de Zipf excepte per als valors menys populars, que segurament es deu al fet que el període d'observació no va ser prou llarg.

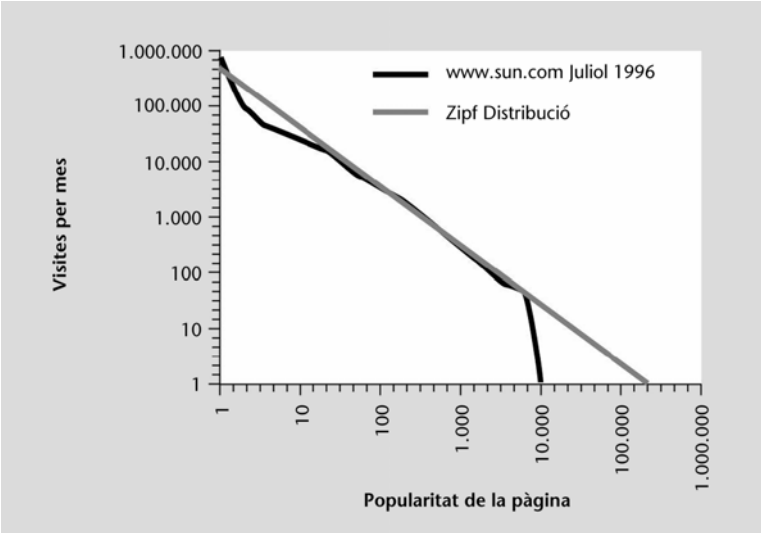

Figura 6. Nombre de visites de les pàgines de www.sun.com ordenades per popularitat. Es pot veure com s'ajusta a una distribució de Zipf.

Com a resum, diversos estudis del trànsit web contribueixen a definir un perfil típic o regles a ull del web –segons M. Rabinovich i O. Spatscheck (2002). *Web Caching and Replication.* Addison Wesley. ISBN: 0201615703:

– La mida mitjana d'un objecte és de 10-15 kbytes, i la mitjana de 2-4 kbytes. La distribució es decanta clarament cap a objectes petits encara que es troba una quantitat gens menyspreable d'objectes grans (de l'ordre de Mbytes).

– La majoria d'accessos al web són per objectes gràfics, seguits dels documents HTML. L'1-10% són per objectes dinàmics.

– Una pàgina HTML inclou de mitjana deu imatges i múltiples enllaços a altres pàgines.

– Un 40% de tots els accessos són per a objectes que es considera que no es poden inspeccionar.

– La popularitat d'objectes web és molt diversa: una petita fracció d'objectes és la responsable de la majoria d'accessos, seguint la llei de Zipf.

– El ritme d'accés per a objectes estàtics és molt superior al ritme de modificació.

– En una escala de temps inferior al minut el trànsit web és a ràfegues, per la qual cosa valors mesurats amb mitjanes durant algunes desenes de segon són molt poc fiables.

– Un 5-10% d'accessos al web es cancel·len abans de finalitzar.

– Gairebé tots els servidors utilitzen el port 80.

Cada lloc web és una mica diferent i com que un servidor web és un sistema complex i, per tant, difícil de modelar, resulta convenient de fer experiments

tant per veure com els nivells de càrrega (peticions de pàgines web) creixents afecten el nostre servidor, com per observar periòdicament el comportament d'un servidor web analitzant els diaris (*logs*) que pot generar.

Per a provar el rendiment d'un servidor web, normalment es fa servir algun programa que, instal·lat en un altre computador, generi un ritme de peticions equivalents per a mesurar l'efecte de les visites simultànies des de diversos clients. El ritme de peticions es pot configurar d'una manera lleugerament diferent per a cada eina, però l'objectiu és ser estadísticament equivalent a la demanda real que el servidor pugui experimentar durant el seu funcionament normal. Això rep el nom de *càrrega sintètica*.

Hi ha moltes eines. A continuació es descriuen breument tres eines populars i gratuïtes:

Visites web sobre generadors de càrrega

Apache Jmeter: jakarta.apache.org/jmeter/

Surge: www.cs.bu.edu/faculty/crovella/links.html

– *Microsoft web application stress* (WAS) és una eina de simulació per a Windows dissenyada per mesurar el rendiment d'un lloc web. Té en compte les pàgines generades dinàmicament (ASP) en un servidor web de Microsoft.

– *Apache JMeter*, una aplicació Java per a mesurar el rendiment de documents i recursos estàtics i dinàmics (arxius, *servlets*, *scripts* Perl, objectes Java, consultes de bases de dades, servidors FTP, etc.), que simula diferents tipus de càrrega extrema de la xarxa, del servidor o d'un objecte concret.

– *Surge* de la Universitat de Boston: una aplicació Java que genera peticions web amb característiques estadístiques que simulen amb molta precisió la demanda típica d'un servidor web.

Durant el funcionament normal del servidor és convenient de supervisar la demanda i el rendiment del servei per detectar la degradació del servei (respon molt lentament per excés de peticions o trànsit) o situacions crítiques (sobrecàrrega: no respon).

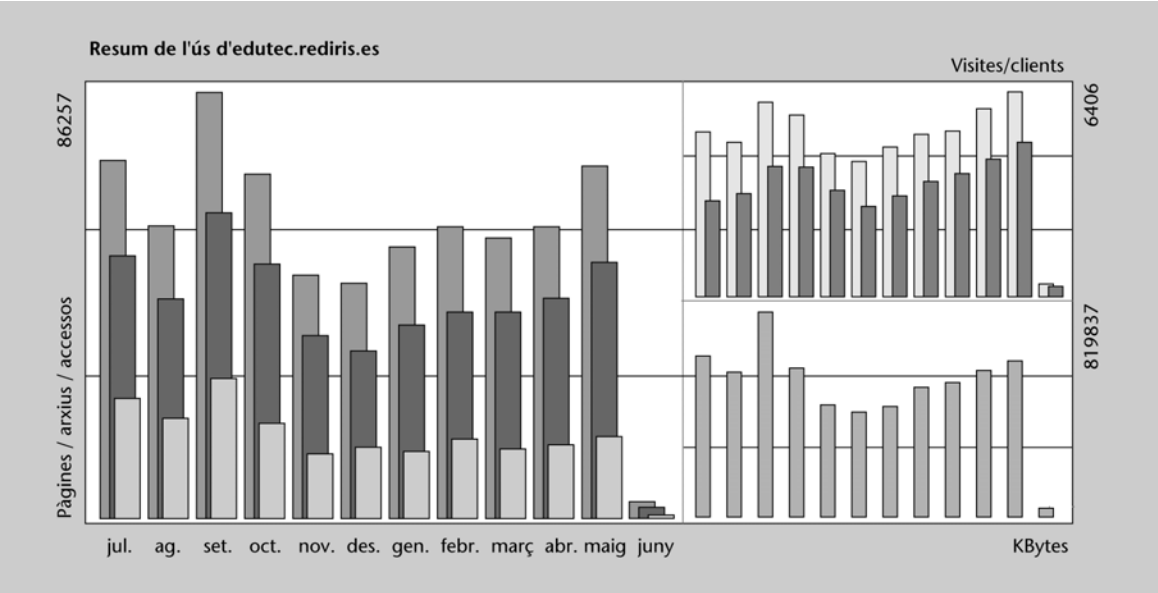

Figura 7. Una de les moltes gràfiques que genera una eina d'estadístiques web popular i gratuïta anomenada *webalizer*, que es pot trobar en l'adreça http://www.mrunix.net/webalizer/ i que facilita molt l'anàlisi de l'activitat d'un servidor web.

Les eines de visualització i anàlisi d'activitat del servidor es basen en el fet que gairebé tots els servidors són capaços de generar arxius històrics de l'activitat del servidor (coneguts com a *diaris*, en anglès *logs*) en un format conegut com a *common log format* o CLF.

En CLF cada línia (de vegades anomenada *entrada*) registra una petició rebuda pel servidor. La línia està formada per diversos elements separats per espai:

màquina ident usuariautoritzat data petició estat bytes

Si una dada no té valor, es representa per un guió (-). El significat de cada element és el següent:

– Màquina: el nom DNS complet o la seva adreça IP si el nom no està disponible.

– Ident: si està activat, la identitat del client tal com ho indica el protocol identd. Pot no ser fiable.

– UsuariAutoritzat: si es va demanar un document protegit per contrasenya, correspon al nom de l'usuari utilitzat a la petició.

– Data: la data i hora de la petició en el format següent:

```
date = [dia/mes/any:hora:minut:segon zona]
dia = 2*digit
mes = 3*letter
any = 4*digit
hora = 2*digit
minut = 2*digit
segon = 2*digit
zona = ('+' | `-') 4*digit
```

– Petició: l'URL sol·licitat pel client, delimitat per cometes (").

– Estat: el codi de resultat de tres dígits tornat al client.

– Bytes: el nombre de bytes de l'objecte servit, sense incloure capçaleres.

Aquest format és adequat per a registrar la història de les peticions, però no conté informació útil per a mesurar el rendiment del servidor. Per exemple, no indica el temps transcorregut en servir cada URL.

Per a permetre de construir un format d'entrada de diari (*log*) que contingui la informació necessària, es pot definir un format particular que pot contenir una altra informació. A continuació es mostra una llista de les variables que el servidor web Apache pot guardar (mod_log).

Nom	Descripció de la variable
%a	Adreça IP remota
%A	Adreça IP local
%B	Bytes enviats, excloent les capçaleres HTTP
%b	Bytes enviats, excloent les capçaleres HTTP. En format CLF: un '-' en lloc d'un 0 quan no s'ha enviat cap byte
%c	Estat de la connexió quan la resposta s'acaba 'X' = connexió avortada abans d'acabar la resposta '+' = connexió que pot quedar activa després d'haver enviat la resposta '-' = connexió que es tancarà després d'haver enviat la resposta
%{NOM}e	El contingut de la variable de la variable d'entorn NOM
%f	Nom del fitxer
%h	Nom de la màquina remota
%H	El protocol de la petició
%{Nom}i	El contingut de la capçalera o capçaleres "Nom:" de la petició enviada al servidor
%l	Usuari remot (de identd, si ho proporciona)
%m	El mètode de la petició
%{Nom}n	El contingut de la "nota" "Nom" des d'un altre mòdul
%{Nom}o	El contingut de la capçalera o capçaleres "Nom:" de la resposta
%p	El port del servidor servint la petició
%P	L'identificador del procés fill que va servir la petició.
%q	El text d'una consulta o *query string* (precedit de ? si la consulta existeix, sinó un text buit)
%r	Primera línia de la petició
%s	Estat de peticions que van ser readreçades internament, l'estat de la petició original - %>s per al de la darrera
%t	Temps (data) en format de LOG (format estàndard anglès)
%{format}t	El temps (hora), en el format especificat, que s'ha d'expressar en format strftime (possiblement localitzat)
%T	El temps de servei de la petició, en segons
%u	Usuari remot (d'autenticació; pot ser incorrecte si l'estat de la resposta %s és 401)
%U	La part de camí (*path)* de l'URL, sense incloure el text de la consulta (*query string*).
%v	El nom original o *canònic* del servidor depenent de la petició
%V	El nom del servidor segons el valor de l'ordre *UseCanonicalName*.

