Tema 1 – Introducció

Guia d'estudi

Objectius

- Entendre per què tot sistema que utilitze una xarxa d'intercomunicació és un sistema distribuït.
- Identificar què és un sistema distribuït, per què són rellevants i quines són les seues aplicacions principals.
- Conèixer alguns exemples de sistemes distribuïts.
- Estudiar l'evolució dels sistemes distribuïts escalables i identificar la computació en el núvol ("*cloud computing*") com l'etapa actual d'aquesta evolució.

CONTINGUT

1 Concepte de sistema distribuït

Un sistema distribuït és un conjunt d'agents autònoms, que interactuen per a aconseguir algun objectiu comú.

- Cada agent és un procés seqüencial, amb el seu propi estat independent, que avança al seu propi ritme. En interactuar amb la resta poden fer-ho mitjançant intercanvi de missatges o usant memòria compartida.
- L'objectiu comú de la cooperació pot usar-se per a avaluar el comportament global del "sistema".

En la pràctica, un sistema distribuït és un sistema en xarxa.

2 Rellevància

L'estudi de les propietats dels sistemes concurrents va nàixer de la necessitat d'entendre com coordinar activitats paral·leles que es desenvolupaven sobre un conjunt de recursos (principalment memòria) compartits. Això va ser necessari amb la finalitat d'assentar sobre bases sòlides les estratègies d'implantació de sistemes operatius que permetien compartir els recursos d'un ordinador entre activitats concurrents.

Els sistemes distribuïts i en xarxa formen un cas d'especial dels sistemes concurrents. La característica principal que diferencia a un sistema distribuït dins dels sistemes concurrents és el fet que cadascun dels seus elements s'executa de forma autònoma i, a més, en el cas més comú, cada element té el seu propi conjunt de recursos (memòria inclosa), havent de cooperar utilitzant alguna "xarxa de comunicacions" que permeta l'intercanvi d'informació.

En els seus inicis, estudis sobre les propietats dels sistemes distribuïts havien de començar amb algun tipus de justificació sobre per què era interessant fixar-se en aquests sistemes que semblaven aportar més problemes que solucions.

La justificació que es donava era quàdruple i segueix sent vàlida: d'una banda es deia que augmentava els escenaris d'ús (funcionalitat) dels ordinadors, resultat de la possible cooperació entre sistemes a priori autònoms.

En segon lloc, s'incrementa el grau d'aprofitament dels recursos, de manera que els disponibles per a un ordinador (p. ex., impressores, discos...) puguen ser accedits des de qualsevol altre ordinador, augmentant les oportunitats de treball d'aquests recursos i la seua rendibilitat.

D'altra banda, es deia que era l'única forma d'augmentar el rendiment obtenible ja que hi havia límits físics a la potència que un sol ordinador podia aportar (velocitat, memòria, nombre de nuclis de processament…). La idea bàsica consistia en seleccionar una activitat (problema) complexa, després dividir-la en tasques (subproblemes) per a poder assignar cada tasca a un ordinador diferent.

Finalment, s'adduïa que era l'única manera d'augmentar la fiabilitat dels sistemes, al potencialment poder protegir-se de fallades localitzades en un sistema repartint la informació i còmput entre sistemes les maneres de fallada dels quals podien no estar correlacionades.

Aquesta situació va canviar relativament ràpid al llarg dels 80, quan es va començar a implantar el model client/servidor entre elements simples (per exemple, d'una estació de treball a una impressora), sustentat en l'existència de tecnologies de xarxa local variades.

En els 90, l'aplicabilitat dels resultats sobre sistemes distribuïts es va veure augmentada en realitzar-se la possibilitat de crear agrupacions d'un nombre limitat d'ordinadors per a augmentar la fiabilitat del conjunt, obtenint els anomenats clusters d'alta disponibilitat.

Amb el temps, la tecnologia de xarxa local que va guanyar ha sigut Ethernet (amb totes les seues variants), i amb aquesta tecnologia, el protocol de xarxa més adoptat sobre ella: l'IP (Internet Protocol).

L'aparició i desenvolupament de la Web en els anys 90 va popularitzar enormement l'adopció del protocol IP, i va permetre augmentar les possibilitats que els entorns basats en conjunts autònoms d'ordinadors comunicants permetien.

Avui dia podem afirmar que totes aquestes raons mantenen la seua vigència perquè l'entorn de computació actual ESTÀ distribuït i interconnectat, amb infinitat d'"ordinadors" connectats que interactuen per a oferir o consumir un gran nombre de serveis remots als quals accedir com a recursos compartits, com la Web.

D'entre els desafiaments que posseeixen importància en aquest àmbit en destaquem il·lustrativament dos:

1. Com aprofitar la connectivitat per a obtenir resultats útils. El desenvolupament de solucions amb aquests recursos suposa un replantejament en el qual les tècniques convencionals perden validesa. El canvi d'escala pot ser tal que la interpretació dels problemes i la concepció de les seues solucions no s'assemblen a cap de les referències que posseïm.
    - Imagina que a partir de la versió tradicional del sedàs d'Eratòstenes per a calcular nombres primers es desitja desenvolupar una altra en la qual un miler d'ordinadors col·laboren. Els nous problemes que apareixen afecten al redisseny de l'algorisme (com "dividir" el treball?, com "reunir" els resultats?), al balanç entre activitat i comunicació (un missatge per a cada treball o resultat?), a la modificació dinàmica del nombre d'intervinents (un ordinador falla), i si, després de totes aquestes complicacions, la millora produïda és significativa.
2. Com crear subsistemes capaços de proporcionar serveis robusts. Les noves tecnologies permeten abordar problemes i escales anteriorment impensables, en les quals apareixen noves qüestions:
    - Com se les apanya Google per a implantar el seu servei de cerca?
    - Com gestiona Dropbox l'ús compartit de fitxers per part de milions d'usuaris?

- Com distribuir entre milions de voluntaris la simulació de nous fàrmacs contra el càncer?

3 Àrees d'aplicació

Les àrees d'aplicació més destacables dels sistemes distribuïts són:

1. *World Wide Web*
2. Xarxes de sensors
3. *Internet of Things (IoT)*
4. Computació cooperativa
5. *Clusters* altament disponibles

Per això les tractem a continuació…

3.1 Aplicació a WWW

Basada en el model client/servidor, el servidor espera peticions de documents, mentre els clients són els navegadors web, que envien i reben documents.

- Les peticions a servidors impliquen la lectura o modificació d'un document.
- Els navegadors analitzen el document d'hipertext cercant metadades, entre les quals destaquen els enllaços (*links*) que apunten a altres documents, que poden trobar-se en aquest o un altre servidor.

Es tracta d'un paradigma simple i potent, inicialment dissenyat per a compartir documents, però estés per a permetre que les peticions sobre documents es convertisquen en peticions de servei, de manera que els "documents" retornats inclouen el resultat de la petició efectuada.

3.2 Aplicació a xarxes de sensors

Es tracta de mini-ordinadors de propòsit específic ("motes") que han sorgit gràcies a l'abaratiment dels equips. Es troben empotrats en dispositius d'ús quotidià (p.ex., en alguns electrodomèstics) i solen contenir sensors (humitat, temperatura, consum elèctric...) i actuadors (almenys un sistema d'avisos), la qual cosa els capacita per a un ample rang d'aplicacions potencials com la vigilància, detecció de desastres (químics, biològics...), monitoratge del consum elèctric i algunes altres.

3.3 Aplicació a la "*Internet of Things*"

Avui dia no és possible concebre un sistema informàtic sense pensar en la seua connexió a una xarxa de comunicacions. De fet, quasi s'està començant a assumir que qualsevol dispositiu electrònic que genera o maneja informació ha d'estar connectat a internet.

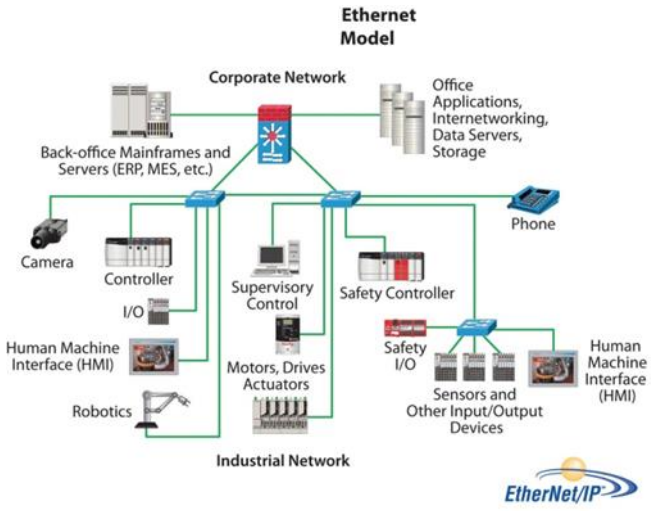
Tanta és la dependència de qualsevol element informàtic d'algun sistema de comunicació que s'ha encunyat un nou *mot* per a denotar l'extrem al que ens estem encaminant: *L'internet de les coses* (IoT), amb la visió que qualsevol dispositiu que manege informació quede connectat a Internet, a través de la qual puga oferir i demanar serveis i informació, i actuar sobre el seu propi entorn físic o el d'altres dispositius també connectats.

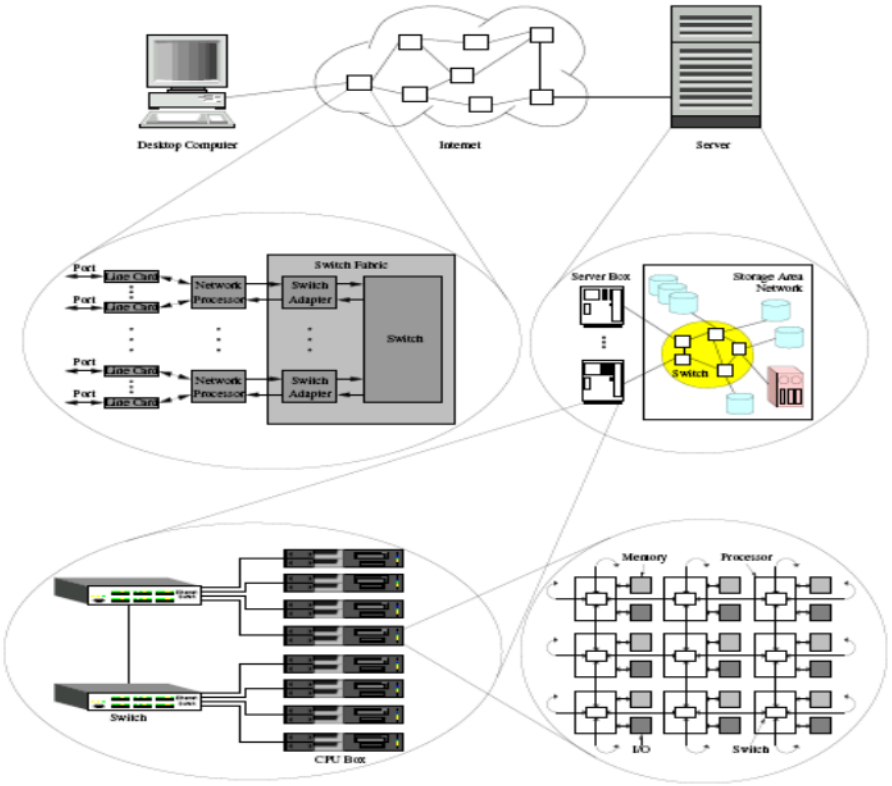
La IoT pot ser estudiada com una generalització de les xarxes de sensors en la qual tots els dispositius poden interactuar entre si i alterar el seu entorn físic. Aquesta capacitat obri nous escenaris:

- Ciutats intel·ligents
- Automatització de múltiples processos (construcció, fabricació...)
- Cura mèdica informatitzada

En aquest entorn els sistemes distribuïts intenten facilitar la connectivitat i interoperabilitat de tots els dispositius.

El concepte d'Internet de les coses vol capturar les àmplies possibilitats funcionals que s'obrin quan s'habilita que quasi qualsevol dispositiu tinga alguna manera de cooperar amb qualsevol altre dispositiu.





