GRUP:		Nota	C	I	N
NOM:		COGNOMS:			
SIGNATURA:		DNI:			

Aquest examen consta de 40 preguntes de tipus test. En cada qüestió ha de seleccionar una única alternativa d'entre totes les propostes. Escriga una "X" en la casella que trie. El valor de cada qüestió és 0.25 punts en cas de resposta correcta i -0.05 en cas d'error (equivalent a 1/5 del valor correcte).

Si dubta, escriga un asterisc "*" o un número "1", "2"... al final del text de l'apartat corresponent i utilitze l'espai en els marges de la fulla per a explicar la seua resposta, que haurà de ser forçosament breu.

L'examen pot ser completat en hora i mitja, però es disposa de 2 hores per a finalitzar.

1. Els sistemes distribuïts

	En cas de ser escalables, no podran ser sistemes concurrents. Tot sistema distribuït és un sistema concurrent. En un sistema concurrent hi haurà múltiples activitats que col·laboraran entre si. En el cas particular del sistema distribuït, les activitats podran estar situades en múltiples ordinadors. Un sistema "no concurrent" estaria format per una sola activitat i mai podria ser "distribuït".
	Utilitzen només missatges com a mecanisme d'intercomunicació. Les activitats d'un sistema distribuït necessitaran utilitzar missatges per a intercanviar informació quan residisquen en ordinadors diferents. No obstant això, res prohibeix que diverses activitats del sistema residisquen en un mateix node i utilitzen memòria compartida per a intercomunicar-se (un fitxer, per exemple, en cas de tractar-se de diversos processos; una variable global, en cas de tractar-se de diversos fils d'execució...).
	Poden estar formats per un únic procés. Un sistema distribuït hauria d'estar format per múltiples activitats que s'executen en múltiples ordinadors, que col·laboren entre si, i que oferisquen una imatge de sistema únic. Aquests tres requisits no poden ser satisfets simultàniament per un únic procés.
	Actualment només segueixen el model de computació en el núvol. Qualsevol aplicació concurrent que utilitze la xarxa serà un exemple de sistema distribuït. No s'exigeix que es despleguen en un entorn elàstic basat en virtualització, com és el cas dels sistemes de computació en el núvol.
	Totes les anteriors.
X	Cap de les anteriors.

2. Els rols desenvolupador, proveïdor de serveis, administrador de sistema i usuari

X	Estan clarament diferenciats en un sistema SaaS, però no sempre estan exercits per diferents agents.
	En els ordinadors personals estan exercits conjuntament pel propietari de l'ordinador. Normalment el propietari d'un ordinador no és un programador avançat capaç de desenvolupar totes les aplicacions que necessita. Per tant, el rol "desenvolupador" no està exercit per l'usuari en la majoria dels casos.
	En els "*mainframes*" l'usuari també actuava a voltes com a administrador del sistema. Els rols d'usuari i administrador del sistema estaven clarament diferenciats en aquells sistemes. Les tasques d'administració eren delicades i no podien ser realitzades pels usuaris.
	En els centres de còmput empresarials resulta trivial i barat gestionar els rols de proveïdor de serveis i administrador de sistema. Ni resulta trivial ni barat. Cal recórrer a personal especialitzat per a desenvolupar aquestes tasques. Això suposa un cost econòmic alt per a l'empresa.
	Totes les anteriors.
	Cap de les anteriors.

3. Exemples de serveis SaaS

	Linux Linux és un exemple de sistema operatiu. No segueix el model SaaS de computació en el núvol. El model SaaS consisteix a oferir una aplicació concreta com a servei elàstic. Un sistema operatiu no pot considerar-se una aplicació.
X	Google Drive Google Drive és el servei oferit per Google per a: facilitar cert espai d'emmagatzematge de fitxers a través de la xarxa i accedir a les aplicacions de Google Docs (gestió de presentacions, fulls de càlcul, processament de textos...). Sí que compleix amb el que s'espera d'un model SaaS: són aplicacions oferides com a servei a través de la xarxa.
	ZeroMQ ZeroMQ és un middleware de comunicacions. S'utilitzarà al costat d'altres eines a l'hora de desenvolupar aplicacions distribuïdes escalables, però no és una aplicació. Per tant, no respecta el model de servei SaaS.
	Microsoft Word Encara identifiquem a Microsoft Word amb una aplicació que s'instal·la en un sistema Windows i que s'executa de manera local. En aquest supòsit, no respecta el model de servei SaaS.
	Totes les anteriors.
	Cap de les anteriors.