Segons aquestes variables, el format CLF seria:

"%h %l %u %t \"%r\" %>s %b"

El format CLF incloent el servidor web virtual sol·licitat (un servidor web pot servir diversos dominis DNS diferents o servidors virtuals):

"%v %h %l %u %t \"%r\" %>s %b"

– El format NCSA estès/combinat seria:

"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""

Analitzar la demanda i rendiment d'un servidor és una tasca necessària, ja que els servidors web estan subjectes a variacions de demanda molt extrema. Després de l'anàlisi dels fitxers de diari del servidor pot ser necessari limitar, resituar, ampliar els recursos del servidor i la xarxa d'accés a Internet per a poder atendre acceptablement l'"estrany i voluble" trànsit de peticions que visita els servidors web.

2. Organització de les aplicacions en servidors web

Les aplicacions web –aplicacions que van associades o són extensions d'un servidor web–, poden necessitar un disseny i ajust molt acurat per a oferir un rendiment adequat en situacions d'alta demanda o simplement per a respondre ràpidament o aprofitar adequadament els recursos de la màquina on estan instal·lats.

En primer lloc, s'ha de saber com està organitzat un servidor web per a atendre peticions HTTP de la manera més eficient.

En segon lloc, s'ha de conèixer com es pot estendre el servidor web per a oferir altres serveis gestionats per un codi addicional.

2.1. Organització del servidor web

Per a caracteritzar com s'organitza un servidor web per a atendre peticions d'una manera eficient i econòmica és necessari definir alguns termes:

– Procés: la unitat més "pesada" de la planificació de tasques que ofereix el sistema operatiu. No comparteix espais d'adreces ni recursos relacionats amb fitxers, excepte de manera explícita (heretant referències a fitxers o segments de memòria compartida), i el canvi de tasca el força el nucli del sistema operatiu (*preemptiu*).

– Flux o *thread*: la unitat més "lleugera" de planificació de tasques que ofereix el sistema operatiu. Com a mínim hi ha un flux per procés. Si diversos fluxos coexisteixen en un procés, tots comparteixen la mateixa memòria i recursos d'arxius. El canvi de tasca en els fluxos el força el nucli del sistema operatiu (*preemptiu*).

– Fibra: fluxos gestionats per l'usuari de manera cooperativa (***no*** *preemptiu*), amb canvis de context en operacions d'entrada/sortida o altres punts explícits: en cridar certes funcions. L'acostumen a implementar llibreries fora del nucli i l'ofereixen diversos sistemes operatius.

Arquitectura del servidor web

L'apartat "4.4 Server Architecture" del llibre: Krishnamurthy, Web Protocols and Practice (pàg. 99-116), descriu un estudi sobre el funcionament d'Apache 1.3.

Per a veure quins models de procés interessen en cada situació, cal considerar les combinacions del nombre de processos, flux per procés i fibres per flux. En tot cas, cada petició la serveix un flux que resulta la unitat d'execució en el servidor.

Els models que es poden construir són (U: únic, M: múltiple):

Procés	Flux	Fibra	Descripció
U	U	U	És el cas dels processos gestionats per *inetd*. Cada petició genera un procés fill que serveix la petició
M	U	U	El model del servidor Apache 1.3 per a Unix: diversos processos preparats que es van encarregant de les peticions que arriben. L'implementa el mòdul Apache: *mpm_prefork*
M	U	M	En cada procés una llibreria de fibres canvia de context tenint en compte l'acabament d'operacions d'entrada/sortida. En Unix se l'anomena *select-event threading* i l'usen els servidors Zeus i Squid. Ofereix un rendiment millor a Unix que a MUU. L'implementa el mòdul Apache *state-threaded multi processing*.
M	M	U	El model MMU canvia fibres per fluxos. L'implementen els mòduls Apache: *perchild* i *mpm_worker_module*. El nombre de peticions simultànies que pot atendre són *ThreadsPerChild* x *MaxClients*.
U	M	U	El model més senzill amb diversos fluxos. Es pot muntar en Win32, OS2, i amb fluxos POSIX. L'implementa el mòdul Apache: *mpm_netware*
U	M	M	Probablement el que proporciona un rendiment més alt. En el Win32 es pot aconseguir amb els anomenats *completion ports*. S'usen els fluxos necessaris per a aprofitar el paral·lelisme del maquinari (nombre de processadors, targetes de xarxa) i cada flux executa les fibres que han completat la seva operació d'entrada/sortida. És el model amb un rendiment més alt en el Windows NT. L'implementa el mòdul Apache: *mpm_winnt_module* Molts servidors com Internet Information Server o IIS 5.0 utilitzen aquest model amb Windows NT. El servidor web intern al nucli de Linux *Tux* també utilitza aquest model.
M	M	M	Pot ser una generalització d'UMM o MUM i en general la presència de diversos processos fa que el servidor es pugui protegir de fallades com a conseqüència que un procés hagi de morir per fallades internes, com l'accés a memòria fora de l'espai del procés, etc.

En general, els models amb molts processos són costosos de memòria (cada procés n'ocupa la seva part) i de càrrega (creació de processos). En servidors d'alt rendiment, els models amb fluxos semblen millors, encara que tenen la dificultat de la portabilitat difícil i la possible necessitat de mecanismes d'exclusió mútua. Hi ha models amb *pools* de processos o fluxos que anticipen el cost de creació de processos o fluxos i, per tant, són molt ràpids atenent peticions.

TUX: servidor web al nucli de Linux

Tux és un servidor web incorporat al nucli de Linux que usa un *pool* amb molt pocs fluxos del nucli (un flux per processador), llegeix directament de la memòria del sistema de fitxers de dins el nucli, usa el seu propi algorisme de planificació de fibres, i usa el TCP/IP "zero-copy" que minimitza les vegades que les dades que vénen de la xarxa es copien a la memòria (en xarxes d'alta velocitat com Gigabit Ethernet, les còpies de blocs de memòria són el factor limitant). Pot servir entre dues i quatre vegades més peticions/segon que Apache o IIS.

Per a més informació: www.redhat.com/docs manuals/tux/

Quan la xarxa limita el servei web, llançar processos per a atendre peticions pot funcionar raonablement bé, però en xarxes d'alta velocitat, com ATM o Gigabit Ethernet, el cost d'iniciar un procés en rebre una petició és excessiu.

En màquines uniprocessador, els models amb un sol flux funcionen bé. En màquines multiprocessador, és necessari usar múltiples fluxos o processos per a aprofitar el paral·lelisme del maquinari.

El major obstacle per al rendiment és el sistema de fitxers del servidor, ja que la funció principal del servidor web és "traslladar" fitxers del disc a la xarxa. Si aquest s'ajusta, el següent obstacle és la gestió de la concurrència.

A més, els estudis de trànsit web indiquen que la majoria de peticions són de fitxers petits, per la qual cosa seria convenient d'optimitzar el servidor per a

poder prioritzar aquestes peticions que són més freqüents i milloren el temps de resposta percebut pels usuaris, per exemple, en carregar pàgines web amb molts gràfics petits inserits.

Per aquesta raó, Apache 2.0 és un servidor web modular en el qual la gestió del procés està concentrada en un mòdul que es pot seleccionar de la instal·lació, segons les característiques del sistema operatiu, la màquina, els recursos que haurà d'utilitzar el servidor, etc.

2.2. Organització de les aplicacions web

Els servidors web s'encarreguen d'atendre i servir peticions HTTP de recursos, que en la seva forma més simple acostumen a ser documents guardats en el sistema de fitxers. Tanmateix, l'altra funció important d'un servidor web és la d'actuar de mitjancer entre un client i un programa que processa dades: rep una petició amb algun argument, la processa i torna un resultat que el servidor web lliura al client. La interacció entre el servidor web i els processos que té associats és un altre aspecte a considerar.

Hi ha diverses maneres d'organitzar una aplicació web. A continuació es presenten per ordre cronològic de complexitat i rendiment creixent diversos models d'organització:

– CGI: és el model més antic i simple. Per a cada petició HTTP s'invoca un programa que rep dades per les variables de l'entorn i/o d'entrada estàndard, i torna un resultat per la sortida estàndard. Consumir un procés per cada petició genera problemes importants de rendiment, que el model FastCGI intenta millorar.

– *Servlets*: és un model dissenyat per al Java, més eficient i estructurat, que permet de triar diversos models de gestió de fluxos o *threads*, durada de processos del servidor, etc. Partint d'aquest model, s'han construït servidors d'aplicacions amb múltiples funcions addicionals que faciliten el desenvolupament d'aplicacions web complexes.

2.3. Interfície comuna de passarel·la (*common gateway interface*, CGI)

La interfície comuna de passarel·la és un estàndard per a proporcionar una interfície web a programes que s'executen a cada petició i que, per tant, poden tornar informació *dinàmica*. Com que un programa CGI és executable –és com deixar que tothom executi un programa en el servidor–, s'han de prendre moltes precaucions en fer accessibles programes CGI per via del servidor web.

La CGI és un programa extern que s'executa i respon a cada petició del servidor web dirigida a la CGI. El mode de funcionament és el següent:

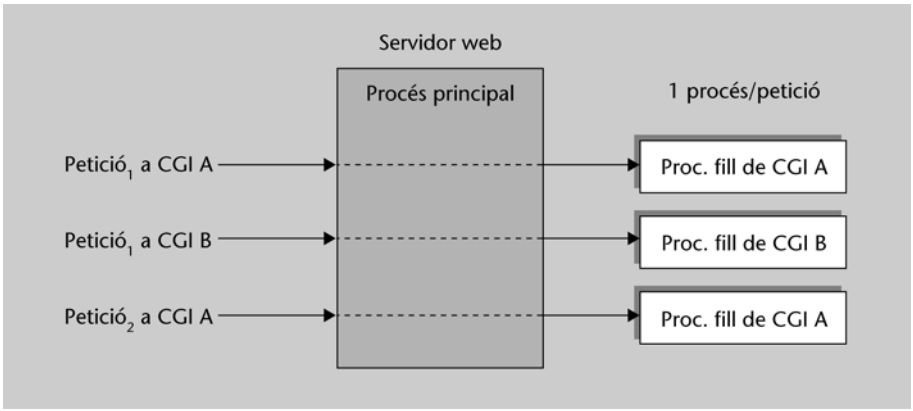

Figura 8. Un servidor web que utilitza processos (comandaments) CGI

Cada petició crea un procés que rep les dades d'entrada per mitjà de l'entrada estàndard i de l'entorn, i genera una resposta per la sortida estàndard.

Per exemple, si es vol "connectar" una base de dades al web, perquè tothom la pugui consultar, s'hauria d'escriure una CGI. En demanar al servidor web un URL dirigit a la CGI, el programa consultarà la base de dades i retornarà un text, possiblement HTML, que presenti el resultat de la pregunta.

Es pot connectar qualsevol programa que segueixi les regles senzilles de les CGI per a ser invocat pel servidor web, sempre que el programa no trigui molt a respondre, ja que l'usuari o el navegador es poden cansar d'esperar.

Aquest procediment consumeix molts recursos del servidor i és lent. Per a ampliar, la informació visiteu l'adreça següent: http://www.w3.org/CGI/.

2.3.1. FastCGI

Per a solucionar el problema anterior, es va crear una millora de les CGI que fa que un sol procés carregat vagi servint peticions sense descarregar-se (un procés persistent o daemon), però de vegades no n'hi ha prou amb un procés i fa falta tenir diversos processos alhora atenent diverses peticions que es donen simultàniament.