3.4 Aplicació a la computació cooperativa

La major part dels recursos computacionals s'infrautilitzen, destacant els ordinadors personals, que passen moltes hores diàries sense fer res.

D'altra banda, molts problemes científics i d'enginyeria poden dividir-se en peces menors (tasques)

- Cada tasca pot resoldre's en un interval breu, i els resultats de totes les tasques es poden combinar per a construir la solució completa del problema.
- El servidor crea el conjunt de tasques per al problema, i els clients poden obtenir una instància d'alguna d'aquestes tasques.

Els ordinadors amb accés a Internet poden subscriure's actuant com a "voluntaris" per a rebre tasques que resoldre:

- Instal·len un client especial: el "runtime" per a executar tasques.
- El client es registra al servidor, i intercala fases de processament en desconnexió amb altres d'intercanvi de dades i resultats amb el servidor.
- El servidor distribueix tasques entre els clients registrats i arreplega els seus resultats.

3.5 Aplicació a *clusters* altament disponibles

El desenvolupament de l'internet *de les coses* sorgeix de l'evident interès d'escenaris útils que es produeixen quan es consideren les comunicacions entre dispositius "*intel·ligents*" (que manegen informació) capaços, en alguns casos, d'interactuar amb l'entorn físic.

Els clusters d'ordinadors sorgeixen de la necessitat d'augmentar la fiabilitat dels sistemes, fent-los capaços de suportar fallades sense corrompre les dades o parar el processament normal.

Tot dispositiu està subjecte a fallades en el seu comportament. Els ordinadors no són una excepció. És més, un ordinador està predisposat a fallades de diversos tipus: uns provocats pel mateix maquinari, altres provocats per la impossibilitat d'assegurar el funcionament 100% correcte de tot el programari necessari per a engegar les aplicacions.

Des del principi, un dels objectius de les estratègies de maneig de fallades va ser la conservació de la consistència de dades. Diversos tipus de sistemes transaccionals es van encarregar d'assegurar que no es corromperen les dades que havien de persistir quan apareixien fallades que interrompien les activitats d'actualització. Almenys hauria de ser possible detectar la corrupció evitant la propagació d'informació incorrecta.

Amb el temps es va veure la possibilitat d'usar diversos ordinadors en configuracions diverses per a, a més de garantir la integritat de la informació emmagatzemada, permetre que el sistema resultant no parara encara que algun dels seus components ho fera. Això era possible per la redundància de mitjans ficats a disposició d'una computació. Si en lloc d'un ordinador es disposa de dos, en principi la fallada d'un d'ells pot ser compensada per l'altre, que començaria a respondre a les peticions abans manejades per l'ordinador caigut. Aquesta descripció aprofita la propietat que no tots els elements d'un sistema fallen alhora, excepte excepcions derivades de dependències mal plantejades (tots depenen de la mateixa instal·lació elèctrica, per exemple) o inevitables (un terratrèmol arrasa l'edifici on es troba l'empresa).

Aquesta estratègia ha acabat donant lloc a diverses tecnologies implementades en el que es coneix com **clusters d'alta disponibilitat**, dedicats usualment al maneig de grans aplicacions de gestió empresarial crítiques, on la disponibilitat i la integritat de les dades és essencial.

Els *clusters* d'alta disponibilitat intenten donar una resposta a la necessitat de garantir la integritat i disponibilitat de la informació. No obstant això les tècniques utilitzades en la seua implementació, donades les exigències de partida, fan que aquests sistemes no ***escalen*** adequadament.

L'escalabilitat és la propietat que té un sistema de ser configurat per a poder processar una major càrrega de peticions.

Els protocols de replicació i consistència utilitzats per un cluster d'alta disponibilitat típicament imposen una cota superior en el nombre d'ordinadors que ho formen, penalitzant-lo en rendiment si simplement s'afigen més ordinadors al cluster (els protocols d'alta disponibilitat consumeixen més recursos quants més ordinadors hi ha).

Per tant, l'única forma que té un cluster d'alta disponibilitat típic d'escalar consisteix a augmentar la capacitat de còmput/emmagatzematge/memòria de cadascun dels ordinadors que componen el cluster. Donats els límits físics existents en la construcció d'ordinadors, això implica que o bé siga impossible adaptar-se a certs nivells de càrrega, o siga excessivament car per haver d'utilitzar tecnologia nova que encara no ha sigut amortitzada en el mercat.

En qualsevol cas es requereix un pressupost extra per a poder disposar de sistemes d'aquest tipus, per la qual cosa únicament s'apliquen als entorns que poden invertir en aquest alt nivell de disponibilitat: bancari, empresarial, assistència mèdica, instal·lacions crítiques…

Des de la perspectiva de l'eficiència de la inversió, els centres de còmput empresarials se sobredimensionen per a donar cobertura al *pitjor cas*, i com a conseqüència s'infrautilitzen aquests recursos assignats a l'excés. És possible fer factor comú de molts dels serveis bàsics i recursos de diverses empreses per a un ús més eficient: programes i equips comuns, sous dels enginyers que administren aquestes aplicacions i equips, cost del subministrament elèctric, cost de la infraestructura de xarxa, etc.

Aquesta tendència a la racionalització conduirà a la computació en el núvol ("*cloud computing*")…

4 Computació en el núvol (CC)

L'evolució paral·lela de necessitats en la construcció de sistemes informàtics està confluint en el que avui dia es coneix com *Cloud Computing*.

Mentre que la IoT cerca maximitzar el nombre d'escenaris en què es pot extraure valor de la interacció de dispositius, i els clusters d'alta disponibilitat intenten utilitzar sistemes distribuïts per a incrementar la robustesa, cap d'aquestes aproximacions se centra a fer la creació i explotació de serveis més eficient. En aquestes aproximacions esmentades tampoc es presta especial atenció a la necessitat d'escalar "fàcilment".

En el que segueix donarem una visió de l'evolució del concepte de programari com a servei, i com això desemboca de forma natural en la visió actual del Cloud Computing. Aquesta serà una aproximació detallada en la qual abordarem:

1. Programes i serveis
2. Rols en el cicle de vida d'un SaaS
3. Evolució dels serveis de programari (*mainframes*, ordinadors personals, centres de còmput empresarials, SaaS, IaaS, SaaS sobre IaaS, i PaaS)

4.1 CC: Programes i serveis

Podem establir que l'objectiu general del CC és "*convertir la creació i explotació dels serveis de "programari" en un procediment més senzill i més eficient*". Al cap i a la fi els programes sempre s'han desenvolupat per a oferir algun tipus de servei amb l'ajuda dels ordinadors.

L'evolució de la indústria informàtica ha ocultat parcialment aquest fet, especialment perquè el fenomen dels ordinadors personals ha imposat una manera particular d'interacció dels usuaris amb els seus ordinadors.

4.2 Rols en el cicle de vida d'un SaaS

Considerem aquests 4 rols:

a) El desenvolupador, que implanta els components de les aplicacions.

b) El proveïdor de serveis, que decideix les característiques del servei, els components que ho constitueixen i com ha de ser configurat i administrat.
c) L'administrador del sistema, encarregat de que cada peça de *programari* i *maquinari* estiga en el seu lloc apropiat i adequadament configurat.
d) L'usuari, que accedeix al servei.

4.3 Evolució dels serveis de programari

Aquest apartat es basa en els anteriors rols per a il·lustrar l'operativa en cada model històric de servei de programari.

4.3.1 Mainframes

En els orígens de la computació, els *mainframes* eren gestionats per personal especialitzat, encarregat d'assegurar que tot estava correctament, i de carregar el sistema operatiu i, fins i tot, els programes d'aplicació.

Com l'ús d'aquests sistemes era marginal, no hi havia focus de contenció provocats per un excés de demanda.

Un gran percentatge d'usuaris d'aquests sistemes havien de desenvolupar codi per a poder extraure valor dels mateixos. Els papers de desenvolupador i d'usuari es confonien, fins i tot molts usuaris exercien de proveïdors de serveis tant amb els seus propis programes com amb els desenvolupats per uns altres.

- Al no diferenciar-se rols, trobem usuaris implicats en massa detalls de la gestió dels serveis que ells mateixos havien d'utilitzar.

D'altra banda els equips s'usaven eficientment, atès que estaven compartits per múltiples usuaris rebaixant el cost per ús de cadascun. L'adquisició era a càrrec de la institució propietària de l'equip.

4.3.2 Ordinadors personals

Amb el pas del temps, el descens en els costos, l'augment de la potència de càlcul disponible en el maquinari, i els avanços en les capacitats dels sistemes operatius, es va anar establint el que s'ha conegut com el model d'informàtica personal. El desenvolupament del programari passa a les mans d'empreses i organitzacions especialitzades, que destinen els seus desenvolupaments a un gran nombre d'usuaris. Aquests usuaris han d'instal·lar el programari en les seues estacions de treball amb la finalitat d'extraure valor al conjunt del sistema.

En aquest model, es promou la idea que l'usuari/posseïdor del PC té el poder de gestionar lliurement el seu sistema i instal·lar una gran varietat de programari sobre ell. El cost de la compra recau en l'usuari-propietari, però ja no competeix amb altres possibles usuaris. Aquesta “sobredotació” provoca que els recursos s'aprofiten deficientment (infrautilització de l'ordinador).

L'altra cara de la moneda en aquesta aproximació és que l'usuari-propietari ha d'encarregar-se ara d'administrar el seu sistema. No solament encarregar-se de resoldre les incidències de maquinari, sinó també assegurar-se que el programari que instal·la està en un estat adequat per a poder ser usat.

Estem davant un model d'"usuari com a administrador de sistemes", que encara preval en bona mesura avui dia, i la complexitat de la qual és una seriosa barrera per a la majoria de persones.

4.3.3 Centres de còmput empresarials

Els clusters d'alta disponibilitat no es lliuren d'aquesta servitud. Usualment instal·lats en els centres de dades d'empreses, aquestes tenen la necessitat de comptar amb personal especialitzat, i d'alt cost, que s'encarregue de la gestió i posada a punt del maquinari i programari que constitueix els clusters d'alta disponibilitat. L'empresa usuària és aleshores qui administra i proveeix els serveis basats en aquest cluster, com a l'entorn d'ordinadors personals.

En ocasions s'agrega el rol de desenvolupador de programes interns, depenent del volum i necessitats de l'empresa.

També es pot trobar una variant basada a mantenir aquests programes en centres de dades externs, principalment per motius econòmics:

- Evita el cost d'adquisició dels equips.
- Redueix i externalitza el cost d'administració i manteniment dels equips.
- Evita el cost fix de consum elèctric.
- La gestió dels costos informàtics resulta més senzilla.

4.3.4 Software as a Service (SaaS)

El programari sempre s'ha "fabricat" amb l'objectiu de proveir algun servei. L'accepció "moderna" de *Programari com a Servei* (SaaS, en anglès), realment denota un fet que sempre ha existit.

Per què, aleshores, sembla una novetat parlar de SaaS?

Pel canvi de percepció. En els últims 30 anys, amb la implantació del model d'informàtica personal, el programari s'ha vist com alguna cosa que calia manipular per a poder obtenir alguna funció d'ell.

No obstant això, d'un temps a ara, l'evolució tecnològica de les xarxes, amb la millora en ample de banda disponible per a bona part de la població, està fent cada vegada més factible reduir el paper d'usuari-administrador a un mínim. L'objectiu és que l'usuari no administre el seu "sistema PC" per a accedir als serveis del programari, sinó que aquests serveis li siguen prestats directament a través de la seua connexió a la xarxa, i usant una varietat cada vegada major de dispositius.

Un altre element relacionat és la millora de la interfície universal de web (el navegador) que permet executar localment interaccions complexes. El fenomen conegut com a "Web 2.0" destaca per presentar interfícies d'usuari més atractives, i per la reducció en la càrrega en els servidors, millorant l'escalabilitat.

L'últim ingredient destacable és l'excés de capacitat en els centres de dades existents[1], i que permeten comerciar amb la capacitat computacional excedent per a clients externs.

Apareix la figura explícita del proveïdor de serveis (o proveïdor de SaaS), qui ha d'encarregar-se de gestionar el desplegament del programari sobre tants servidors com necessite per a acollir a la seua població d'usuaris.