4. Sobre els models de servei en la computació en el núvol:

X	En el model IaaS el proveïdor ofereix xarxes i màquines virtuals perquè el "usuari" desplegue allí les seues aplicacions distribuïdes.
	En el model SaaS el proveïdor ofereix xarxes i màquines virtuals perquè el "usuari" desplegue allí les seues aplicacions distribuïdes. Un proveïdor en el model SaaS ha d'oferir una aplicació com a servei.
	En el model PaaS el proveïdor ofereix aplicacions distribuïdes, garantint la seua escalabilitat i mantenibilitat perquè l'usuari les utilitze directament. Això és el que ocorre en el model SaaS.
	En el model IaaS es necessita un proveïdor PaaS subjacent perquè els serveis IaaS puguen facilitar-se als usuaris. Just al contrari. Un model PaaS sol utilitzar una infraestructura escalable (que pot ser facilitada per un IaaS). Per tant, PaaS necessita com a nivell inferior a IaaS.
	Totes les anteriors.
	Cap de les anteriors.

5. S'han vist diferents models o nivells de serveis en el núvol...:

X	N'hi ha tres: IaaS, PaaS i SaaS. Infraestructura com a servei (IaaS), plataforma com a servei (PaaS) i *programari* com a servei (SaaS).
	N'hi ha cinc: Fiables, Disponibles, Segures ("Safe"), Mantenibles i Segures ("Secure"). Aquests són els atributs d'un sistema robust. No són models de serveis.
	N'hi ha cinc: Seqüencials, Causals, Processadors, FIFOs, i "*Caches*". Aquests són exemples de models de consistència. No són models de serveis.
	N'hi ha dos: Actius i Passius. Són exemples de models de replicació. No són models de serveis.
	Totes les anteriors.
	Cap de les anteriors.

6. Mecanismes necessaris en els sistemes de computació en el núvol:

X	Virtualització. La virtualització és un mecanisme mitjançant el qual es facilita l'administració dels recursos de maquinari en un sistema de computació en el núvol. És la base per a proveir una infraestructura elàstica.
	Gestió seqüencial (sense concurrència). Qualsevol sistema distribuït és concurrent. Almenys hi haurà una activitat per cada agent o component que participe en el sistema.
	Escalabilitat. L'escalabilitat és un objectiu de la computació distribuïda (i, per tant, de la computació en el núvol). No és un mecanisme.
	Consistència forta. La consistència es considera una propietat d'un sistema. No és un mecanisme. En qualsevol cas, la gestió de la consistència necessita algun protocol de replicació que controle els accessos de lectura i escriptura. En cas de requerir-se una consistència forta, aquest protocol exigiria una sincronització pesada i el sistema no seria escalable.
	Totes les anteriors.
	Cap de les anteriors.

7. El desplegament d'una aplicació distribuïda...:

X	Plantejarà alguns problemes pràctics: actualització de components, mantenibilitat, disponibilitat...
	Està automatitzat en els sistemes IaaS actuals. No és així. És un dels objectius dels sistemes PaaS. No està resolt amb procediments automatitzats en els sistemes IaaS.
	Està gestionat per l'usuari en els sistemes SaaS actuals. No ho gestiona l'usuari en aquests sistemes, sinó el proveïdor.
	És un problema ja resolt des dels sistemes basats en "*mainframes*". No. Encara és un problema obert en certs entorns i per a certes tasques incloses en aquest desplegament. No està resolt de manera general en els sistemes distribuïts. L'afirmació del primer apartat esmenta els aspectes a considerar per a "resoldre-ho".
	Totes les anteriors.
	Cap de les anteriors.

8. Un model teòric senzill de sistema distribuït sol considerar que...:

X	Els processos són agents seqüencials.
	No hi ha cap tipus de fallada. S'assumia que els processos podien fallar, parant.
	Els processos no necessiten comunicar-se entre si. Si els processos no es comunicaren, no hi hauria un sistema distribuït. No podrien col·laborar. Parlaríem d'un conjunt de processos no relacionats.
	Els esdeveniments rellevants són externs en tots els casos i provocats pel transcurs del temps. Es distingeix entre esdeveniments d'entrada, d'eixida i interns. Els de entrada i eixida formen el superconjunt d'esdeveniments externs. És a dir, no tots són externs. Els esdeveniments externs estan associats a la comunicació entre agents (no es generen només perquè transcorre el temps).
	Totes les anteriors.
	Cap de les anteriors.