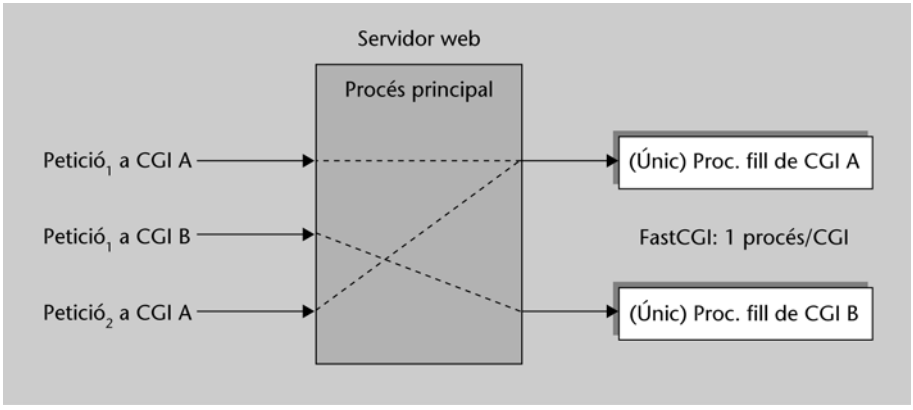

Figura 9. Un servidor que utilitza processos persistents FastCGI

Tant els CGI com els FastCGI no poden interactuar amb l'interior del servidor (per exemple, generar una línia als fitxers de diari del servidor). Vegeu http://www.fastcgi.com.

Una altra alternativa per a estendre el servidor d'una manera més eficient consisteix a utilitzar les API d'extensió de cada servidor (NSAPI al servidor de Netscape/Sun, ISAPI al servidor de Netscape, o un mòdul en Apache). Les extensions formen part del procés servidor i aquests API ofereixen molta més funcionalitat i control sobre el servidor web, a més de velocitat, en estar compilades i formar part del mateix executable.

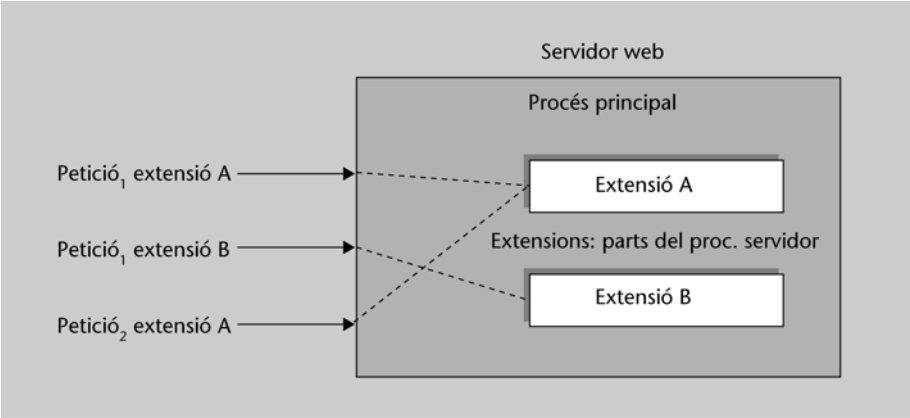

Figura 10. Un servidor que incorpora codi extern usant l'API d'extensió

Tanmateix, tenen tres problemes importants: són extensions no portables, específiques per a un únic servidor; són més complexes de desenvolupar i mantenir; introdueixen risc en el procés servidor –perill pel que fa a la seguretat i fiabilitat (un error de l'extensió pot aturar el procés servidor de web).

2.4. Servlet Java

Un *servlet* és una extensió genèrica del servidor: una classe Java que es pot carregar dinàmicament al servidor per "estendre" la funcionalitat del servidor web. En molts casos, substitueix els CGI amb avantatges.

Servlets: una aportació de Sun

L'especificació de Servlets continua evolucionant.

Els *servlets* són una extensió estàndard de Java, i les classes acostumen a venir amb les distribucions de Java SDK (Equip de Desenvolupament Java) o en l'adreça següent: http://java.sun.com/products/servlet

És una extensió que s'executa en una màquina virtual java (JVM) dins el servidor: és segur i transportable. Com que s'executa des del servidor, el client web l'invocarà com si fos un CGI, i en la resposta només veurà el text HTML, sense que el client hagi de tractar amb Java (com sí que fa en les miniaplicacions, que és codi Java que s'executa en una màquina virtual Java del client web).

En general, un *servlet* és un tros de codi que s'executa en un servidor: es pot usar per "estendre" servidors de web però també serveixen per a altres servidors (ex: ftp per a afegir comandaments nous, correu per filtrar o detectar virus, etc.).

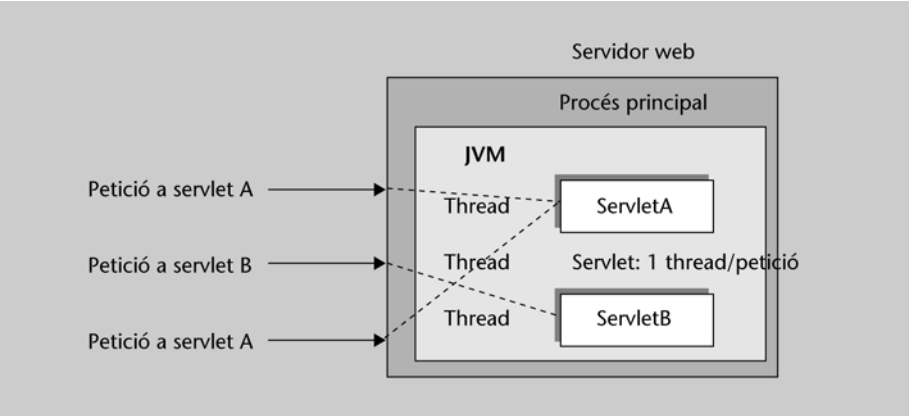

Figura 11: un servidor estès amb *servlets* en el seu JVM

Per a usar els *servlets* fa falta un "motor" on provar-los i posar-los en servei. Poden ser servidors que ja suportin *servlets* (per exemple, Domino Go de Lotus, Website d'O'Reilly, Jigsaw del Consorci Web), o mòduls que es poden afegir a servidors que inicialment no els suportaven (per exemple, Jserv per a Apache: http://tomcat.apache.org).

Tomcat

El servidor tomcat, és un servidor web escrit amb Java que directament suporta *servlets* i és d'ús lliure: http://jakarta.apache.org/tomcat

Els avantatges principals dels *servlets* són els següents:

– **Portabilitat**: sempre usen les mateixes crides (API) i circulen sobre Java, això fa que siguin veritablement portàtils entre entorns (que suportin *servlets*).

– **Versatilitat**: poden usar tot l'API de Java (excepte AWT), a més de comunicar-se amb altres components amb RMI, CORBA, usar Java Beans, connectar amb bases de dades, obrir altres URL, etc.

– **Eficiència**: un cop carregat resta a la memòria del servidor com una única instància. Diverses peticions simultànies generen diversos fluxos sobre el *servlet*, que és molt eficient. A més, en estar carregat a la memòria pot mantenir el seu estat i mantenir connexions amb recursos externs, com bases de dades que puguin reclamar un cert temps per a connectar.

– **Seguretat**: a més de la seguretat que introdueix el llenguatge Java (gestió de memòria automàtica, absència de punters, tractament d'excepcions), el gestor de seguretat o *security manager* pot evitar que *servlets* malintencionats o mal escrits puguin danyar el servidor web.

– **Integració amb el servidor**: poden cooperar amb el servidor d'unes maneres que els CGI no poden, com canviar el camí de l'URL, posar línies de diari al servidor, comprovar l'autorització, associar tipus MIMI als objectes o fins i tot afegir usuaris i permisos al servidor.

2.4.1. L'API de *servlets*

Els *servlets* usen classes i interfícies de dos paquets: `javax.servlet` que conté classes per a *servlets* genèrics (independents del protocol que usin) i

`javax.servlet.http` (que afegeix funcionalitat particular de l'HTTP). El nom `javax` indica que els *servlets* són una extensió.

Els *servlets* no tenen el mètode `main()` com els programes Java, sinó que s'invoquen uns mètodes quan es reben peticions. Cada vegada que el servidor envia una petició a un *servlet*, s'invoca el mètode `service()` que s'haurà de reescriure (*override*). Aquest mètode accepta dos paràmetres: un objecte petició (*request*) i un objecte resposta.

Els *servlets* HTTP, que són els que usarem, ja tenen definit un mètode `service()` que no fa falta redefinir i que crida el `doXxx()` amb `Xxx`, el nom de l'ordre que ve amb la petició del servidor web: `doGet()`, `doPost()`, `doHead()`, etc.

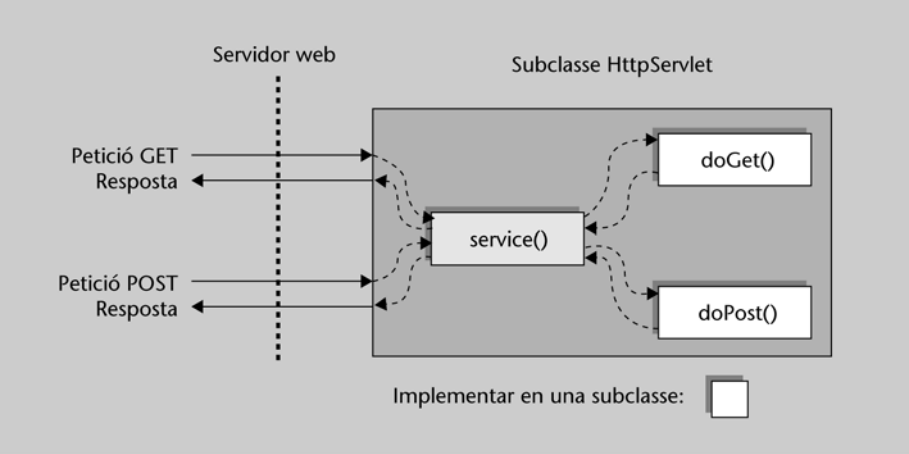

Figura 12. Un *servlet* HTTP que tracta peticions GET i POST.