Els **problemes als quals ha d'enfrontar-se un proveïdor de serveis** són els següents:

- Garantir la qualitat del servei (QoS, *Quality of Service*), segons compromís amb els usuaris del mateix. L'acord especifica les característiques qualitatives i quantitatives que s'han de satisfer.
- Encarregar-se de seleccionar els components de programari més adequats per a poder muntar el servei i satisfer les garanties.
- Reaccionar enfront de les variacions de demanda del servei per part dels seus usuaris, de manera que per a satisfer les garanties utilitze els recursos justs.
- Reaccionar enfront d'incidents en el desplegament, de manera que el servei no es veja interromput.
- Mantenir el servei viu, mentre realitza actualitzacions del programari necessàries al llarg del cicle de vida del servei.

Notar que, fins ara, tots i cadascun dels punts anteriors han de ser duts a terme d'una manera o una altra per un usuari d'informàtica personal (qui és el seu propi proveïdor de serveis). Ara això ja no és responsabilitat de l'usuari.

Quins avantatges comporta aquest model?

Per a l'usuari són evidents: aquest se cenyeix al seu paper principal com a usuari del servei, evitant participar en tasques que en principi no són del seu interès, ni per a les quals té experiència. Això permet la participació potencial d'un major nombre d'usuaris d'un servei, en baixar la barrera d'adopció del mateix, així com racionalitzar les despeses associades a aquesta tasca, mitjançant l'adopció de procediments predictibles, i l'automatització de tasques.

Addicionalment, és possible un ajust raonable de preus del servei ja que el proveïdor pot aconseguir un gran nombre d'usuaris i, a més, es rebaixen potencialment les barreres d'entrada a més d'un proveïdor, augmentant la competència possible.

Per al proveïdor els avantatges es basen en l'economia d'escala que pot aconseguir de l'explotació d'una infraestructura comuna i uns procediments comuns per a donar servei a un gran nombre d'usuaris. El model d'ingressos també es fa més atractiu per al proveïdor, passant d'un model de venda de llicències de programari a un de subscripció amb ingressos periòdics compromesos (similar a una companyia elèctrica o telefònica).

No queda tan clara la separació dels altres rols, atès que sobre el proveïdor recau la responsabilitat del desenvolupament de les primeres aplicacions, i la realització de tasques d'administració d'equips i programes.

[1] Origen de la plataforma en el núvol d'Amazon

Cal fer notar que el proveïdor podrà aconseguir aquestes economies d'escala si és capaç de racionalitzar els seus propis costos, la qual cosa implica utilitzar/pagar a cada moment pels recursos justs (ni més ni menys) que necessita per a poder donar el seu servei, i usar els seus recursos per a donar servei a tots els seus usuaris, en lloc de dedicar uns recursos en exclusiva per a cadascun d'ells.

Idealment, a cada moment, si un proveïdor necessita 50 ordinadors, hauria de pagar per l'ús dels 50 ordinadors. Si més tard a causa de canvis en la demanda, només en necessita 25, hauria de pagar per aquests 25. Addicionalment, els 50 (o 25) ordinadors han de ser utilitzats per a donar diversos serveis a la seua població d'usuaris, repartint els costos entre ells.

En aquest entorn apareix el problema de l'adaptació àgil a la demanda dels usuaris, per a fer això es necessita que el servei estiga construït de manera que puga reconfigurar-se ràpidament per a aconseguir aquesta adaptació.

La capacitat que un servei té d'adaptar el seu consum de recursos segons la demanda amb la finalitat de mantenir una certa Qualitat de Servei s'anomena *elasticitat.*

Vegem alguns dels models de servei que s'han establit dins del que es coneix com Cloud Computing.

4.3.5 Infrastructure as a Service (IaaS)

Un proveïdor d'un SaaS concret està limitat en la seua capacitat de compartir els recursos que usa. Només té a la població d'usuaris del seu SaaS.

Mentre que el model de negoci del proveïdor pot aconsellar que aquest compre una sèrie de recursos dedicats per a proveir aquest SaaS, una aproximació que li permet gestionar millor els seus costos és l'accés a una infraestructura elàstica en si mateixa (és a dir, va reservant els ordinadors i emmagatzematge que necessita a cada moment segons la demanda dels seus propis usuaris).

Perquè un proveïdor de SaaS puga adaptar-se *elàsticament* a un cost raonable, han de complir-se almenys aquestes propietats:

1. Ha de disposar d'un proveïdor d'infraestructura capaç de cobrar-li per l'ús exacte que faça dels recursos computacionals.
2. El programari del SaaS ha de ser capaç d'adaptar-se al tipus de recursos que el proveïdor d'infraestructura posa a la seua disposició.
3. Els seus procediments han de permetre-li determinar la quantitat de recursos que el SaaS necessita a cada moment, i automatitzar la reconfiguració del desplegament del programari que conforma el servei.

D'un temps a ara, la iniciativa empresarial ha donat resposta a la necessitat de ser flexibles en la demanda de recursos computacionals i d'emmagatzematge, produint el que es coneix com *IaaS* (*Infrastructure as a* Service). Un IaaS és un servei elàstic el focus exclusiu del qual és la provisió de recursos computacionals, de comunicació i emmagatzematge a uns preus unitaris per ús.

El model SaaS de servei elàstic es filtra així cap a les capes inferiors (la infraestructura), convertint al proveïdor SaaS en usuari de l'IaaS.

El proveïdor de IaaS carrega així amb la responsabilitat de mantenir la infraestructura de maquinari, oferint al proveïdor de SaaS la possibilitat de contractar programàticament un nombre arbitrari i dinàmicament canviant de màquines, comunicacions i emmagatzematge. Això és possible gràcies a la tecnologia de virtualització d'equips, mitjançant la qual…

- L'assignació de recursos de còmput (virtuals) és simple i ràpida.
- La capacitat dels recursos de còmput es configura fàcilment.
- És molt senzill instal·lar una imatge de sistema sobre una màquina virtual

Un proveïdor de SaaS pot així usar els serveis dels proveïdors IaaS per a incrementar/decrementar el nombre de recursos dedicats a executar el programari del seu SaaS, segons la demanda provinent dels seus usuaris. Això fa, a priori, factible l'ajust de la despesa incorreguda pel proveïdor SaaS a allò imprescindible per a satisfer la QoS compromesa amb els seus usuaris, permetent-li predir amb gran precisió els seus costos en funció de la demanda, reduint els costos fixos, i ajustant al màxim el cost per als seus usuaris.

Notar que, d'acord amb el punt 2 a dalt indicat, el programari del SaaS ha de ser capaç d'escalar per unitats de màquina, no per la grandària de les mateixes (per exemple, velocitat de CPU, quantitat de memòria…). Aquest tipus d'escalat es coneix com *escalat horitzontal*, en contraposició a l'escalat típic dels grans servidors consistent a augmentar la potència d'un servidor concret, incrementant el nombre dels seus processadors, velocitats, memòria cau, memòria principal, etc. i que es coneix com *escalat vertical*. Cal tenir en compte que l'escalat vertical, com hem esmentat més amunt, té els seus límits en les possibilitats tecnològiques de construcció d'ordinadors.

El proveïdor IaaS ofereix un servei simple (molt més que els SaaS que s'executen sobre els seus recursos), per la qual cosa pot aspirar a comptar amb un gran nombre d'usuaris (proveïdors SaaS) que rendibilitzen la inversió que necessàriament ha de fer tant en els seus centres de dades com en el programari para automatitzar la seua gestió.

4.3.6 SaaS sobre IaaS

IaaS introdueix un model de "pagament per ús", que constitueix una característica central de la computació en el núvol, i trasllada aquest mateix model als usuaris dels sistemes SaaS. D'altra banda IaaS facilita la creació de SaaS que s'adapten a la càrrega generada pels seus usuaris, mantenint la propietat d'elasticitat.

Per a ser rendible, el proveïdor SaaS està obligat a un ús eficient dels recursos en els quals la majoria dels costos són variables. No hi ha costos directes per reservar certa capacitat (compra o compromís de pagament), i allò que s'estalvie beneficiarà a l'usuari SaaS, donant aparició a un mercat competitiu de serveis.

Els proveïdors IaaS prenen els riscos de la inversió directa (compra dels recursos físics) sota el supòsit que existisca una gran població de proveïdors SaaS, els qui, al seu torn, facilitarán serveis a un alt nombre d'usuaris SaaS provocant una gran demanda de recursos virtualitzats.

El proveïdor SaaS encara exerceix diversos rols, a més del seu natural com a proveïdor de serveis de *programari*:

- Ha de gestionar l'assignació de recursos de *maquinari.*
- Ha de gestionar les imatges de sistema a instal·lar, les seues actualitzacions i la base de programes a utilitzar sobre aquests sistemes.
- Ha d'implantar la seua pròpia estratègia de gestió de serveis, desenvolupant mecanismos de monitoratge i actualització.

4.3.7 Platform as a Service (PaaS)

L'ús d'un IaaS i de programari potencialment capaç de ser escalat horitzontalment cobreix els punts 1 i 2 dels requeriments establits anteriorment per a aconseguir un servei elàstic.

No obstant això, per a complir el punt 3, el proveïdor de SaaS encara ha d'enfrontar-se als següents problemes:

1. Detectar a cada moment els recursos necessaris per a satisfer una QoS.
2. Modificar la configuració del desplegament de programari que constitueix el SaaS, sense produir interrupcions del servei, incomplint el seu objectiu de QoS.

L'ideal és comptar amb un *executiu*: aplicació que inspecciona tant la información estructural d'un SaaS com la informació d'execució del servei, per a prendre decisions d'elasticitat de forma automàtica, eliminant aquesta responsabilitat del SaaS.

- Aquest tipus d'executiu sobre sistemes elàstics s'anomena PaaS (*Platform as a Service*).

Quan un SaaS és integrat per a funcionar sobre un PaaS, el SaaS ha de complir una sèrie de *requeriments* addicionals. En particular, el SaaS ha de ser ara especificat amb informació extra que puga ser utilitzada pel PaaS per a prendre decisions d'escalat (metadades). També serà necessari que els components del SaaS s'adherisquen a unes normes d'interacció amb el PaaS, mitjançant un API/protocol adequat.

La funció realitzada per un PaaS sobre un IaaS és similar a la realitzada per un Sistema Operatiu (l'executiu) sobre un ordinador. En tots dos casos, el sistema operatiu (PaaS) decideix quin i quants recursos és necessari aportar a cada aplicació en execució (SaaS), basant les decisions en els recursos disponibles, les necessitats de les altres aplicacions (altres SaaS), i les metadades disponibles sobre l'aplicació en execució.

El PaaS ha d'abordar els aspectes següents:

- Models de configuració i de gestió del cicle de vida (incloent les relacions de dependència entre components)
    - Mecanismes de composició, configuració, desplegament i actualització
- Model de rendiment
    - Monitoratge automàtic de paràmetres rellevants
    - Expressió de punts d'elasticitat

- Reconfiguració automatitzada en funció de la càrrega

4.4 Resum

(A la dreta es mostra l'estructura ideal en nivells)

La computació en el núvol (CC) se centra en l'eficiència i la facilitat d'ús:

- Compartició eficient dels recursos
  - Consumir solament el que es necessite
  - Pagar solament pel que s'ha utilitzat
- Adaptació senzilla a una quantitat d'usuaris variable
- Facilitar formes senzilles per a desenvolupar i proveir un servei

S'han identificat tres models de serveis en el núvol:

1. *Software as a Service* (SaaS)
   - El seu objectiu és facilitar aplicacions com a servei a un gran nombre d'usuaris
2. *Platform as a Service* (PaaS)
   - Aconsellable per a automatitzar la gestió de recursos per als SaaS i la fàcil creació i desplegament d'aquests serveis
3. *Infrastructure as a Service* (IaaS)
   - Proporciona elasticitat per als sistemes SaaS

Des de la perspectiva dels usuaris, CC és com un retorn a l'era dels "*mainframes*".

5 Paradigmes de programació

Un model de sistema bàsic promou una comunicació asincrònica mitjançant pas de missatges: després d'executar esdeveniments d'enviament, els agents poden continuar amb la seua execució sense bloquejar-se. Així mateix, els agents avançaran les seues pròpies tasques, gestionant els seus esdeveniments de recepció quan aquests succeïsquen (també de manera asincrònica, sense romandre pendents de les recepcions).