9. Respecte a la sincronia en un model teòric simple de sistema distribuït...:

	Hi haurà processos sincrònics si tots ells són capaços de gestionar exactament un esdeveniment en cada pas de l'algorisme.
	Hi haurà canals sincrònics quan es puga limitar el temps de propagació i lliurament de cada missatge.
	Hi haurà comunicació sincrònica quan l'emissor es bloquege mentre no reba una resposta del receptor.
	Tindríem rellotges sincrònics si el rellotge de cada node estiguera sincronitzat amb un “temps real global”.
X	Totes les anteriors. Aquestes van ser les definicions utilitzades en el model teòric simple per a cadascun d'aquests conceptes.
	Cap de les anteriors.

10. En els sistemes distribuïts escalables, en comparar servidors multi-fil amb servidors asincrònics...:

	Els asincrònics plantegen el problema de suspendre's amb cada petició rebuda. Al contrari: minimitzen les necessitats de bloqueig o suspensió.
X	Els asincrònics solen estar orientats a esdeveniments i es corresponen millor amb un model teòric senzill de sistema distribuït.
	Els multi-fil sempre ofereixen millor rendiment en poder realitzar múltiples accions simultàniament. Sí que permeten realitzar múltiples accions simultàniament, però quan aquestes accions impliquen accedir a algun recurs compartit caldrà utilitzar mecanismes de control de concurrència que introduiran bloquejos i penalitzaran el rendiment.
	Els multi-fil ofereixen una gestió d'estat més senzilla ja que no cal preocupar-se per les “guardes” ni per les “accions” associades als “esdeveniments”. És cert que no hi haurà “guardes” ni “accions” associades a elles, però la gestió d'estat és més complexa, malgrat això. En els multi-fil caldrà identificar apropiadament les seccions crítiques i protegir-les amb els mecanismes de sincronització adequats. Així, la gestió resulta ser més complexa, provocant amb freqüència el bloqueig de les activitats.
	Totes les anteriors.
	Cap de les anteriors.

11. Els servidors multi-fil...:

	Proporcionen un model de programació asincrònic. Al contrari, el model de programació asincrònic està proporcionat pels servidors asincrònics. Els servidors multi-fil necessiten mecanismes de sincronització per a no generar inconsistències en modificar recursos compartits.
	No requereixen eines de sincronització en accedir a recursos compartits. Ja explicat en l'apartat anterior. Sí que necessiten aquest tipus d'eines.
	Minimitzen l'ús de missatges entre agents, incrementant l'escalabilitat dels serveis que implanten. És possible que no requerisquen tants missatges per a intercomunicar els diferents agents, però això no garanteix una major escalabilitat. En haver d'utilitzar eines de sincronització i accedir als recursos compartits en exclusió mútua, s'introduiran bloquejos perllongats quan calga suportar altes càrregues de treball. Per tant, l'escalabilitat no millora.
	Poden augmentar l'escalabilitat del servei que implanten si en la majoria de les operacions s'utilitzen recursos compartits. Podria millorar l'escalabilitat SI NO S'UTILITZAREN recursos compartits. Si cada fil tinguera el seu conjunt propi de recursos i mai arribara a bloquejar-se, podria incrementar-se el rendiment i l'escalabilitat milloraria. Per desgràcia, els fils d'un mateix servidor solen compartir recursos.
	Totes les anteriors.
X	Cap de les anteriors.

12. Els servidors asincrònics...:

	Utilitzen un model de programació dirigit per esdeveniments. Cert. En un servidor asincrònic es van atenent les diferents peticions rebudes a mesura que es complete el servei de l'anterior. Cada petició es gestiona com un esdeveniment.
	Poden implantar-se utilitzant NodeJS i ZeroMQ. NodeJS permet implantar servidors asincrònics en JavaScript. ZeroMQ és un middleware de comunicacions basat en intercanvi asincrònic de missatges que pot utilitzar-se en NodeJS.
	Estructuren el seu codi utilitzant "*callbacks*", que són les accions associades a determinats esdeveniments. Pot estructurar-se d'aquesta manera.
	Eviten els problemes que plantegen les seccions crítiques. Cert. Normalment hi haurà un sol fil d'execució en cada procés servidor. Així s'eviten les seccions crítiques ja que no hi ha recursos compartits l'accés concurrent dels quals haja de protegir-se.
X	Totes les anteriors.
	Cap de les anteriors.