A continuació es pot observar el codi d'un *servlet* HTTP mínim que genera una pàgina HTML que diu "Hola amics!" cada vegada que s'invoca en el client web.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class hola extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
                   throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<html><big>Hola Amics!</big></html>");
  }
}
```

El servidor estén la classe `HttpServlet` i reescriu (*overload*) el mètode `doGet()`. Cada vegada que el servidor web rep una petició GET per a aquest *servlet*, el ser-

vidor invoca el seu mètode `doGet()` fent-li arribar un objecte amb dades de la petició `HttpServletRequest` i amb un altre objecte `HttpServletResponse` per a retornar dades en la resposta.

El mètode `setContentType()` de l'objecte resposta (`res`) estableix com a text/HTML el contingut MIME de la resposta. El mètode `getWriter()` obté un canal d'escriptura que converteix els caràcters Unicode que usa Java en el joc de caràcters de l'operació HTTP (normalment ISO-8859-1). Aquell canal s'usa per a escriure el text HTML que veurà el client web.

2.5. Resum i comparació

Per tant, és possible estendre la funcionalitat d'un servidor web incorporant *servlets* que atenen peticions web. Usen els mètodes d'HTTP GET, amb els arguments codificats en l'URL (*URL encoded*), i POST, on els arguments viatgen al cos de la petició. En general, els *servlets* serveixen per a estendre la funcionalitat de qualsevol servidor afegint nous serveis o "comandaments".

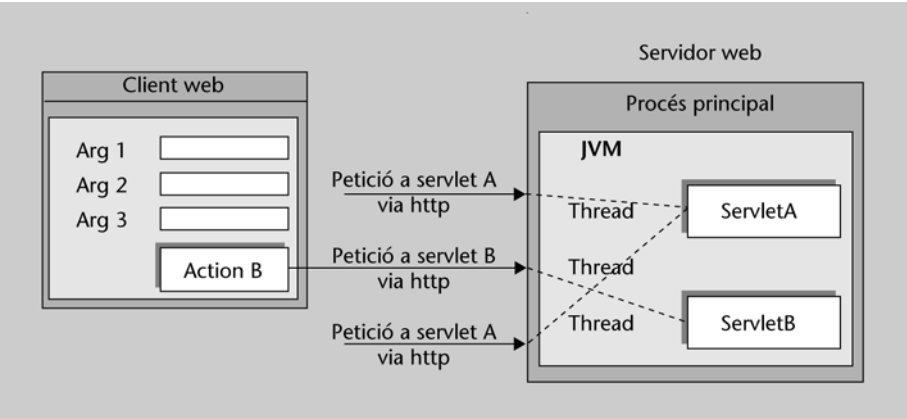

Figura 13. Interacció del navegador amb el formulari i del servidor web amb els *servlets*.

Un avantatge important dels *servlets* respecte als CGI és que es pot seleccionar la política de servei (fluxos i instàncies): una vegada instanciat un *servlet* a la màquina virtual Java (JVM) pot servir diverses peticions usant un flux per a cada una o, al contrari, un objecte (amb un sol flux) per a cada petició. Un *servlet* també pot guardar informació que persisteix durant la vida de l'objecte o connexions a bases de dades. A més, la llibreria de *servlets* facilita i abstreu el pas de paràmetres i la seva codificació, el seguiment de successives visites d'un mateix client (sessions), la transformació de jocs de caràcters entre client i servidor, etc.

Servidors d'aplicacions

Són servidors que incorporen molts serveis i components reutilitzables que permeten de desenvolupar aplicacions complexes accessibles per HTTP.

Alguns exemples de servidors d'aplicacions molt populars (tot dos implementen el model *Java 2 Enterprise Edition* o J2EE):

www.jboss.org de codi lliure, www.bea.com: WebLogic Server, comercial.

El model client web + formulari HTML / servidor web + extensions (CGI o *servlet*) és adequat per a aplicacions en les quals l'usuari utilitza un client web que invoca una única operació al servidor amb un conjunt de paràmetres textuals i sense estructura que recull el client web mitjançant un formulari HTML.

3. Servidors *proxy-cache* web

Per a la web, es va dissenyar un protocol simple (HTTP) d'accés a documents sobre un transport fiable com el TCP. Un objectiu del disseny era la interactivitat: el client es connecta amb el servidor web, sol·licita el document (petició) i immediatament la rep del servidor. Aquest esquema és ràpid en situacions en què el trànsit a la xarxa i la càrrega dels servidors sigui reduïda, però no és eficient en situacions de congestió.

Per a totes aquelles situacions en què la comunicació directa client-servidor no és convenient, s'han introduït uns tipus de servidors que fan de mediadors entre els extrems.

Un servidor intermediari (*proxy*) o mediador és un servidor que accepta peticions (HTTP) de clients o altres intermediaris, i genera al seu torn peticions cap a altres mediadors o cap al servidor web de destinació. Actua com a servidor del client i com a client del servidor.

La idea del servidor intermediari s'usa en molts protocols a més de l'HTTP. En general, se situen en una discontinuïtat per a fer-hi una funció. Per exemple:

Canvi de xarxa: una màquina connectada a la xarxa interna d'una organització, que usa adreces IPv4 privades (per exemple, 10.*.*.*), i també a Internet, pot fer de servidor intermediari per a la traducció d'adreces IP entre les dues xarxes (NAT).

Si no es vol fer servir aquest mecanisme, que té certes limitacions, es pot superar la discontinuïtat al nivell de l'HTTP posant un servidor intermediari HTTP: una màquina connectada a totes dues xarxes que rep peticions de pàgines web des de qualsevol client intern per una adreça interna i torna a generar la mateixa petició des del servidor intermediari, però cap a Internet, amb l'adreça externa.

NAT (*network address translation*)

Als paquets IP sortints: substituir l'adreça IP de les màquines internes (no vàlides a Internet) per la seva pròpia, als paquets IP entrants: substituir la seva pròpia adreça IP per la d'una màquina interna i reenviar el paquet cap a la xarxa interna.

Per poder saber a qui s'ha de lliurar, el servidor intermediari ha d'associar cadascun dels seus ports a les màquines internes, ja que una adreça de transport és una parella (adreça IP, port).

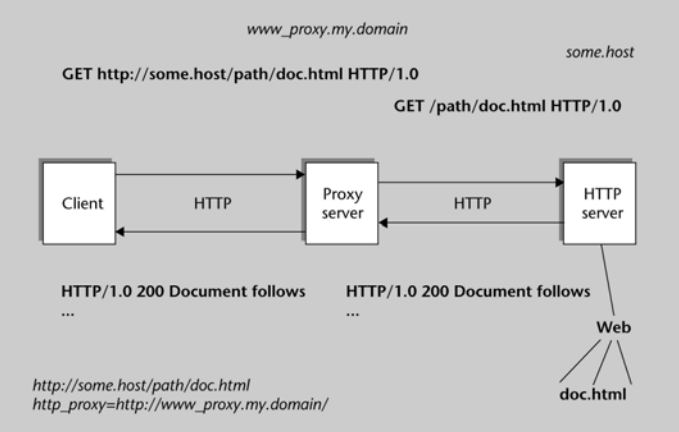

Figura 14. Interacció entre client-servidor intermediari-servidor tal com apareix en la publicació original (Luotonen, 94) descrivint el servidor intermediari HTTP del CERN (Centre Europeu de Recerca en Física) on va ser inventada la web.

Com podeu suposar, aprofitant que tant la petició com el resultat (la pàgina web) han de passar pel servidor intermediari, aquest es pot aprofitar per a fer algunes funcions:

Control d'accés a continguts: el servidor intermediari consulta si la pàgina sol·licitada és o no permesa per la política de l'organització. Es pot fer consultant una taula de permisos o usant el mecanisme anomenat PICS.

Control de seguretat: el servidor intermediari genera totes les peticions que surtin cap a Internet, això oculta informació i evita atacs directes sobre les màquines internes de l'organització. A més, pot existir un tallafoc que no permeti d'accedir directament a l'exterior, de manera que l'ús del servidor intermediari és imprescindible.

Aprofitar peticions reiterades (funció cau)*:* el servidor intermediari pot emmagatzemar (en memòria o disc) una còpia dels objectes que arriben com a resultat de peticions que han passat pel servidor intermediari. Aquests objectes es poden usar per a estalviar peticions a Internet i servir peticions successives d'un mateix objecte amb una còpia emmagatzemada en el servidor intermediari, sense haver de sortir a buscar-la a Internet. Els ***proxy-cache*** són el cas més freqüent de servidor intermediari, i són els que aquí detallarem.

Adaptar el contingut: un servidor intermediari també podria adaptar els objectes que vénen del servidor a les característiques del seu client. Per exemple, convertir els gràfics al format que el client entén, reduir la seva mida per a clients com telèfons mòbils amb poca capacitat de procés, comunicació o presentació.

El servidor intermediari pot ser transparent, invisible per al client i el servidor: s'obté el mateix resultat amb ell o sense ell, encara que en els navegadors web habitualment l'usuari ha de configurar expressament el seu navegador per a usar-lo, ja que el navegador no té cap mecanisme per a detectar-lo i usar-lo automàticament.

En algunes instal·lacions, es poden instal·lar extensions del programari de l'encaminador que fan que l'encaminador intercepti les connexions TCP sortints cap al port 80 (HTTP) i les redirigeixi a un *proxy-cache*. Té el perill que l'usuari no n'és conscient i pot portar a situacions equívoques si el *proxy-cache* falla o gestiona malament la petició.

En la figura següent podem veure com hi ha servidors intermediaris per a diversos protocols a més de l'HTTP, i que un servidor intermediari es pot comunicar amb el servidor origen (el que té el document original que l'usuari ha sol·licitat) o demanar-lo a un altre servidor intermediari, formant una jerarquia de servidors intermediaris.

PICS: Platform for Internet Content Selection

El PICS defineix mecanismes perquè es puguin catalogar llocs i pàgines web segons certs criteris, i així controlar l'accés a continguts no desitjats. És útil a les escoles i les llars per a controlar l'accés dels menors a Internet.

Control de continguts: certs continguts que es consideren no apropiats per l'organització poden ser bloquejats o redirigits.

Podeu trobar més informació en l'adreça següent: http://www.w3.org/PICS/.

El protocol HTTP 1/1...

... té definit un codi de resposta a una petició:

305 Use proxy

El problema és que no es va arribar a cap acord en escriure l'especificació i la resposta fa que en el navegador simplement hi aparegui el mateix missatge d'error, en lloc de tractar de buscar un servidor intermediari i reintentar la connexió per mitjà d'aquest. Com ho podem arreglar?

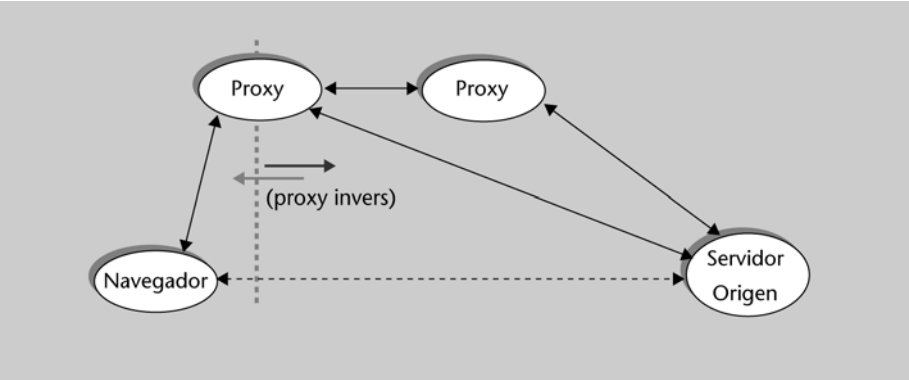

Figura 15. Un servidor intermediari se sol situar amb proximitat a un client, pot col·laborar amb altres servidors intermediaris, però també pot estar a prop d'un servidor (servidor intermediari invers).