Un estil de programació apropiat per a representar els algorismes que cada agent execute és el següent:

1. Cada algorisme està representat por una col·lecció d'accions $[A_1,\ldots,A_n]$, dissenyades per a ser executades atòmicament (sense interrupcions).

2. Cada acció A_i, està associada a una condició C_i (també coneguda com *Guarda*). Quan la condició C_i siga certa, l'acció A_i estarà habilitada i podrà ser seleccionada per a executar-se. En cada instant, cada agent P selecciona una de les seues accions habilitades per a execució.

3. Les condicions poden dependre de l'estat local.

4. Les condicions poden dependre de la recepció d'un missatge.

5. Les accions poden enviar un missatge al final de la seua execució.

Hi ha diversos llenguatges de programació que utilitzen aquests principis, cadascun oferint una sintaxi diferent per a expressar variables, condicions i accions.

L'especificació típica d'algorismes utilitza alguna variació de la sintaxi del llenguatge C per a expressar les accions, les condicions i les definicions d'estructures de dadas (incloent l'estructura dels missatges) en el seu pseudocodi.

En programes asincrònics, les estructures de dades han de ser definides per a influir sobre com les accions s'executaran una vegada habilitades. Això és fàcilment aplicable en especificar algorismes xicotets o amb cohesió forta. Així i tot, en situacions reals aquesta aproximació pot resultar tediosa i propensa a errors.

Per a resoldre aquest assumpte, alguns llenguatges de programació permeten la construcció dinàmica de les accions disponibles, que hauran sigut ja total o parcialment configurades amb les dades rellevants per a la seua gestió una vegada estiguen habilitades. Així es redueix la necessitat de mantenir dades en un context global per a permetre la seua execució.

Una forma típica de construir les accions és com a clausures en un llenguatge de programació funcional. La clausura comporta l'accés a aquelles variables (possiblement locals a la funció) que la funció haja de consultar o modificar durant la seua execució. La guarda especifica com una condició la conclusió d'alguna acció asincrònica (recepció d'un missatge, finalització d'una crida al sistema…). Podrà haver-hi variables "lliures" que seran especificades com a arguments de les funcions. En aquests casos les guardes han de referir-se a les fonts de dades per a aquests arguments, i el "run-time" ha d'associar-les quan cride a la funció (i s'empre la seua clausura) per a la seua execució.

Per exemple, en un entorn basat en JavaScript com nodejs, els "*callbacks*" són clausures de funció. Aquestes funcions poden especificar paràmetres extres que han de ser "omplits" abans que el *callback* siga executat (una vegada habilitat). Aquesta estratègia evita la utilització d'estructures globals que mantinguen els dades extres necessàries perquè l'acció siga executada.

5.1 Paradigma concurrent amb estat compartit

Els agents es modelen com a executors d'accions atòmiques en cada esdeveniment. Si pensem en un servidor, això significa que el procés espera l'arribada d'un missatge de petició i llavors executa aquesta fins a la seua finalització abans de retornar el resultat al client, sense admetre cap tipus d'interrupció. Aquesta va ser, de fet, la forma de programar els primers servidors.

L'avantatge d'aquesta aproximació és que l'atomicitat està garantida: mentre una petició està sent processada cap altra petició pot interferir.

Així i tot, hi ha un problema amb aquesta aproximació simplista. Els servidors sovint necessiten fer peticions sobre altres elements: altres servidors o el sistema operatiu (que també pot veure's com un altre servidor). L'espera del resultat d'aquestes peticions bloquejaria el

servidor i, per tant, el deixaria incapaç d'atendre noves peticions d'altres clients. El resultat final és un sistema amb molt baixa capacitat de servei.

Una tècnica utilitzada per a evitar aquesta situació de bloqueig va ser que el servidor principal llançara un nou procés per cada petició rebuda. Aquest procés secundari era qui realment servia la petició del client.

La planificació de processos feta pel sistema operatiu permetia que tant el servidor principal com els processos secundaris s'executaren concurrentment. El servidor principal, després de delegar la gestió de la petició en el procés creat, tornava per a esperar noves peticions dels clients.

Així la capacitat de resposta del servei millora ja que el bloqueig d'un procés secundari no bloqueja la capacitat de procés del servidor principal. Aquest seguirà atenent peticions noves. L'estat global entre el servidor principal i tots els subprocessos era compartit utilitzant fitxers (un espai globalment accessible dins d'una instància de sistema operatiu). L'accés a aquests fitxers va necessitar ser coordinat pels processos per a implementar l'atomicitat en les operacions utilitzades per la lògica del servidor. La coordinació es va obtenir utilitzant mecanismes del sistema operatiu per a accedir a fitxers.

Les dificultats que comportaven aquests mecanismes de compartició d'estat van conduir a un estil de programació en el qual la compartició d'estat global es va minimitzar. Gran part de la informació que necessitaven els processos secundaris es preparava per al seu ús exclusiu (incloent la petició del client). Així la coordinació a través de fitxers compartits es va reduir al mínim possible i va haver-hi molt pocs problemes derivats del control de concurrència entre processos secundaris.

La generació de subprocesos resulta ineficient tant en temps com en recursos sol·licitats al sistema operatiu. Conseqüentment altres aproximacions basades en múltiples fils d'execució (on la concurrència es dóna internament als processos) van ser explorades. El servidor ja no genera nous processos per a gestionar l'arribada de peticions. En el seu lloc, genera fils d'execució concurrents en el seu propi espai d'adreces.

Amb aquesta aproximació, cada petició és lliurada a un fil diferent. La compartició d'estat s'aconsegueix trivialment entre tots els fils ja que tots ells comparteixen l'espai d'adreces del procés en el qual s'executen. Això facilita un estil de programació en el qual l'estat compartit global és utilitzat per a comunicació i coordinació de fils, requerint l'ús de mecanismes de control de concurrència (semàfors, monitors…) per a implantar l'"atomicitat" implícita exigida per la lògica del servidor.

Igual que la generació de processos, els fils poden suspendre's independentment sense bloquejar la resta del servidor, que roman actiu. A més, la gestió de fils és més eficaç quant a consum de recursos que la gestió de processos (el sistema operatiu té menys treball) i l'accés a l'estat compartit és molt més ràpid.

Aquest ha sigut el patró predominant per a construir servidors fins al moment. De fet l'ús de múltiples fils d'execució es manté directament en alguns llenguatges de programació. Llenguatges sense suport natiu per als fils també poden ser utilitzats, enllaçant amb

biblioteques que faciliten la creació i gestió de fils, així com l'accés a alguns mecanismes de control de concurrència.

Així i tot hi ha almenys dos desavantatges clars en la utilització de fils:

1. Els fils, malgrat ser més barats que els processos, també consumeixen un bon nombre de recursos. La creació i la destrucció de fils són operacions relativament complexes i, encara que puguen utilitzar-se "pools" de fils per a rebaixar els costos de creació i destrucció, han de sintonitzar-se acuradament per a evitar un ús abusiu de recursos. A més, el desenvolupament de mecanismes de control de concurrència també incorre en notables sobrecàrregues. Si considerem que la motivació principal de l'ús de fils és permetre el seu bloqueig independent, el que realment s'obté és que molts dels fils generats no estarà fent res la major part del temps.

2. La programació concurrent amb moltes variables compartides és propensa a errors. Aquests errors poden conduir a inconsistències en l'estat (l'atomicitat serà violada) o al bloqueig del servidor (l'atomicitat es garanteix, a costa de comprometre el progrés). Malgrat l'èmfasi que trobem en molts plans d'estudi sobre programació concurrent basada en sistemes multi-fil, la situació no sembla que puga millorar molt. Això imposa barreres a la velocitat en la qual el codi pot ser produït i a la fiabilitat del producte final.

5.2 Paradigma asincrònic (o dirigit per esdeveniments)

Els entorns de programació asincrònica operen de manera semblant al model de programació guarda-acció presentat anteriorment. No són nous, existeixen des de fa temps com a sistemes dirigits per esdeveniments, específicament en el camp de la interacció persona-ordinador (L'usuari està constantment enviant peticions al programa que maneja la interfície d'usuari. Aquestes peticions han sigut tradicionalment manejades com a esdeveniments asincrònics dins d'un bucle d'esdeveniments).

En la pràctica, els entorns de programació asincrònics utilitzen un sol fil en un únic procés. Per tant, l'estat mai està compartit per diverses activitats i no es necessiten mecanismes de control de concurrència. S'ofereix un model similar al dels servidors de la primera generació, implantats mitjançant un sol procés.

Els entorns de programació asincrònica han de superar dues dificultats:

1. Evitar el bloqueig de l'únic fil disponible en el procés (reproduint el desavantatge dels servidors amb un sol procés).

2. Que el desenvolupament dels programes siga raonablement senzill. Això requereix:

    a. Una estructuració intuïtiva de l'estat que cada acció necessitarà manejar quan haja d'executar-se.

    b. Una forma fàcil d'expressar les condicions.

Per a evitar la suspensió de les activitats es requereix que totes les peticions efectuades en un entorn asincrònic siguen, al seu torn, asincròniques (és a dir, que no impliquen bloquejos). El repte principal es donarà en les crides als serveis del sistema operatiu. Sovint, algunes d'aquestes crides són bloquejants i han de ser convertides utilitzant tècniques multi-fil pel propi "run-time" (però no pel programari de l'aplicació, que només observarà un fil).

Una manera comuna d'evitar el bloqueig és la utilització de "callbacks". En fer una petició, un dels arguments és una funció. Quan la petició acaba, aquesta funció es crida amb el resultat de la petició com a argument:

```
function handleResults(r) {
        …
}
readFile(f, handleResults);
next_instruction();
```

Bàsicament, el que està fent aquest programa és:

1. Crea una condició: "La crida a *readFile* acaba".
2. Defineix una acció mitjançant una definició de funció: *handleResult*.
3. Fixa aquesta condició ("La crida a *readFile* acaba") com la guarda de handleResult.
4. Continua executant la resta del programa (en aquest cas, començant amb next_instruction), sabent que *handleResult* serà executat MÉS TARD (s'acaba de crear la condició mentre s'executava alguna acció ATÒMICA: no pot ser interrompuda per una altra acció, encara que la seua guarda fóra certa i l'acció fóra habilitada tan aviat com es creara).

Quan *handleResult* siga invocada, rebrà les dades generades per la petició *readFile*. Així, l'estil de programació basat en callbacks sembla resoldre les nostres preocupacions: evita els bloquejos convertint totes les peticions en asincròniques, facilita la creació de condicions dinàmiques (la finalització de peticions), i facilita el maneig de l'estat que necessitaran les accions (paràmetres del callback).

En realitat, encara es necessita: (a) ser prudent amb les activitats que necessiten molt de temps i (b) tenir cura especial en la gestió d'estat.

Les operacions llargues acapararan a l'únic fil, retardant excessivament l'execució de qualsevol altra acció habilitada. Això pot generar sistemes poc eficients. Si un programador no és prudent, el "run-time" no podrà fer res per a evitar aquesta situació.

Per a evitar-ho, el programador hauria de dividir aquesta operació perllongada en fragments que admeteren execució atòmica. Això pot ser fàcil amb un suport apropiat de l'entorn de programació.

Per exemple, suposem que volem computar la funció factorial sense acaparar l'únic fil disponible. Podríem utilitzar aquesta aproximació:

```
function factorial(n) {
  if (n == 0) return 1;
  executeLater(factorial, n-1)
}

factorial(10000);
```

En aquest codi, hem suposat que el "run-time" té una operació, `executeLater`, que pren una funció i l'argument que haja de rebre, actuant d'aquesta manera:

1. Crea una condició que serà immediatament certa.
2. Associa la condició amb una acció que consisteix a executar la funció rebuda en el primer paràmetre, amb els arguments rebuts en el segon paràmetre.
3. Acaba l'execució de la funció.

Així, quan *factorial*(10000) és executat, la qual cosa hauria sigut una computació llarga i contínua dins de l'únic fil, serà convertit en 10000 accions molt curtes, donant a les altres accions habilitades una oportunitat d'executar-se.