També s'usen servidors intermediaris propers a un servidor per tal de reduir la càrrega de peticions sobre aquest. Són els servidors intermediaris inversos: pot ser més senzill i barat col·locar un o diversos *proxy-cache* que rebin totes les peticions, ja que respondran directament les peticions ja mediades pel servidor *proxy-cache* (*cached* en anglès), i al servidor només n'arribaran algunes que o bé encara no són al *proxy-cache*, o bé han expirat, o són continguts dinàmics.

a)

b)

Figura 16. a) Petició HTTP 1.0 client → servidor intermediari, b) petició HTTP 1.0 servidor intermediari → servidor. Podem observar com el servidor intermediari introdueix un camp "reenviat" per a notificar la seva presència al servidor.

A causa de la gran difusió de l'ús de servidors *proxy-cache*, sobretot en entorns acadèmics, l'especificació HTTP/1.1 defineix una nova capçalera que permet de controlar el comportament d'un servidor *proxy-cache* tant des del client en la petició, com des del servidor en una resposta.

Cache-control (petició):

No-cache	client↔origen (les memòries cau s'inhibeixen)
No-store	El servidor intermediari no ha d'emmagatzemar permanentment petició/resposta
Max-age = sgs	La màxima "edat" acceptable dels objectes en memòries ràpides de treball.

Max-stale	S'accepten objectes vells.
Max-stale = sgs	S'accepten objectes **sgs** segons vells.
Min-fresh = sgs	A l'objecte li han de quedar **sgs** de vida.
Only-If-Cached	Petició si només està al servidor *proxy-cache*.

Cache-control (resposta)

Public	Es pot mediar per servidors intermediaris i client.
Private	Només es pot desar a la memòria cau del client.
Private="cabc"	Tots poden mediar l'objecte, excepte la capçalera **cabc**: només a la memòria cau del client
No-cache	No es pot mediar ni en servidors intermediaris ni en el client.
No-cache="cabc"	Combinació dels dos anteriors.
No-store	Ningú no pot emmagatzemar permanentment (només a la memòria del navegador).
No-transform	Els servidors intermediaris no poden transformar el contingut.
Must-revalidate	Revalidar (amb origen) si és necessari.
Max-age	Marge d'edat en segons.

Els servidors intermediaris *proxy-cache* tenen algorismes per a decidir si quan arriba una petició d'un contingut que ja tenen, és o no necessari de confirmar que no hagi canviat en el servidor d'origen, el camp cache-control permet influir en la decisió. Un altre problema greu és la pèrdua de privacitat potencial si es guarden continguts al servidor intermediari, encara més si s'emmagatzemen objectes en disc. Per a això també serveix el camp cache-control.

Un *proxy-cache* està format per un magatzem de missatges de resposta (objectes), un algorisme de control del magatzem (entrar, sortir, esborrar o reemplaçar), i un algorisme ràpid de consulta (*lookup*) d'un mapa del contingut; pren la decisió de servir-lo del magatzem o demanar-ho al servidor d'origen o a un altre *proxy-cache*.

El seu ús pot produir una reducció de trànsit a la xarxa i en el temps d'espera en peticions repetitives si els objectes que es demanen estan en la memòria i el contingut es pot mediar. Estudis en diverses organitzacions mostren amb freqüència taxes d'encert en la memòria del 15%-60% d'objectes, encara que això pugui variar molt en funció dels usuaris i el contingut.

Els servidors *proxy-cache* són sistemes passius que aprofiten per guardar les respostes dels usuaris a les seves peticions. Descarten, automàticament, guardar continguts que semblen privats, dinàmics o de pagament: els detecten per la presència de camps a la capçalera com ara: *WWW-Authenticate, Cache-Control:private, Pragma:no-cache, Cache-control:no-cache, Set-Cookie*.

Cookies (galetes): estat i privacitat

El protocol HTTP no té estat: cada interacció (demanda + resposta) no té relació amb les altres i cada una pot usar una connexió TCP diferent.

Per a poder relacionar diverses interaccions HTTP i passar informació d'estat entre aquestes, Netscape va inventar les galetes (*cookies*): un servidor web pot enviar al client un objecte fins a 4096 bytes, que conté dades que el navegador tornarà al mateix servidor en el futur (o als altres servidors que indiqui la galeta).

Les galetes s'han fet servir molt per a qüestions publicitàries (quins llocs web visita la gent i amb quin ordre), per a guardar contrasenyes, identificar usuaris, etc. sense que l'usuari sigui conscient que està rebent aquestes "galetes", ni que les està enviant quan visita la web.

Hi ha qui diu que són un error portat a la perfecció. Què en penseu? Heu mirat alguna vegada quines galetes teniu al vostre navegador?

S'han proposat millores per fer dels servidors *proxy-cache* intermediaris actius: acumular documents d'interès en hores de trànsit baix per a tenir el contingut preparat i ajudar així a reduir el trànsit en hores d'alta demanda, o mecanismes de *pre-fetch* en què la memòria cau s'avança a portar, abans que li ho demanin, pàgines web que l'usuari sol·licitarà, amb molta probabilitat, immediatament. Tanmateix, no són d'ús comú perquè no impliquen sempre un estalvi o una millora del temps de resposta, també poden fer desaprofitar els recursos de comunicació portant objectes que després no es demanen.

Els mecanismes de *proxy-cache* són útils perquè una comunitat d'usuaris que comparteixen una connexió a Internet puguin estalviar amplada de banda, reduint transferències repetitives d'objectes. Tanmateix, aquesta només és una part del problema.

El problema en conjunt es coneix humorísticament com el ***World-Wide Wait***. Diversos agents participen en el rendiment d'una transferència web:

– Proveïdor de continguts (servidor web): ha de planificar la seva capacitat per a poder donar servei en hores punta i aguantar possibles allaus de peticions, i així tenir una certa garantia que els seus clients tindran un bon servei amb certa independència de la demanda.

– Client: podria usar una memòria cau per a "economitzar" recursos de la xarxa. Quan un contingut es troba en més d'un lloc, hauria de triar el millor servidor en cada moment: l'original o una rèplica "ràpida" (o portar l'objecte a trossos des de diversos llocs alhora).

– Rèplica: si un servidor desa una rèplica d'algun contingut probablement és perquè el vol oferir i reduir la càrrega del servidor original. Per tant, ha de ser "conegut" pels seus clients, però, a més, el contingut ha de ser consistent amb el contingut original.

Una rèplica (*mirror*)?

El gener de 2007 el programa HTTPD Apache, el servidor web més utilitzat a Internet, és capaç de baixar unes **300 rèpliques** del lloc web original.

La llista de rèpliques és a l'adreça següent: http://www.apache.org/mirrors/ i la informació del servidor a l'adreça http://httpd.apache.org.

– Proveïdor de xarxa: hauria de triar el millor camí per a una petició, via encaminament IP, via resolució DNS (resoldre noms a l'adreça IP més pròxima al client que consulti: no és el més habitual), via servidors *proxy-cache* HTTP i evitar zones congestionades (*hot spots*) o *flash crowd* (congestió al servidor i a la xarxa pròxima deguda a una allau de demandes).

Per tant, les memòries cau només són part de la solució. Les xarxes de distribució de continguts són la solució d'una altra part del "W-W-Wait".

4. Continguts distribuïts

Una altra millora que ajuda a combatre el mal del "W-W-Wait" són els sistemes de distribució de documents: n'hi ha prou amb un únic servidor per a qualsevol "audiència"? La resposta és que no, si és que es desitja oferir una qualitat adequada per a les demandes, tant les més petites com les més grans, per a continguts que poden arribar a ser molt populars en certs moments, que es poden visitar des de qualsevol lloc del món, que poden tenir una audiència potencial enorme.

Les aplicacions que ofereixen continguts a Internet s'enfronten al repte de l'escala: un únic servidor davant milions de persones que eventualment poden demanar els seus serveis tots alhora: el proveïdor d'informació ha de posar tants recursos com audiència pugui tenir.

En conseqüència, ha de pagar més qui té alguna cosa interessant per a explicar o vol arribar a més públic. La web no funciona com una antena de ràdio: la potència d'emissió determina la cobertura, però el nombre de receptors no l'afecta; és més similar al telèfon, ja que la capacitat es mesura en nombre de línies i això determina el nombre de persones que es poden atendre alhora.

Per això, pot fer falta disposar de diversos servidors per a poder repartir la càrrega.

La primera web pública, ara ja fora de funcionament, va ser http://info.cern.ch. Però la primera web amb una demanda important va ser www.ncsa.uiuc.edu. Aquesta web va haver d'usar quatre servidors replicats per a satisfer la demanda. En la gràfica següent podeu veure l'evolució del nombre de peticions entre juliol de 1993 (91.000 peticions/setmana) i abril de 1994 (1.500.000 peticions/setmana).

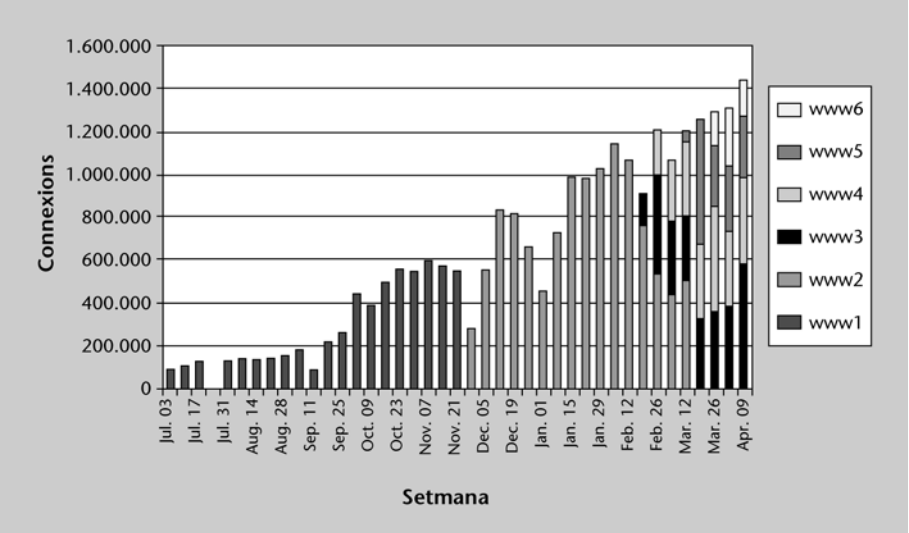

Figura 19. Creixement del nombre de peticions setmanals durant dos anys. Cada to de gris es correspon amb una màquina diferent. A mitjan febrer de 1994, diverses màquines van començar a servir peticions alhora fins a arribar a quatre.

Hi ha diversos "trucs" per a repartir peticions entre diverses màquines:

– *Mirrors* amb un programa que readreça la petició HTTP a la millor rèplica (vegeu l'exemple següent d'Apache)

– Fer que el servei de noms DNS torni diverses adreces IP (el mètode *round robin*). D'aquesta manera, els clients poden fer peticions HTTP a una adreça IP diferent cada vegada.

www.google.com

Si es pregunta al DNS per www.google.com, la resposta conté diverses adreces IP: nslookup www.google.com.

– Readreçament a nivell de transport (commutador de nivell 4, *L4 switch*): un encaminador mira els paquets IP de connexions TCP cap a un servidor web (port 80) i les redirigeix a la màquina interna menys carregada

– Readreçament a nivell d'aplicació (commutador de nivell 7, *L7 switch*): un encaminador que mira les connexions HTTP i pot decidir amb quina rèplica contactar en funció de l'URL sol·licitat. Molt complex.

– Enviar totes les peticions a un servidor intermediari invers que respongui o amb contingut guardat en la memòria o passi la petició a un o diversos servidors interns.

Si, a més, les rèpliques se situen a prop dels clients, el rendiment serà millor i més previsible. L'inconvenient és que muntar un servei web distribuït és difícil i car.

La Fundació Apache distribueix el servidor httpd Apache amb la col·laboració de més de 200 rèpliques distribuïdes arreu del món. Tot i que les pàgines web es poden consultar al servidor d'origen, a l'hora de baixar el programa hi ha un programa que calcula i redirigeix el visitant al servidor més pròxim. D'aquesta manera, les rèpliques ajuden a repartir la càrrega alhora que ofereixen un servei millor al client. Aquí el contingut s'actualitza periòdicament.

Com ser *mirror* d'Apache?

Apache demana que el contingut s'actualitzi com a mínim dues vegades a la setmana.

Les eines per a fer-ho són:

1) rsync: un programa que compara dos llocs remots i transfereix els blocs o fitxers que hagin canviat. Vegeu http://rsync.samba.org.

2) cvs: un programa pensat per al desenvolupament del codi a distància, permet de detectar i intercanviar diferències. Vegeu http://www.cvshome.org.

3) Apache com a *proxy-cache*: configurar un servidor apache per a passar i fer *cache* de les peticions: Ordre: ProxyPass / http://www.apache.org/ CacheDefaultExpire 24.

4.1. Xarxes de distribució de continguts

Han sorgit empreses que s'han dedicat a instal·lar màquines a molts llocs del món i algorismes per a decidir quina màquina és la més adequada per a atendre les peticions, segons la ubicació del client i la càrrega de la xarxa. Aquestes empreses venen aquest "servidor web distribuït" a diversos clients que paguen per a poder disposar d'un sistema de servei web de gran capacitat i que pot respondre a demandes de continguts extraordinàries.

Aquests sistemes es denominen **xarxes de distribució de continguts** (*content delivery networks* o CDN) i es poden trobar diverses empreses que ofereixen aquest servei: Akamai és la més important.

Una CDN és un programari intermediari (***middleware***) ***entre proveïdors de continguts i clients web***. La CDN serveix continguts des de diversos servidors repartits per tot el planeta. La infraestructura de la CDN està compartida per diversos clients de la CDN i les peticions tenen en compte la situació de la xarxa per a fer que cada client web obtingui el contingut sol·licitat des del servidor més eficient de la CDN (usualment una combinació del més proper, el menys carregat, el que té un camí amb més amplitud de banda amb el client).

És com un servei de "logística": la CDN s'encarrega de mantenir còpies a prop dels clients als seus propis magatzems (no en un punt central sinó als extrems de la xarxa). Per això, ha de disposar d'un gran nombre de punts de servei en multitud de proveïdors d'Internet (servidors *surrogate*: funcionalitat entre servidor *proxy-cache* i rèplica). Per exemple, Akamai ofereix el gener del 2007 uns 20.000 punts de servei a tot el món.

Una proposta difícil de rebutjar

L'oferta d'una CDN com Akamai a un proveïdor d'accés a Internet (ISP) podria ser: "ens deixes a la teva sala de màquines l'espai equivalent al d'una nevera de casa, nosaltres t'enviem unes màquines i tu les endolles a la xarxa elèctrica i a la teva xarxa (Internet). Nosaltres les administrem".

Què passarà? Totes les peticions web a diversos milers d'empreses seran servides per les nostres màquines.

L'ISP estalvia despesa d'amplada de banda amb Internet, ja que els seus usuaris no necessitaran anar a Internet per a buscar alguns continguts que seran servits per les màquines d'Akamai en l'ISP.

Akamai cobra del proveïdor de continguts per servir el contingut més ràpid i millor que ningú.

Algunes CDN, a més, se serveixen d'enllaços via satèl·lit d'alta capacitat (i retard) per a traslladar continguts d'un lloc a un altre evitant la possible congestió d'Internet. Encara que potser no siguin ideals per a objectes petits, poden funcionar bé per a objectes grans (per exemple, programari, àudio, vídeo). Alguns exemples són les empreses Amazon 33, SkyCache, Real Networks.

Mentre que la transferència d'una pàgina web normal funciona com segueix:

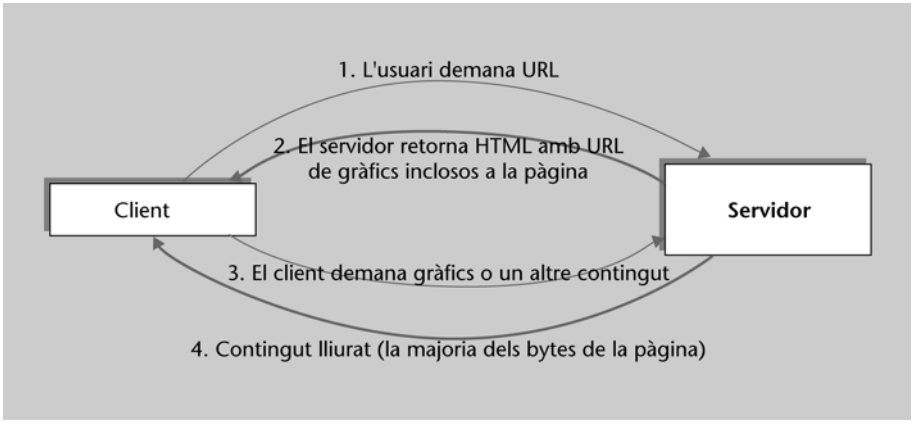

Figura 20. Fases d'una petició web

Una petició d'una pàgina web (document HTML + alguns gràfics) fa diverses operacions: 0.1) resolució DNS del nom del servidor, 1.1) connexió TCP amb el servidor, 1.2) sol·licitud de l'URL del document, 2) el servidor torna el document HTML que conté referències a gràfics inclosos a la pàgina, 3) resolució DNS del nom del servidor on hi ha els gràfics, que acostuma a ser el mateix servidor que la pàgina HTML, 3.1) sol·licitud dels gràfics necessaris (es pot repetir diverses vegades), 4) contingut servit i pàgina visualitzada pel client (navegador).

Una petició a una CDN com Akamai aprofita per a prendre decisions a cada pas de la petició:

Akamai

És interessant visitar la web d'Akamai:
http://www.akamai.com

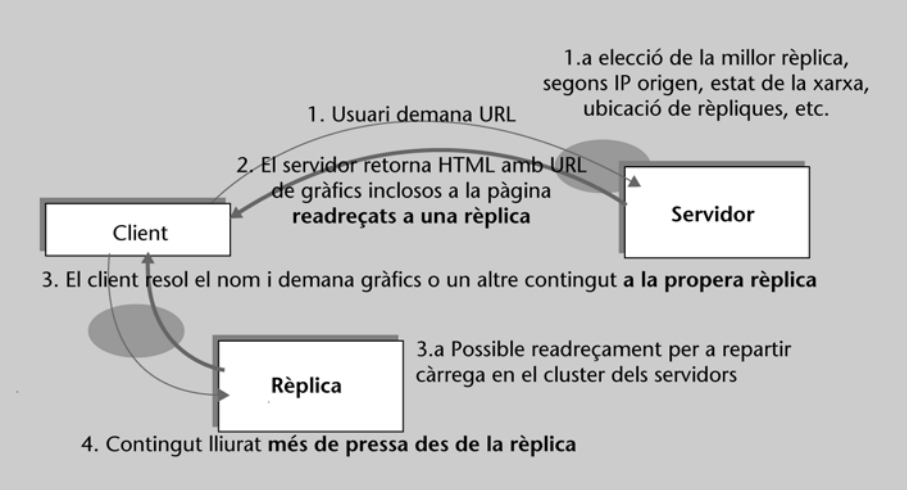

Figura 21. Fases d'una petició web usant una CDN

La CDN pot actuar sobre el següent:

– el DNS: quan es resol un nom, el servidor DNS respon, segons la ubicació de l'adreça IP del client;

– el servidor web: quan es demana una pàgina web, es reescriuen els URL interns segons la ubicació de l'adreça IP del client;

– quan es demana un objecte a l'adreça IP d'un servidor de la CDN, el commutador (*switch*) d'accés a un conjunt de diversos servidors *proxy-cache*, o rèpliques, pot redirigir la connexió al servidor menys carregat del grup (tot el conjunt respon sota una mateixa adreça IP).

Els passos de la petició d'una pàgina web amb gràfics podrien ser els següents:

0. L'usuari escriu un URL (per exemple, http://www.adobe.com) en el seu navegador i el navegador ***resol*** el nom en DNS (192.150.14.120).

1. L'usuari demana l'URL a l'adreça IP (192.150.14.120).

1.a El servidor web decideix la millor rèplica, segons l'IP d'origen, estat de la xarxa, ubicació de les rèpliques, o com a mínim classifica el client en una zona geogràfica determinada (per exemple, amb Akamai decideix que som a la zona del món o regió "g").

2. El servidor torna l'HTML amb l'URL dels gràfics inclosos a la pàgina (marques <img>) s'han **redirigit a una rèplica.** D'aquesta manera, els gràfics, que constitueixen el major nombre de bytes d'una pàgina web, seran transferits per la CDN. Per exemple,

<img src="http://a1944.g.akamai.net/f/1944/1482/8h/fondo.jpg">.

3. El client ***resol*** el nom i demana gràfics o un altre contingut ***a la rèplica més propera***.

3.a Possible redirecció amb el DNS o per readreçament de la connexió TCP (a nivell de TCP o HTTP: un commutador de nivell 4 o 7), ***repartiment de la càrrega en un grup de servidors***.

4. Contingut lliurat ***més de pressa des d'una rèplica***.

L'avantatge de tot aquest muntatge és que l'usuari continua veient en el seu navegador un URL normal (el de la pàgina HTML), el servidor de l'empresa té diaris amb visites a HTML i el departament de màrqueting pot fer estadístiques, però la major part del trànsit (els gràfics) la serveix Akamai des de la proximitat del client.

Akamai manté informació sobre l'estat de la xarxa, dels grups de sectors (*clusters*), dels seus "servidors". No sempre tria el "millor possible", però sí un dels millors, mentre que evita els pitjors servidors a cada moment.

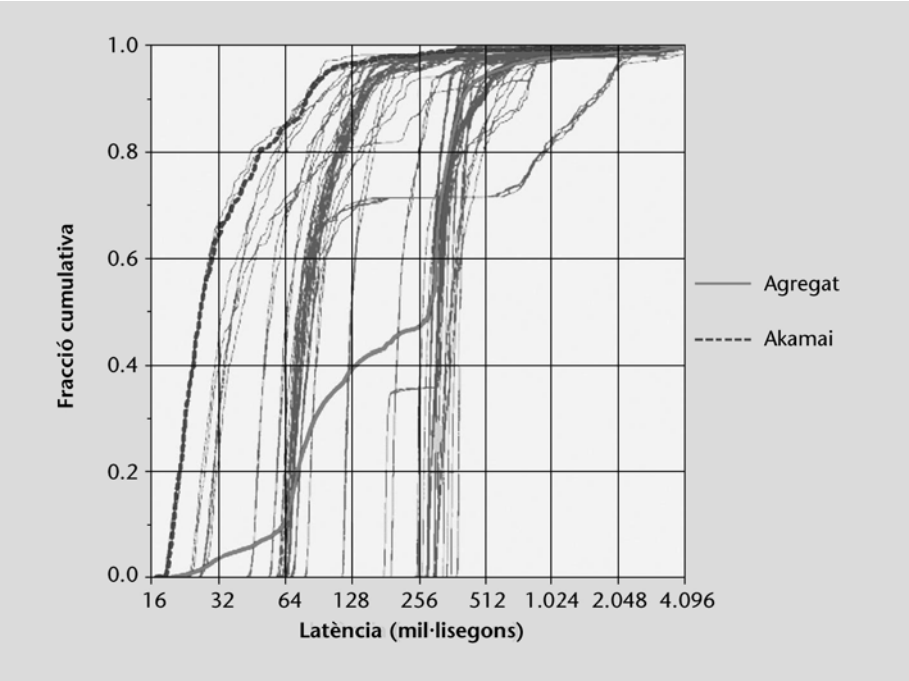

Figura 22. Distribució acumulada de temps de resposta (latència) de diversos servidors Akamai.

La gràfica anterior mostra en línies fines el temps que es triga a obtenir, des d'un lloc determinat, un objecte de 4 kb reclamat a molts servidors d'Akamai.

L'eix *x*, en escala logarítmica, mesura el temps (ms), l'eix *y* mesura la distribució acumulada de casos en què el temps de descàrrega és inferior a un cert valor. La línia gruixuda (*aggregate*) mesura la mitjana de tots els servidors, i la línia gruixuda intermitent (*akamai*) mostra el temps del servidor que selecciona Akamai a cada moment (gairebé sempre selecciona el millor).

La línia vermella indica que si es fes una elecció aleatòria, per al 20% (0,2) dels casos, el temps de servei seria inferior a 64 ms, però per al 80% dels casos el temps seria inferior a 400 ms. Amb l'elecció d'Akamai, el 80% de les peticions se serveixen en menys de 50 ms. Si triéssim un dels pitjors servidors, només podríem dir que el 80% de les peticions se serveixen en menys de 1.000 ms.

Conseqüència: la decisió d'Akamai és gairebé ideal, molt millor que la decisió aleatòria, i sens dubte que el pitjor cas (Llei de Murphy).

5. Computació orientada a serveis