La gestió d'estat es dóna com a resultat del pas d'arguments als callbacks. Molts "run-times" permeten accedir a l'estat global del procés. L'accés a aquest estat global no és concurrent. Desafortunadament, en alguns casos extrems en els quals es realitzen altres invocacions, la hipòtesi d'atomicitat pot trencar-se i s'ha de parar esment per a no introduir inconsistències. Per a això s'han de prendre certes precaucions en escriure les condicions que habilitaran als diferents callbacks.

A més, quan s'escriguen els callbacks serà necessari en molts casos passar informació addicional, a part de la rebuda pel callback en els seus paràmetres. Diferents llenguatges de programació habiliten això de manera diferent. En entorns de programació funcional asincrònica (com per exemple, JavaScript quan s'utilitze `node`) la forma habitual és passant "clausures" com callbacks, on les clausures encapsulen l'accés a múltiples àmbits niats de definició de variables.

6 La Vikipèdia com a Cas d'Estudi de TSR

El propòsit d'aquest apartat és il·lustrar el tipus de problemes que acompanyen als Sistemes d'Informació en Xarxa, partint de l'estudi d'un cas concret del que es van desprenent successivament oportunitats i necessitats de millora, que al seu torn provoquen nous problemes que han de ser abordats. La primera sigla de l'assignatura, T, representa les tecnologies aplicables per a resoldre aquest tipus de problemes.

Ha d'entendre's que la varietat de sistemes baix estudi no pot ser representada únicament amb un cas, però que, a títol introductori, facilita un context en el qual té sentit identificar les necessitats i proposar solucions raonables.

El servei més conegut que pertany al nostre àmbit d'estudi és la World Wide Web, que coneixeu tant per la seua utilitat universal com per haver sigut objecte d'estudi en l'assignatura de Xarxes. D'entre els llocs web existents, alguns destaquen mundialment per la seua envergadura, com:

- el cercador de Google[2], amb més d'un milió de servidors,
- la xarxa social Facebook[3], amb més de mil milions d'usuaris,
- la web dels ferrocarrils xinesos[4], la més visitada amb més de mil milions de clics diaris,
- les tendes online Amazon, eBay i Alibaba, les vendes de les quals en 2013 van totalitzar[5] quasi cent mil milions de dòlars,
- els serveis de vídeo de Netflix[6] i YouTube
    - En 2015, Netflix compta amb 60 milions de subscriptors[7] que descarreguen 45 GB al mes… cadascun! A EUA i Canadà representa el 35% del tràfic total.
    - Les xifres[8][9] de YouTube són més esgarrifoses encara, amb més de mil milions d'usuaris, i set mil milions de reproduccions al dia en 2015[10]!

Potser podria pensar-se que siguen excepcions que requereixen solucions especialitzades, però el tipus de problemes als quals s'enfronten no deixen de repetir-se en altres serveis més modests ni en uns altres menys duradors:

- La popularitat d'una web modesta, dissenyada per a una activitat de magnitud moderada, es veu propulsada per aparèixer esmentada en algun mitjà de comunicació d'ampli abast, ocasionant l'"*efecte Slashdot*[11]", que produeix una "mort per èxit" del servei al no poder respondre satisfactòriament a la demanda.
- Un projecte nascut per a cobrir mundialment un esdeveniment temporal, ha d'estar preparat per a oferir un molt elevat nivell de servei durant el període de vida de l'esdeveniment. Exemples destacables són les retransmissions esportives associades a

[2] https://www.google.com/
[3] https://www.facebook.com/
[4] http://www.12306.cn/, segons article en http://www.chinadaily.com.cn/china/2012-01/09/content_14406526.htm
[5] http://uk.reuters.com/article/2014/09/19/alibaba-ipo-idukl1n0rk1kw20140919
[6] https://www.netflix.com/
[7] http://expandedramblings.com/index.php/new-updated-netflix-stats/
[8] https://www.youtube.com/yt/press/statistics.html
[9] http://expandedramblings.com/index.php/youtube-statistics/
[10] http://www.forbes.com/sites/edmundingham/2015/04/28/4-billion-vs-7-billion-can-facebook-overtake-youtube-as-no-1-for-video-views-and-advertisers/
[11] https://en.Vikipèdia.org/wiki/slashdot_effect

tornejos de tennis (Wimbledon[12], Roland Garros[13]…), els JJOO d'estiu[14], o els campionats mundials de futbol[15].

Desitgem seleccionar un lloc web en el qual identificar els problemes i les solucions que els seus dissenyadors han ideat; això requereix l'accés a una bona documentació, la qual cosa no és habitual en empreses privades la infraestructura de les quals es qualifica com a confidencial. D'altra banda, acadèmicament, és aconsellable la simplificació de detalls per a facilitar la comprensió del funcionament global.

A la llum de les consideracions anteriors, i donada la seua rellevància social que la va fer mereixedora del premi de Cooperació Internacional 2015 de la Fundació Princesa d'Astúries[16], la **Vikipèdia** és una molt bona candidata.

6.1 Vikipèdia en l'actualitat

El jurat del premi resumeix les dades rellevants de la Vikipèdia en la data de concessió del guardó:

"*Vikipèdia és una enciclopèdia digital d'accés lliure creada en 2001 i que està escrita en diversos idiomes per voluntaris de tot el món, els articles dels quals poden ser modificats per usuaris registrats. Utilitza la tecnologia wiki, que facilita l'edició de continguts i l'emmagatzematge de l'historial de canvis de la pàgina. Fundada per l'empresari nord-americà Jimmy Wales, amb l'ajuda del filòsof Larry Sanger, està gestionada des de 2003 a través de la Fundació Wikimedia.*

Vikipèdia va començar el 15 de gener de 2001 com a complement d'una enciclopèdia escrita i avaluada per experts anomenada Nupedia. […]. Tots dos projectes van coexistir fins que l'èxit de Vikipèdia va acabar eclipsant a Nupedia, que va deixar de funcionar en 2003. El creixement de l'enciclopèdia, que figura entre els deu llocs web més visitats del món, ha sigut continu i actualment té més de 35 milions d'articles en 288 idiomes […]. La Vikipèdia en anglès és la més àmplia, amb més de 4.800.000 articles. Vikipèdia compta, actualment, amb més de 25 milions d'usuaris registrats, dels quals més [de] 73.000 editors són actius i té uns 500 milions de visitants únics al mes.

La Fundació Wikimedia, creada en 2003 i amb seu en Sant Francisco (EUA), és la que dirigeix i gestiona Vikipèdia. Té a més altres projectes que complementen l'enciclopèdia, tots multilingües, lliures i recolzats en la tecnologia wiki […]."

Tenint en compte[17] el grau de repercussió de la Vikipèdia (prop de 10,000 milions de pàgines accedides al mes, cinquè en el rànquing d'Alexa), un reduït nombre de treballadors (uns 200) i sense més finançament (prop de 50 milions de dòlars) que les donacions de particulars,

[12] http://www.wimbledon.com/index.html
[13] http://www.rolandgarros.com/en_fr/index.html
[14] http://www.olympic.org/london-2012-summer-olympics
[15] http://es.fifa.com/worldcup/archive/brazil2014/
[16] http://www.fpa.es/es/premios-princesa-de-asturias/premiados/2015-Wikipedia.html?especifica=0&idCategoria=0&anio=2015&especifica=0
[17] https://annual.wikimedia.org/2014/

s'entén que els seus responsables hagen sigut molt selectius a l'hora d'invertir en el funcionament del seu servei web.

6.2 MediaWiki, el motor de la Vikipèdia, és un sistema LAMP

El programari que constitueix el motor de la Vikipèdia ha sigut desenvolupat per ells mateixos i perfeccionat al llarg del temps. Es tracta d'un programari en constant evolució basat en el concepte de Wiki[18]: *"(de l'hawaià wiki, 'ràpid')[2] és el nom que rep un lloc web les pàgines del qual poden ser editades directament des del navegador, on els usuaris creen, modifiquen o eliminen continguts que, generalment, comparteixen*". Es tracta d'un sistema d'escriptura col·laboratiu d'ús senzill però amb grans capacitats.

La Vikipèdia ha impulsat aquesta tecnologia, desenvolupant programari des de gener de 2001[19], data en què van crear UseModWiki[20]:

- Escrit en PERL, sobre LINUX, i emmagatzemant la informació en arxius en lloc de base de dades.

Aquest programari va ser reemplaçat per Phase-II al començament de 2002, destacant per usar un motor wiki en PHP, però al juliol va ser de nou millorat (Phase-III = MediaWiki!) per a adaptar-se al creixement del projecte. És significatiu que fins a juny de 2003 no s'afegira un segon servidor, incloent en ell la base de dades (se separa del servidor web) i les webs en altres idiomes diferents de l'anglès. Més detalls en els apartats 1 a 4 del capítol 12 (MediaWiki[21]) del llibre "*The Architecture of Open Source Applications, Volume II*"

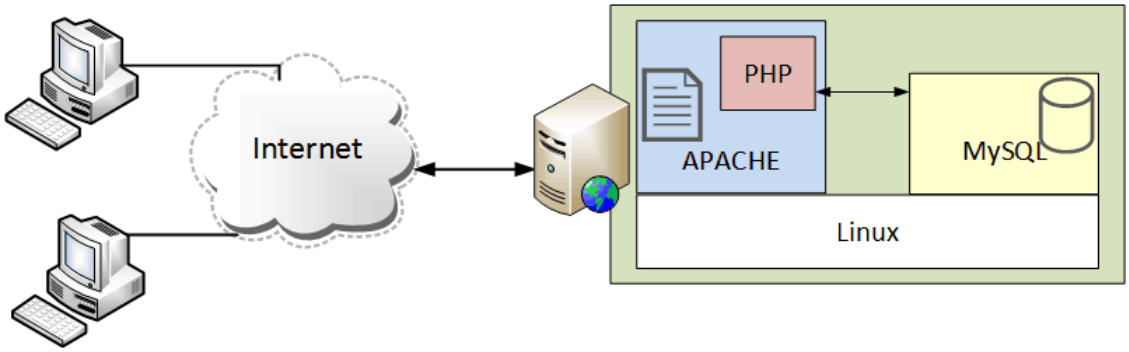
Tot aquest programari ha continuat evolucionant, però comparteix un nucli tecnològic: són sistemes LAMP. Aquesta combinació gira entorn de les seues 4 inicials: Linux + APACHE + MySQL + PHP.

- Linux és un sistema operatiu tipus UNIX
- APACHE és un servidor de web. Rebrà peticions HTTP dels clients, i executarà les accions necessàries per a retornar les respostes
- MySQL és un gestor de bases de dades relacionals
- PHP és un llenguatge de scripting lleuger i senzill, amb biblioteques de tot tipus, i amb facilitat per a produir documents d'hipertext i per a interactuar amb SGBD relacionals.

Una plataforma LAMP encaixa amb el model de 3 capes de les aplicacions client/servidor:

1. La capa **d'interfície d'usuari** conté els components que formen les interfícies d'usuari de les aplicacions (lògica de presentació). Es correspon amb els documents d'hipertext que retorna APACHE i visualitza el navegador del client.
2. La capa **d'aplicació** (o lògica de negoci) conté les regles de funcionament de l'aplicació, sense integrar dades concretes. Es correspon amb part del codi escrit en PHP.
3. La capa **de dades** (o de persistència) conté la informació que els clients manipulen mitjançant els altres components de l'aplicació. Es correspon amb la BD gestionada per MySQL.