Les aplicacions web retornen resultats que poden correspondre al contingut d'un arxiu (continguts estàtics) o ser el resultat de l'execució d'un programa (continguts dinàmics). Les aplicacions que generen continguts dinàmics poden requerir petites o grans dosis de computació i emmagatzematge. Alguns exemples són els cercadors web, les botigues a Internet, els llocs de subhastes, els serveis de mapes o imatges de satèl·lit, que poden necessitar realitzar complexos càlculs, cerques, o servir enormes volums d'informació.

Dins les organitzacions també s'utilitzen sistemes complexos que processen quantitats ingents de dades i que presenten els seus resultats a través d'un navegador. En són exemples les aplicacions financeres, científiques, d'anàlisi de mercats, que tracten amb enormes quantitats de dades que cal recollir, simular, predir, resumir, estimar, etc. perquè unes persones puguin avaluar una situació i prendre decisions.

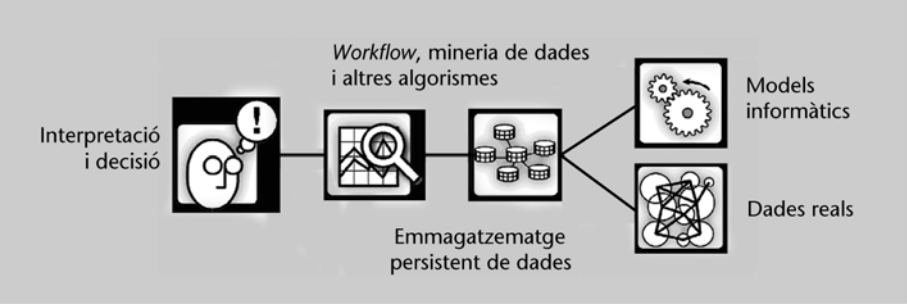

Figura 23. Exemple d'una aplicació de computació intensiva, en què a partir d'unes fonts de dades reals o simulades, s'emmagatzemen i processen per a la visualització de dades, anàlisi i presa de decisions.

En aquests sistemes cada unitat autònoma de processament es pot agrupar en un servei i la forma d'interacció entre ells se sol basar en serveis *grid* o serveis web. Quant a la interacció amb l'usuari, tot aquest conjunt de serveis es poden amagar rere un servidor web i una interfície d'usuari que pot ser tradicional (basada en formularis HTML senzills) o bé ser més interactiva i més semblant a una aplicació centralitzada utilitzant les capacitats avançades dels navegadors com l'AJAX.

Sobre l'AJAX podeu veure el mòdul 3, "Mecanismes d'invocació".

En aquest model d'aplicacions o sistemes, el processament està repartit entre el que ocorre al navegador de l'usuari (relacionat amb la presentació i interacció amb l'usuari), el servidor web (mediació entre una aplicació al servidor i el navegador remot) i altres servidors i serveis localitzats potencialment en altres màquines, ubicacions i fins i tot altres organitzacions seguint un model *grid*. Aquests sistemes poden tenir una estructura fixa o bé dinàmica, en els quals les eines i recursos s'incorporen segons la demanda (un model d'ús d'un *grid* anomenat *utility computing*).

5.1. Computació sota demanda

Utility computing (el "servei públic" de computació) és un model tècnic i de negoci en què els recursos informàtics es proporcionen sota demanda i pagant segons l'ús.

Difereix del model de computació convencional en el fet que en aquest no han d'invertir a disposar de capacitat per al cas de màxima demanda, sinó que paguen per l'ús real dels recursos. La suma de diversos clients amb diferents demandes que usen certs recursos fa que s'optimitzi la seva utilització. Aquest model de *grid* és comparable al de l'ús de l'electricitat, gas, correus i molts altres serveis públics, per la qual cosa es podria anomenar *servei públic de computació*. Atenent la seva dependència de l'ús, també s'anomena *computació sota demanda*.

La cerca a Internet

La cerca a Internet és una activitat intensiva en computació. El següent article mostra que fa falta un sistema massivament replicat per a dedicar a cada petició de cerca la màxima capacitat de computació necessària per a trobar les referències a qualsevol cosa en qualsevol lloc del Web ordenat per rellevància.

Luiz Barroso; Jeffrey Dean; Urs Hoelzle (2003, març). "Web Search for a Planet: The Google Cluster Architecture". *IEEE Micro* (vol. 23, núm. 2, pàg. 22-28).
<http://labs.google.com/papers/googlecluster.html>

Comparant aquest model amb la xarxa elèctrica, veiem que aposta per no limitar la computació a la capacitat d'una màquina aïllada (amb capacitat limitada de càlcul del processador, emmagatzematge del seu disc dur, aplicacions instal·lades i llicenciades, elèctrica de la seva bateria), i prendre computació, emmagatzematge, aplicacions i electricitat dels corresponents serveis públics.

Aquestes idees no són noves, i el 1969 un dels inspiradors de la xarxa Internet va dir:

"As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of 'computer utilities' which, like present electric and telephone utilities, will service individual homes and offices across the country."

Leonard Kleinrock (2005, novembre). "A vision for the Internet". *ST Journal of Research* (vol. 2, núm. 1, pàg. 4-5).

En aquest model, les aplicacions web no queden limitades a la interacció navegador-servidor web, sinó que es converteixen en aplicacions completament distribuïdes en termes de processament i emmagatzematge, on múltiples components interactuen entre ells mitjançant l'intercanvi de dades, la invocació de serveis, i que requereixen considerar en el seu disseny tots els conceptes presentats en aquesta assignatura: des de l'estructura i organització dels components, la sincronització i els problemes d'ordre en la concurrència, la fiabilitat i tolerància a fallades, la replicació i el consens entre les rèpliques, el rendiment, els mecanismes d'invocació de serveis, etc.

Resum

En aquest mòdul s'han presentat les formes amb què un servei web o una aplicació associada a un servidor web es pot organitzar per a atendre la demanda que pot rebre d'Internet.

Molts indicadors d'Internet continuen creixent exponencialment: el nombre de persones amb accés a Internet i el trànsit web, que avui en dia predomina sobre la resta de protocols. La popularitat dels continguts segueix la llei de Zipf, la llei de la desigualtat extrema: moltes visites per a molt pocs llocs. Això fa que la demanda que pugui rebre en un moment concret un servidor web pugui ser exageradament gran. Conèixer les característiques principals d'aquesta demanda ajuda a preveure situacions en el disseny d'un servei web.

La capacitat per a servir peticions és difícil de preveure en un sistema tan complex en què influeixen, entre molts factors, la capacitat de la xarxa i la seva càrrega, les característiques de tota la màquina, el sistema operatiu amb nombrosos subsistemes, o el servidor web. És convenient conèixer la capacitat de servei del nostre servidor i observar el seu comportament amb nivells de càrrega elevats usant eines de generació de trànsit i mesura del comportament sota una càrrega sintètica, i també eines de visualització d'estadístiques de demanda real a partir de diaris (*logs*), que ens permetin de detectar sobrecàrregues, sintonitzar o actualitzar el conjunt adequadament.

Les aplicacions en servidors web tenen dos components principals: el servidor web i el codi addicional, que estén el servidor per a oferir serveis o continguts "dinàmics".

El servidor web és un sistema que s'encarrega de servir algunes de les moltes peticions simultàniament per la qual cosa s'ha d'optimitzar l'organització de tota aquesta activitat simultània a partir de processos, fluxos i fibres.

Les aplicacions web reben peticions del servidor i li tornen un resultat per a enviar al client. Les CGI són el model més senzill i antic d'interacció servidor-aplicació, en què el servidor invoca un comandament (executa un procés) per cada petició. *FastCGI* és una millora de les CGI perquè el codi extern al servidor no s'hagi d'executar amb cada petició. Els *servlets*, la proposta més recent, proposen per a Java un model molt complet i eficient de gestió de concurrència i durada dels processos al servidor.

Un altre component important són els servidors intermedis entre navegadors i servidors web que desen còpies dels objectes que veuen passar des d'un servidor cap a un navegador. Principalment, permeten d'"escurçar" peticions servint-les

a mig camí del servidor, en la memòria, amb objectes rebuts recentment, fet que redueix la congestió d'Internet i la càrrega dels servidors.

Mentre que els servidors de memòria ràpida de treball se solen situar a prop dels lectors, la demanda global de certs continguts o la tolerància a fallades pot recomanar oferir un servei web des de diverses ubicacions. Hi ha diverses maneres de muntar un servei distribuït: replicació o *mirroring*, DNS *round-robin*, readreçament a nivell de transport o HTTP, servidors intermediaris inversos.

Una altra alternativa és contractar la provisió de servei a una xarxa de distribució de continguts o CDN que és una infraestructura comercial compartida per diversos clients, amb infinitat de punts de presència pròxims a la majoria d'usuaris d'Internet, que s'encarreguen de repartir i dirigir les peticions web cap als servidors menys carregats i més propers a l'origen de la petició.

La computació sota demanda és un model més general que permet distribuir un sistema en diversos components distribuïts que es comuniquen amb invocacions a serveis i l'ús de recursos (computació, emmagatzematge, serveis d'aplicació) que s'assignen dinàmicament segons quina sigui la necessitat o el volum de demanda dels seus usuaris.

Activitats

1. Esbrineu si el proveïdor d'Internet que té la connexió a Internet disponible al vostre PC té disponible algun servidor *proxy-cache* i, si és així, configureu el vostre navegador per utilitzar-lo. L'ús del servidor estalviarà càrrega en alguns servidors que hagin servit abans el mateix contingut estàtic a un altre usuari del servidor *proxy-cache*. Estigueu atents per notar si es produeix algun efecte apreciable amb l'ús del servidor *proxy-cache*.

Proveu la memòria utilitzant el comandament telnet al port de la memòria. Es pot invocar el mètode GET d'un objecte remot:

```
GET http://www.apache.org http/1.1↵
Host: www.apache.org↵
↵
```

o el mètode *trace* d'HTTP:

```
TRACI http://www.apache.org http/1.1↵
Host: www.apache.org↵
↵
```

i veure el resultat de la petició fixant-se en els camps de la capçalera de la resposta que indiquen el camí seguit per la petició per mitjà d'un o diversos servidors *proxy-cache*. Repetiu l'operació per veure si el contingut arriba fins al servidor o ens respon el servidor *proxy-cache* en lloc del servidor web.

2. Netegeu la memòria cau del vostre navegador i configureu una memòria cau petita del navegador, visiteu llocs amb molts gràfics i deixeu la finestra oberta sobre la carpeta que conté la memòria, observeu què fa quan està plena, quins objectes reemplaça i inferiu una hipòtesi sobre quines dades té en compte per a gestionar la memòria local.

Exercicis d'autoavaluació

1. Considerant que una aplicació web eficient es pugui construir utilitzant *FastCGI* o *servlets*, indiqueu quines característiques podria tenir una aplicació, perquè fos preferible fer-la amb FastCGI, i una altra en la qual la millor opció fos *servlets*.

2. Quines diferències es poden trobar en accedir per HTTP a un contingut web personalitzat (diferent per a cada persona, per exemple, durant la compra de diversos llibres en una llibreria, on per a cada llibre es mostra una fitxa que inclou una foto i un capítol de mostra en PDF) en el cas de:
a) connexió directa, **b)** connexió per mitjà d'un *proxy-cache*, **c)** per mitjà d'una xarxa de distribució de continguts com Akamai.
Comenteu l'efecte sobre el retard (temps a rebre el primer byte), el temps de servei (rebre l'últim byte), la despesa de recursos de la xarxa i la càrrega del servidor d'origen del contingut.

3. HTTP 1.1 té reservat el codi de fallada 305 (*Use proxy*) perquè els servidors d'HTTP puguin no acceptar connexions directes (que no hagin passat abans per un servidor intermediari). Indiqueu les raons i una situació en què aquest codi d'error pugui ser d'utilitat. Descriviu els passos que seguiria un client que intentés inicialment accedir a un servidor que no accepta connexions directes.

4. Hi ha diverses maneres de repartir la càrrega entre els servidors d'HTTP: a) readreçament a un servidor HTTP que té una rèplica del contingut, b) fer que el servidor de DNS resolgui en cada moment a un servidor distint (readreçament del DNS), c) readreçament a un servidor intermediari de la pregunta anterior. Feu una taula que compari quines limitacions té cadascuna i proposeu una situació òptima per a cada cas.

Solucionari

1. Una aplicació amb *FastCGI*: una aplicació no escrita a Java, o que no necessiti interactuar excessivament amb el servidor, o que hagi de ser extremadament eficient o petita.
Una aplicació amb *servlets*: una aplicació escrita a Java, o que necessiti un model de procés sofisticat, o que hagi d'interactuar amb el servidor web més estretament o que hagi de ser transportable amb facilitat a altres servidors i/o a altres màquines.

2. Efecte sobre el retard (temps a rebre el primer byte): $a \leq b$ encara que similars, excepte en els casos en què l'objecte es trobi en algun servidor *proxy-cache* i es pugui servir immediatament sense verificar amb el servidor (objecte estàtic, que canvia poc o gens i obtingut recentment), a *d*; si el contingut el serveix la CDN, segurament serà molt ràpid.
Temps de servei (rebre l'últim byte): *a*, *b* com a mínim igual, *b* pot ser molt més ràpid si el contingut se serveix del servidor intermediari. *d* probablement molt més ràpid, ja que se serveix d'un lloc pròxim al client, tret que el client tingui limitacions grans de velocitat de connexió a Internet.
Despesa de recursos de xarxa: *b*, *c* estalviar recursos de xarxa, ja que poden portar el contingut des de més a prop que *a*.
Càrrega del servidor origen del contingut: *b* estalvia peticions repetitives de contingut estàtic des del grup de clients que usen els servidors *proxy-cache*. En *d* el servidor pot haver delegat gairebé completament el servei d'aquell objecte a la CDN.

3. Quan un servidor està sobrecarregat pot tornar aquest codi d'error per a indicar al client que en lloc de connectar-se directament utilitzi un servidor intermediari. Això podria ajudar a agrupar les peticions i, per tant, a reduir la càrrega del servidor. El client hauria de saber quin servidor intermediari hi ha a prop seu que li pugui fer servei i s'hauria de connectar amb ell i repetir la petició via servidor intermediari. L'inconvenient és que el client hagi de tenir un servidor intermediari accessible o que l'hagi de descobrir. Una alternativa útil seria que aquest codi d'error pogués "orientar" el client donant-li la informació d'un servidor intermediari que el pogués ajudar (un servidor intermediari invers, per exemple, que acceptaria connexions de qualsevol client que demanés pàgines d'aquell servidor).

4. La resposta es podria expressar en una taula com la següent:

	A (a rèplica)	B (DNS)	C (readreçament)
Limitacions	S'ha de tenir preparada la rèplica abans. Propagar i actualitzar el contingut perquè sigui consistent. El retard és més gran, ja que s'ha de readreçar (establir una nova connexió TCP).	S'ha de replicar un domini complet (no només un conjunt de pàgines). Un client pot mediar la resolució i carregar més un servidor que un altre. Si es redueix el TTL de les respostes, es genera més trànsit DNS. S'han de modificar tots els servidors de DNS d'aquest domini (excepte si s'usa *round robin*).	Implica establir una altra connexió TCP amb el retard que això significa. El client pot no saber tractar el codi d'error o no tenir coneixement de cap servidor intermediari, no es pot obtenir, doncs la pàgina sol·licitada. Fa falta, a més, estar segur que el contingut estigui sincronitzat.
Situació òptima	Contingut d'alta demanda que canvia poc (*mirror* d'Apache).	Domini complet replicat en diversos servidors.	Situació estranya. Aquest codi d'error no està ben especificat i, per tant, no està implementat correctament. Serviria per a reduir la càrrega d'un únic servidor sense haver d'usar rèpliques.

Glossari

CDN *f* Vegeu **content delivery/distribution network**.

CGI *f* Vegeu **interfície comuna de passarel·la**.

common gateway interface *f* Vegeu **interfície comuna de passarel·la**.

computació *grid* *f* Model de computació distribuïda o paral·lela en què un sistema pot repartir les seves funcions entre components que es comuniquen en xarxa i utilitzen recursos o serveis pertanyents a diverses organitzacions.

content delivery/distribution network Xarxa de servidors que serveix pàgines web segons la ubicació dels usuaris, l'origen de la pàgina web i l'estat de càrrega de la xarxa. Les pàgines que serveix una CDN estan ubicades (possiblement en forma de memòria cau) en diversos servidors repartits per diverses ubicacions. Quan un usuari demana una pàgina que està allotjada en una CDN, la CDN readreça la petició des del lloc web original a un servidor de la CDN que sigui gairebé òptim en aquell moment per a aquest client, tenint en compte

diversos factors: distància, càrrega de la xarxa, càrrega del servidor, presència del contingut sol·licitat, etc. És molt útil per a llocs web amb molt de trànsit i interès global. Com més pròxim geogràficament estigui el servidor de la CDN del client, més ràpid rebrà el contingut i menys influència tindrà la càrrega de la xarxa. La presència de la CDN pot ser transparent (invisible) per a l'usuari.
sigla: **CDN**

interfície comuna de passarel·la *f* Interfície que especifica com transferir informació entre un servidor web i un programa. El programa recull dades que li passa el servidor web i que vénen d'un navegador, i torna dades en forma de document. El programa es pot escriure en qualsevol llenguatge que es pugui executar a la màquina del servidor web. Un problema és que per a cada petició s'ha de carregar i executar el programa associat, fet que significa una gran despesa de recursos del servidor.
en common gateway interface
sigla: **CGI**

proxy *m* Vegeu **servidor intermediari**.

rèplica *f* Còpia d'una entitat (document, conjunt de documents, servidor) que es manté actualitzada o sincronitzada amb altres rèpliques d'un conjunt. Els canvis realitzats en una rèplica s'apliquen i propaguen a la resta de rèpliques de manera automàtica i transparent per a l'usuari.

servidor intermediari *m* Servidor situat entre un client i un servidor real. Intercepta les peticions del servidor real per veure si pot satisfer la resposta. Si no, reenvia la petició al servidor real. Els servidors intermediaris poden tenir dos propòsits: millorar el rendiment (reduir el camí fins a la informació: servidors *proxy-cache*) o filtrar peticions (prohibir l'accés a certs continguts web, o transformar el format).
en proxy

servlet *m* Miniaplicació Java (o *applet*) que s'executa en un servidor. El terme s'acostuma a aplicar a servidors web. Són una alternativa als programes CGI. Els *servlets* són persistents i una vegada invocats es poden quedar carregats a la memòria i servir diverses peticions, mentre que les CGI es carreguen, executen i desapareixen per atendre una sola petició.

transparent *adj* Invisible en informàtica. Una acció és transparent si s'esdevé sense efecte visible. La transparència se sol considerar una bona característica d'un sistema, perquè aïlla l'usuari de la complexitat del sistema.

Bibliografia

Krishnamurthy, B.; Rexford, J. (2001). *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Cambridge (EUA): Addison-Wesley. ISBN 0-201-71088-9.

Un llibre exhaustiu i detallat sobre el funcionament del web més enllà de la presentació als navegadors. Estudia els detalls de les diverses versions del protocol HTTP, el seu rendiment, la interacció amb els altres serveis involucrats: DNS, TCP, IP, l'evolució històrica dels components del web, *scripts*, cercadors, galetes (*cookies*), autenticació, tècniques per a recollir i analitzar trànsit web, *proxy-caches* web i interacció entre *proxy-caches*.

Luotonen, A. (1998). *Web Proxy Servers*. Londres: Addison-Wesley. ISBN 0-13-680612-0.

Ari va desenvolupar el servidor *proxy-cache* del CERN, i també el servidor intermediari de Netscape. Parla dels detalls de l'estructura interna (processos) dels servidors, cooperació entre servidors, mediació, filtratge, monitoratge, control d'accés, seguretat, aspectes que afecten el rendiment d'un servidor intermediari i altres aspectes.

Luotonen, A.; Altis, K. (1994). *World-Wide Web Proxies*. Actes de la Conferència WWW94.

És l'article històric sobre *proxy-cache* en web, basat en l'experiència amb la memòria web del CERN.

Rabinovich, M.; Spatscheck, O. (2002). *Web Caching and Replication*. Boston (EUA): Addison-Wesley. ISBN 0-201-61570-3.

Un llibre exhaustiu i detallat sobre els avenços recents en memòries ràpides de treball i replicació per al web. Els capítols 4: "Protocols d'aplicació per al web", 5: "Suport HTTP per a *proxy-cache* i replicació", 6: "Comportament del web", ofereixen una bona visió general del problema i els mecanismes que hi influeixen. La part II analitza amb detall els mecanismes de *proxy-cache* i la part III, els mecanismes de replicació en el web.

Hi ha molts llibres específics i detallats sobre cada tecnologia explicada: sobre les especificacions, sobre com utilitzar-la, com instal·lar-la i administrar-la, referències breus, etc. Entre d'altres, l'editorial O'Reilly té molts llibres força bons i molt actualitzats sobre diversos temes de l'assignatura. Són molt detallats, molt més del que es pretén a l'assignatura, però a la vida professional poden ser de gran utilitat per a aprofundir els aspectes pràctics d'un tema (http://www.ora.com).