[18] https://es.wikipedia.org/wiki/wiki
[19] https://en.wikipedia.org/wiki/history_of_Vikipèdia#Hardware_and_software
[20] https://en.wikipedia.org/wiki/usemodwiki
[21] http://www.aosabook.org/en/mediawiki.html

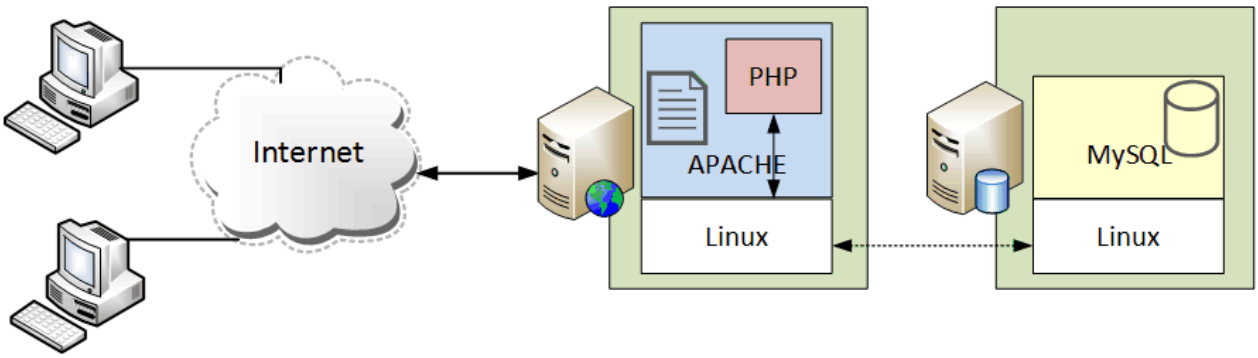


La seqüència **simplificada** de l'atenció d'una petició d'un client podria ser:

0. Originalment hi ha un procés APACHE i un altre MySQL a l'espera de rebre treball.
1. El client envia la petició HTTP al servidor. Aquest client queda en espera.
2. El servidor APACHE és despertat pel S.O. per a atendre la sol·licitud
3. APACHE determina que ha d'executar-se un programa PHP, al que passa les dades. Com l'intèrpret de PHP és una extensió seua (forma part del procés), APACHE no queda en espera.
4. El programa s'executa emetent ordres SQL que el SGBD rep. El programa PHP (i APACHE) queda tot esperant que el procés SGBD li retorne una resposta.
5. El servidor MySQL és despertat pel S.O., arreplega la sol·licitud i l'executa, retornant la resposta al programa PHP. MySQL queda a l'espera d'una nova sol·licitud.
6. Les interaccions 4 i 5 poden repetir-se. Al final, el programa PHP confecciona una pàgina de resposta que APACHE envia per la xarxa. Finalitzada l'acció, el programa acaba i APACHE queda tot esperant una nova sol·licitud.
7. El client rep el resultat, i el seu navegador pot visualitzar-ho en pantalla.

Per la forma en què s'intercomuniquen, com a processos, no és necessari que el SGBD (actuant com a servidor) i el programa PHP (actuant com a client) s'executen en el mateix equip.

- Aquesta distinció entre node (equip) i component (en aquest cas els servidors APACHE i MySQL) té extrema importància en el plantejament d'un sistema distribuït.
- No obstant això, excepte canvis substancials, no és possible que el programa PHP s'execute en un equip diferent del que ocupe APACHE.



Tornant al tema: què és MediaWiki exactament? MediaWiki és un cas concret de sistema LAMP que inclou scripts per a la part PHP de la il·lustració anterior, juntament amb la configuració necessària dels servidors, alguna BD i taules, i alguns continguts d'hipertext.

6.3 Usant MediaWiki en el context de la Vikipèdia

La major part de la informació estadística sobre la Vikipèdia gira entorn del nombre d'editors, entrades i idiomes disponibles. Addicionalment es poden trobar xifres sobre el nombre d'accessos d'usuaris lectors, però haurem de processar aquesta informació per a esbrinar quins recursos es necessiten.

Per exemple, una aproximació per a conèixer els recursos de xarxa necessaris pot consistir, donada una taxa d'accessos mensual, a multiplicar la grandària de pàgina per aquesta taxa, obtenint l'ample de banda brut mensual consumit en lectures.

- Exemple d'article curt en anglès: *yottabyte* (840 KB segons la consola d'inspecció de Firefox)
- Idem d'article llarg[22]: *computer* (4200 KB)

El consum de xarxa mensual, es troba entre 8400 TB[23] i 42000 TB, per als quals, prenent un patró d'accés constant però irreal, es requereixen velocitats entre 25.93 Gbps i 129.63 Gbps. Si observem els moments de major ús segons les seues estadístiques[24], s'aconsegueixen màxims que superen els 25000 accessos/s, moment en el qual es requereix un **ample de banda entre 168 Gbps i 840 Gbps**. La infraestructura de xarxa no aconsegueix aquestes xifres amb facilitat.

Pot atendre's tota aquesta demanda des d'un únic servidor? Posem-ho a prova.

6.3.1 Accés a Internet

Un proveïdor d'accés a Internet pot allotjar un servidor en les seues instal·lacions perquè aprofite la infraestructura de xarxa d'aquest ISP. Prenent dades[25] de l'any 2014, a EUA, la major velocitat *convencional* aconseguia quasi els 60 Mbps. Fins i tot amb dades més recents, en desembre de 2017, el millor ample de banda el donava Xfinity als EUA amb 2000 Mbps.

En el pitjor cas ens farien falta **420 línies de la millor connexió d'Internet** del món!!!

Cal insistir que els responsables de la Vikipèdia no tiren els diners per la finestra, de manera que ha d'haver-hi alguna solució raonable.

La primera alternativa consisteix a evitar enviar tota aquesta quantitat d'informació. Moltes tècniques pròpies de la web són aplicables, com reutilitzar fulla d'estil, icones i scripts. Quan el navegador els ha carregat, pot reutilitzar-los. Altres tècniques aplicables aconsegueixen reduir encara més la informació, però no aconseguim un estalvi suficient; al cap i a la fi comptem amb milions de clients i la *cache* del navegador és pròpia de cadascun.

La segona alternativa consisteix a "desconcentrar" el servei. Això suposa dispersar múltiples servidors per les xarxes mundials i comptar amb l'ample de banda agregat de tots ells. La caixa de Pandora s'obri amb les paraules màgiques "múltiples servidors": no es tracta del mateix servei? Aquesta solució introdueix els seus propis problemes, destacant:

[22] Realment no és la major grandària, però en aquest document és irrellevant
[23] 10,000,000,000 accessos/mes = 3,858.02 accessos/s, que multipliquem per 437.10 KB i per 8 bits, i representem en Gbps
[24] https://grafana.wikimedia.org/?orgId=1
[25] http://www.pcmag.com/article2/0,2817,2465506,00.asp

- Els servidors són clons idèntics?, i per on "*s'entra*"?
- Com es gestionen les operacions d'escriptura d'articles en la Vikipèdia?
- Què ocorre si algun es desincronitza respecte a la resta? Han de comunicar-se entre si o hi ha una còpia central?
- Algun servidor de recanvi per a casos de fallada?

Els nous problemes poden assumir-se si els beneficis excedeixen els costos, però cal aplicar molta intel·ligència per a identificar cadascun dels nous punts d'error i donar-los solució.

Podem afirmar que multiplicar elements sempre és problemàtic, perquè la fiabilitat d'un sistema depèn de la dels seus components: un increment en el nombre de peces constitueix un problema potencial. Suposem que es precisa disposar de 10 rèpliques per a millorar el rendiment, i que la probabilitat que un dia una rèplica concreta falle és de l'1 per mil (0.1%); quina és la probabilitat de que no falle cap vegada al llarg d'un mes?

- Perquè no falle una en un dia: 1-0.001
- Perquè no falle en 2 dies[26]: (1-0.001)*(1-0.001)= $(1-0.001)^{\mathbf{2}}$
- Perquè no falle una en un mes: $(1-0.001)^{\mathbf{30}}$=97.04%
- Perquè no falle cap de les 10 en un mes: $((1-0.001)^{30})^{\mathbf{10}}$=74.07%

A manera de primera conclusió parcial: *quants més components, menys fiabilitat*; incrementar el rendiment afegint components empitjora la fiabilitat.

I com es pot arreglar? Amb un disseny acurat en el qual un component amb fallades puga ser detectat i reemplaçat, fent-se càrrec algun altre del treball interromput. És més fàcil dir-ho que fer-ho!

Com a segona conclusió, *els problemes introduïts per la replicació se solucionen amb més replicació*. La primera "replicació" es dirigeix a incrementar el rendiment, mentre que la segona intenta resoldre els problemes de fiabilitat introduïts o agreujats per la primera.

Aquestes consideracions són aplicables, amb les seues particularitats, a tots els components del sistema LAMP en què es basa la Vikipèdia.

6.3.2 El servidor web APACHE (objectes estàtics) i els scripts en PHP (resultats dinàmics)

Aquest programari intervé per a rebre les sol·licituds dels clients, pel protocol HTTP, i decidir si ha de retornar un contingut emmagatzemat o executar un programa PHP. Una visió simplificada d'aquesta forma facilita la separació d'objectius. En el primer cas el servidor és un intermediari entre el client i els continguts; en el segon serveix com a transmissor d'informacions del client (p. ex. una interrogació pel cercador local) a una aplicació que ha de construir una resposta a mida.

La primera part pot ser accelerada amb certa facilitat: més memòria! Si es poguera col·locar tota aquesta informació estàtica en memòria física, podria accedir-se a la mateixa amb molta rapidesa, encara que a un preu… Com no és possible representar tota la informació en memòria, se sol usar algun sistema intel·ligent que, adaptativament, accelera l'accés a les

[26] que no falle el primer **NI** el segon.

informacions més demandades. És una memòria cau anomenada *proxy invers*, que "atrapa" i serveix les sol·licituds d'objectes estàtics (pàgines, il·lustracions…). Acceleradors capaços de realitzar aquesta funció per a peticions HTTP poden ser *Squid*, *Varnish* i *nginx*.

Poden col·locar-se diverses instàncies si una sola no aconsegueix el rendiment necessari? Ja que es tracta d'operacions de lectura independents entre si, pot establir-se algun criteri en el qual, del nom del recurs s'obtinga directament el de l'accelerador que ho conté.

- Per exemple, col·locar punts acceleradors com a lletres té l'alfabet, i que cadascun continga còpia dels objectes estàtics que comencen per aquesta lletra.

Ja vam veure que l'addició de components incrementa la probabilitat d'error, per la qual cosa hauran d'assignar-se recursos extra per a tractar els casos en els quals algun proxy invers puga fallar.

Després d'aquest filtrat de sol·licituds que referencien a objectes estàtics, les que arriben als servidors APACHE suposaran la invocació de programes PHP que construïsquen la resposta a la petició. Podem destacar 4 categories principals:

1. **Escriptures** en les quals es crea o modifica un article de la Vikipèdia: què passa amb totes les còpies d'aquest article que poden existir en els acceleradors?
    - S'ha d'assegurar la consistència de les diferents còpies en els diferents llocs. Habitualment s'enviarà algun senyal per a invalidar les còpies dels articles i obligar a *refrescar* el contingut.

2. Pàgines que **depenen de l'usuari** sol·licitant, que determinen quins continguts mostrar i quines operacions són possibles.
    - Tot allò que s'assimila al concepte de sessió HTTP necessita una continuïtat, de manera que el mateix servidor que va començar a atendre a un usuari ha de fer-se càrrec de les seues futures peticions.
    - P. ex., un usuari pot anar acumulant pàgines per a construir un llibre, o un usuari s'ha identificat amb un rol específic les capacitats del qual han de ser recordades.

3. **Consultes per a localitzar informació** depenent d'alguns termes de cerca. És una activitat pròpia del gestor de bases de dades, però pot dissenyar-se un sistema intern de *cache* de respostes, aprofitant resultats que pogueren haver-se calculat abans.
    - Fa falta un disseny molt acurat per a conèixer quan un resultat manté la seua vigència, però pot oferir grans beneficis per a certs casos.

4. **Consultes per a obtenir continguts associats**, de manera que, donat un article, puga consultar-se informació derivada: enllaços que apunten ací, informació de la pàgina, historial de canvis (amb operacions sobre els mateixos), etc.
    - La part estàtica és dominant, la qual cosa facilita una política de *caching*.

Veiem, de nou, que un ingredient fonamental per a l'acceleració del sistema es basa a mantenir còpies d'informacions per a no tornar a accedir a les mateixes o a calcular-les. Potencialment, aquesta és una font molt important d'inconsistències.

Aquesta estratègia de *caching* redueix les ocasions en què és necessari executar un programa. La importància no és menor, però encara queden peticions que no poden satisfer-se d'aquesta manera. Encara que el seu percentatge no siga alt (p. ex. 12% en els JJOO d'hivern de Nagano, 1998), el cost de processament sí que ho és, superant en molts casos al 50% del temps total acumulat.

Encara que existeixen tècniques específiques, i molt efectives, d'acceleració de programes en PHP, el gruix del temps que consumeixen està determinat per:

- El sobrecost d'atendre una petició dinàmica. Ja som selectius per a reduir la seua freqüència, però han d'atendre's els casos inevitables restants.
- El cost d'interactuar amb el gestor de la base de dades. Aquest aspecte es tracta en el següent subapartat.

6.3.3 El servidor de base de dades MySQL

La interacció amb la BD en la Vikipèdia és vital, no solament per a organitzar els articles que manté, sinó per a recopilar informacions, governar el cicle de vida dels articles, representar usuaris, mantenir estadístiques, etc.

Hem vist que una de les primeres mesures adoptades en la Vikipèdia va consistir a separar la BD per a situar-la en un altre servidor. Òbviament aquesta acció va obligar a plantejar una forma de comunicació entre el tàndem APACHE-PHP i el servidor MySQL, típicament mitjançant adreça IP i port.

Les primeres replicacions del servidor de web condueixen a un desequilibri si es manté inamovible el nombre d'instàncies, una, del SGBD: acabarà exhaust davant el devessall de peticions.

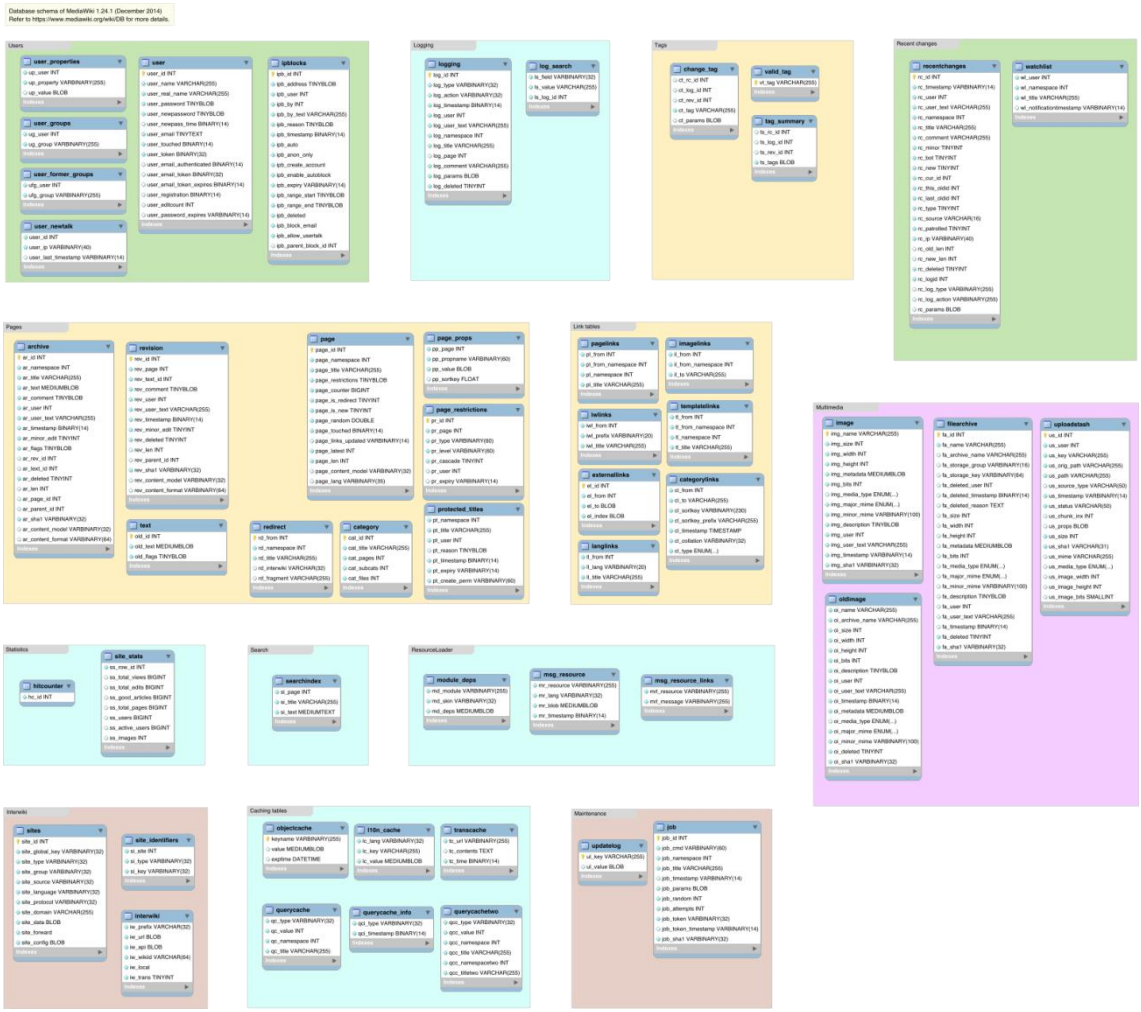
- Imagine's un restaurant amb un cambrer i un cuiner. El cambrer rep les peticions dels clients i li les transmet al cuiner. Si per a admetre molts més clients únicament es contracten cambrers, pot endevinar-se amb facilitat on es trobarà el *coll de botella*.

Col·locar més instàncies del SGBD no és una tasca senzilla si han de mantenir la mateixa informació: haurem de descobrir primer si els continguts poden separar-se en múltiples BBDD independents.

- Dependrà de l'ús que es faça de les informacions i de les relacions entre les seues taules.

Encara que aquesta primera part s'haja realitzat, segurament ens quedarà encara un resultat difícil de digerir[27]: 13 BBDD acumulant 50 taules al desembre de 2014.

[27] https://upload.wikimedia.org/Vikipèdia/commons/f/f7/mediawiki_1.24.1_database_schema.svg



Les limitacions no són preocupants en operacions de lectura si es compara amb les actualitzacions i escriptures: les BBDD que requereixen major ajuda són probablement les que pateixen més modificacions, que poden agrupar-se en diversos nivells:

1. Modificacions **molt freqüents**, que solen acompanyar als accessos de consulta: *Logging* (que afig informació en *manera append* sobre els accessos), *Statistics* (el seu propi nom ho indica), *Caching tables* (de consum intern, que accelera l'accés a algunes informacions de manera adaptativa)

2. Modificacions **per edició**, són menys freqüents que les anteriors perquè suposen l'acció d'un editor (humà?) alterant el contingut de la Vikipèdia: *Multimèdia* (representa el dipòsit d'objectes no textuals), *Recent changes* (que manté una llista de modificacions), *Pages* (és la BD d'articles) i *Link tables* (si el nou contingut referencia a altres elements)

3. Modificacions **eventuals**, que són comparativament infreqüents, com donar d'alta un usuari (*Users*), i la resta.

La freqüència d'escriptures pot ser important però no sempre determinant; p. ex. les escriptures que no condicionen lectures posteriors presenten escàs conflicte (*Logging* i

Statistics), però les escriptures que els seus resultats algú va a necessitar amb immediatesa són molt preocupants (*Caching tables*, *Pages*, …)

6.4 Arquitectura de la Vikipèdia

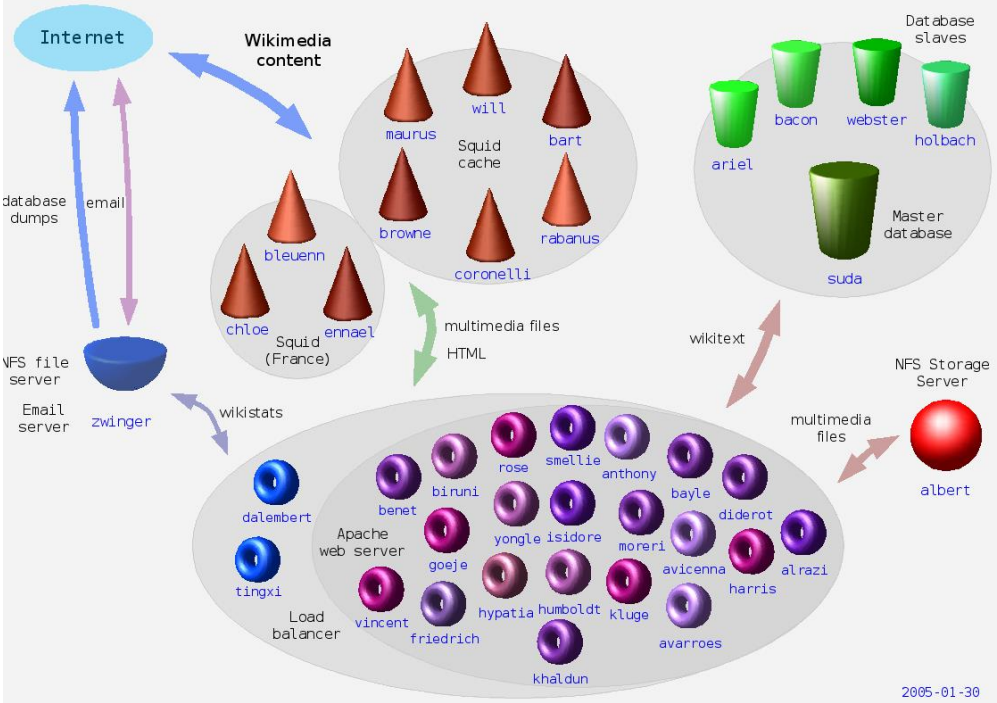
(La informació que es mostra no es troba completament actualitzada perquè les fonts[28] tampoc ho estan)

(Poden consultar-se els arxius de configuració en http://noc.wikimedia.org/conf/)

6.4.1 Evolució global de la seua estructura

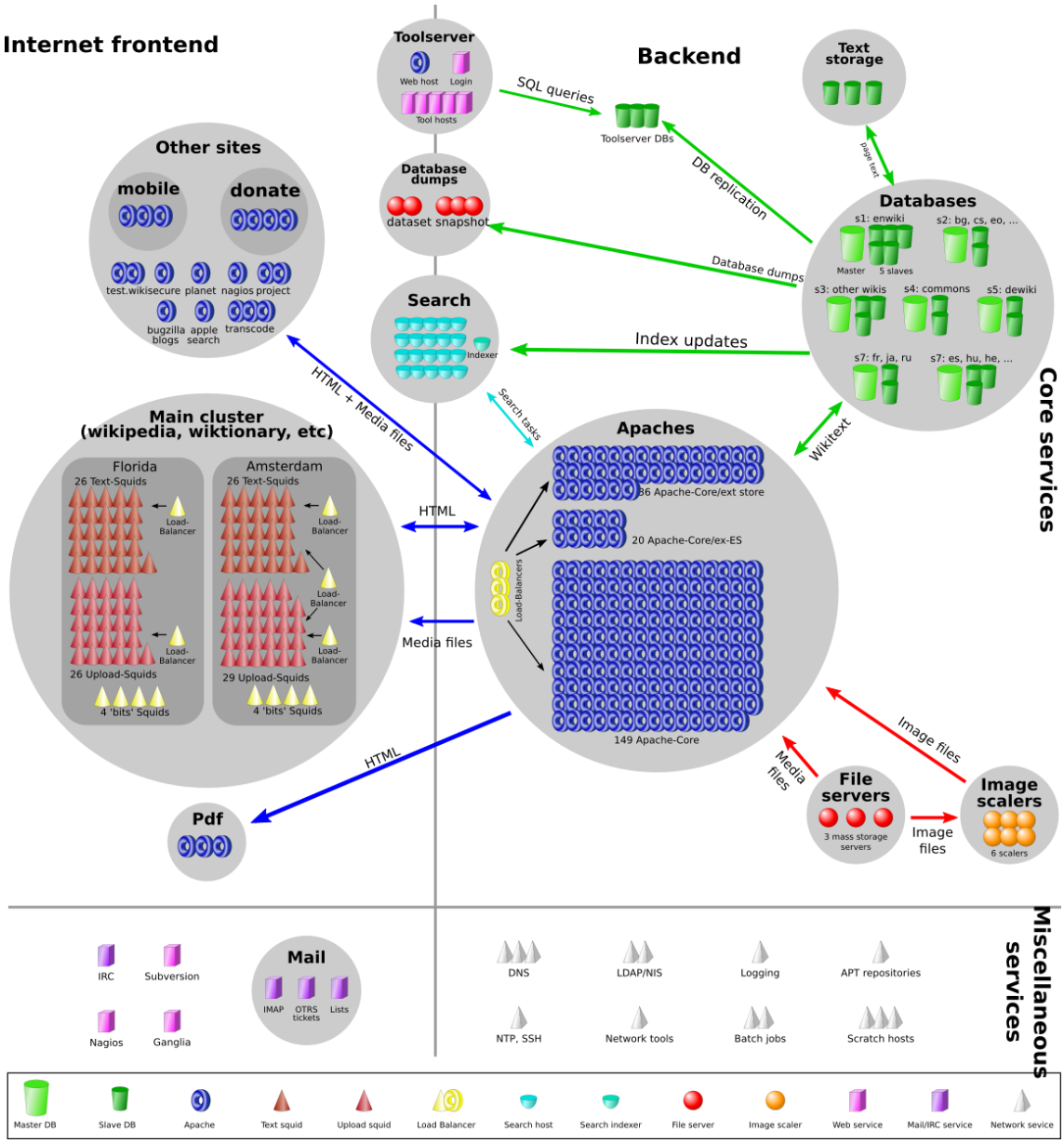
Començant amb una infraestructura mínima, amb un únic equip en el qual s'allotja i serveix tot el material, la Vikipèdia evoluciona en equipament, organització i addició de serveis. A manera d'il·lustració es mostra un parell d'instantànies preses en 2005 i 2010

- **2005**. Destacable: múltiples servidors APACHE, encara amb nom propi, dos serveis de proxy invers, distribució de la BD



- **2010**. Destacable: hi ha tants servidors APACHE que els seus noms passen a ser nombres, especialització de sistemes semiautònoms (serveis de proxy invers, BBDD i indexació), creixement d'altres sistemes auxiliars, comencen a cobrar rellevància altres projectes de la mateixa fundació.

[28] https://meta.wikimedia.org/wiki/wikimedia_servers

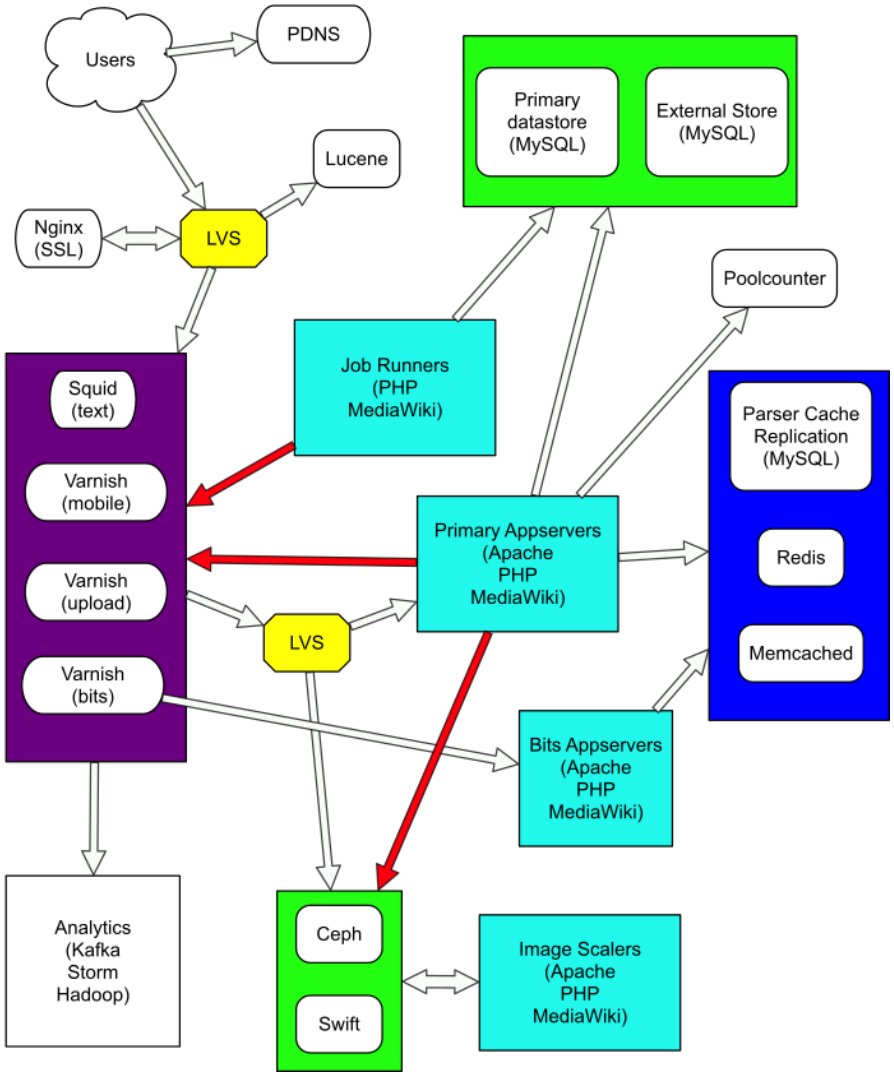


6.4.2 Programari

A l'octubre de 2012, tots els components formen una pila LAMP molt complexa:

- Vikipèdia usa *PowerDNS* per a distribuir geogràficament les sol·licituds entre els dos llocs principals (EUA i Europa), depenent de la ubicació del client.
- Amb Linux Virtual Server (*LVS*) s'equilibra la càrrega de peticions entrants. LVS també s'empra per a repartir la càrrega interna de sol·licituds MediaWiki i Lucene. El monitoratge intern es realitza mitjançant un sistema propi (*PyBal*).
- Per a agilitar el treball HTML s'usen servidors proxy inversos *Squid* i *Varnish* davant de servidors *APACHE*. Per a imatges s'empren els mateixos proxies davant de servidors *Sun Java System Web Server*.
- Tots els servidors executen *Ubuntu*, excepte els de emmagatzematge d'imatges, que executen *Solaris*.
- Els objectes distribuïts s'emmagatzemen mitjançant *Ceph* i *Swift*.
- La principal aplicació web és *MediaWiki*, escrita en PHP.

- Les dades estructurades s'emmagatzemen en *MariaDB* (compatible amb MySQL). Les wikis s'agrupen en clusters, i cada cluster és atès per diversos servidors MariaDB, replicats en una configuració de mestre únic.
- Per a caching intern de queries a la BD i resultats de processament s'empra *memcached*.
- Per a cerques de text en continguts s'empra *ElasticSearch*[29]



La nostra concepció d'un sistema distribuït s'assembla molt més a aquest últim esquema que als relacionats amb l'organització física del sistema, amb equips i connexions de xarxa. Els components d'un sistema distribuït s'assemblen més a processos que a equips.

6.4.3 Presència en Internet

A l'octubre de 2014, els recursos de la Vikipèdia es distribueixen entre les següents ubicacions:

- **eqiad** (Equinix en Ashburn, Virgínia). És la instal·lació principal, i inclou la part interna de la web i les BBDD.

[29] Abans s'usava Lucene amb moltes particularitzacions

- **esams** (EvoSwitch a Amsterdam, Holanda), actuant com cluster de *cache* Squid i altres coses menors.
- **knams** (Kennisnet a Amsterdam, Holanda), donant suport de xarxa per a l'anterior.
- **ulsfo** (United Layer en Sant Francisco), actuant com *cache* per a l'oest d'EUA i llunyà Orient. El pla estratègic de Wikimedia inclou la col·locació de més llocs com *cache* del tràfic d'Amèrica Llatina, Àsia i Orient pròxim.
- **codfw** (CyrusOne en Carrollton, Texas). Suport d'emergència.

7 Conclusions

En l'actualitat els sistemes d'informació són sistemes en xarxa. Tots els casos que hem vist anteriorment formen part dels sistemes de programari coneguts com a *Sistemes Distribuïts*.

Un disseny i desenvolupament adequats requereixen un profund coneixement de la programació concurrent i de les arquitectures utilitzades.

- En un sistema distribuït es disposa de dos models arquitectònics d'interacció: Client/servidor i P2P

En els casos vistos anteriorment es mostren alguns aspectes importants que guien els dissenys dels sistemes distribuïts.

1. Flexibilitat en la funcionalitat. Un sistema es crea amb uns requeriments en un moment donat, però aquests requeriments varien amb el temps.
2. Robustesa. El sistema ha de ser capaç de suportar fallades, mantenint la integritat de la informació, continuant amb la seua funció.
3. Rendiment. El sistema ha de tenir unes característiques de rendiment predictibles i adequades a la seua funció. Normalment el rendiment es mesura en funció de mesures de throughput (nombre de peticions servides per unitat de temps) i latència (tardança a completar una petició).
4. Escalabilitat. El disseny del sistema ha de permetre adaptar-ho a condicions canviants de càrrega sobre el mateix, sense haver de realitzar un redisseny, o alterar la seua implementació, mentre manté les característiques de rendiment esperades. Per tant es tracta d'adaptabilitat dinàmica.
5. Simplicitat. Simplicitat en la construcció, gestió de desplegament i la configuració del mateix. Això inclou simplicitat en la reconfiguració.
6. Elasticitat. Capacitat d'escalar un servei, sense que els seus usuaris noten interrupció o degradació alguna en el mateix, adaptant el conjunt de recursos utilitzats a la càrrega existent a cada moment. L'objectiu de l'elasticitat és l'obtenció d'escalabilitat minimitzant el consum de recursos i, per tant, el seu cost.

Hem considerat la computació en el núvol com a última etapa important en l'evolució de la computació:

- Caracteritzada per l'eficiència en l'ús dels recursos.
- Amb un model d'accés de “pagament per ús”.
- L’elasticitat (que inclou escalabilitat) és l'objectiu principal.

Una propietat implícita que no és aparent en els casos d'ús presentats és la relacionada amb l'ús apropiat d'un sistema: la *seguretat*. En tots els casos presentats, la seguretat en alguns dels seus aspectes és de primordial importància, condicionant en molts casos les tecnologies triades per a implantar la funcionalitat de l'aplicació.

Les decisions d'arquitectura d'un sistema incidiran en les propietats de seguretat que es poden aconseguir en el sistema resultant.

Hem pres la Vikipèdia com a servei d'escala mundial a través del que identificar els problemes i les seues solucions. Com les solucions no són perfectes, provoquen al seu torn nous problemes molt específics[30] de l'àmbit que ens ocupa.

Dins del cas d'estudi se citen i descriuen succintament múltiples exemples d'altres serveis distribuïts actuals, utilitzats per milions d'usuaris. Les seues arquitectures exigeixen un disseny molt intel·ligent i acurat per a fer front a condicions i magnituds excepcionals.

Com a sistema en evolució, les solucions que es desenvolupen van canviant al llarg del temps, de la mateixa manera que ho fan les tecnologies i recursos dels quals es disposa. Cap sistema d'aquesta escala pot romandre inalterable en el temps perquè l'adaptació és un requisit imprescindible.

Les xifres manejades en el cas d'estudi pretenen acostar-nos la realitat sobre la magnitud dels problemes. Disposar de recursos il·limitats, a més d'irreal, no és per si mateix una solució: es necessita aplicar intel·ligència perquè la suma de recursos provoque una suma de capacitats i propietats en el sistema final.

En general, per a prestar els seus serveis s'han de repartir les sol·licituds entre tots els nodes possibles, de manera que cadascun es responsabilitze d'una fracció d'aquestes peticions. D'altra banda s'ha estudiat que la replicació de nodes possibilita la continuïtat del servei, i que existeixen tècniques específiques (*caching*, *proxies* inversos) per a millorar la capacitat de servei.

S'haurà observat la necessitat de disseccionar el funcionament intern de la Vikipèdia per a trobar formes de millorar les seues característiques com a sistema distribuït. La identificació de components és una part crucial en el disseny dels sistemes estudiats en TSR.

TSR és un curs d'arquitectura de sistemes distribuïts. El seu objecte central és l'anàlisi de les alternatives de disseny en l'especificació d'aquest tipus de sistemes, considerant les tecnologies disponibles, i analitzant el seu ús sota el prisma dels aspectes citats anteriorment.

[30] I, amb sort, de menor envergadura.