13. Un middleware...:

	Generalment impedeix que els components siguen interoperables. Un dels objectius tradicionals dels middleware és facilitar la interoperabilitat entre múltiples components, independentment de qui els desenvolupe.
	Quan gestione la intercomunicació estarà per sota del nivell de transport. En un sistema distribuït la majoria dels middleware utilitzats solen situar-se sobre el sistema operatiu. El nivell de transport està facilitat pel propi sistema operatiu; per tant, el middleware se situa sobre el nivell de transport.
X	Sol respectar un o més estàndards i facilita el desenvolupament d'aplicacions distribuïdes. La majoria dels middleware tenen aquests objectius.
	Obliga a utilitzar una interacció client/servidor entre components. No és obligatori. Es poden tenir middleware que faciliten un patró de comunicació diferent al de "petició i resposta" assumit en les interacciones client/servidor. Per exemple, "publish/subscribe". També es podria tenir un middleware per a proporcionar una imatge de memòria compartida, permetent que els processos de diferents ordinadors es comunicaren accedint a objectes compartits. Potser no fóra escalable, però sí suficient per a certs tipus d'aplicació.
	Totes les anteriors.
	Cap de les anteriors.

14. Exemples de tipus de middleware i funcions que desenvolupen:

	URL. Permet localitzar els recursos en un sistema distribuït. Una URL és un tipus de nom, no és un middleware.
	JavaScript. Permet desenvolupar aplicacions distribuïdes. JavaScript és un llenguatge de programació. No és un middleware.
X	RPC. Proporciona un model d'interacció client/servidor amb transparència d'ubicació. Pot considerar-se un tipus de middleware entre els objectius del qual està el proporcionar transparència d'ubicació.
	SaaS. Gestiona la virtualització de recursos i proporciona eines per al desenvolupament de serveis web escalables. SaaS és un model de serveis en la computació en el núvol. No és un tipus de middleware. A més, la definició que apareix en aquest apartat no correspon a un model SaaS sinó a un PaaS.
	Totes les anteriors.
	Cap de les anteriors.

15. Les crides a procediment remot:

	Es comporten exactament igual a les crides a procediment local El terme "comportament" té dues dimensions: la interfície oferida a qui haja d'usar el mecanisme i les operacions dutes a terme per a complir amb la seua funcionalitat. Una RPC pot oferir una interfície similar a la d'una crida a procediment local, però la seqüència de passos necessaris és diferent. Per tant, no es comporten d'igual manera.
	Utilitzen característiques del processador per a passar arguments al servidor. Això ocorre en les crides locals. En una RPC s'utilitzaran missatges per a fer arribar els arguments al servidor.
X	Presenten modes de fallades diferents que les crides locals. Una RPC pot fallar quan hi haja problemes en la transmissió dels missatges o quan falle l'ordinador on es trobe el procediment servidor. En una crida local no es produiran mai aquestes situacions de fallada.
	Precisen d'un mecanisme de broadcast en el transport. Podria ser necessari si s'invocara un procediment d'un servidor replicat. No obstant això, el mecanisme RPC descrit en aquesta assignatura no assumia aquesta situació i, de manera general, no es necessita una difusió per a implantar la RPC.
	Totes les anteriors.
	Cap de les anteriors.

16. Els estàndards en computació distribuïda...:

	Proporcionen una solució racional per a resoldre certs problemes.
	Faciliten la interoperabilitat.
	Proporcionen funcionalitat d'alt nivell.
	Familiaritzen als programadors amb les tècniques a utilitzar.
X	Totes les anteriors. Les quatre propietats citades es llisten com a atribuïbles a qualsevol estàndard en computació distribuïda en el material de teoria de TSR.
	Cap de les anteriors.

17. Una biblioteca d'invocació de mètodes remots...:

	És un exemple de middleware. Cert. Permet la intercomunicació de components en un sistema distribuït que utilitze un model orientat a objectes.
	S'utilitza en Java RMI. Java RMI és un exemple de middleware d'aquest tipus.
	Utilitza el protocol SOAP en els serveis web. És un dels exemples citats en el material de teoria.
	S'utilitza en l'estil arquitectònic REST, prenent com a base els protocols HTTP. També se cita així en el material de teoria.
X	Totes les anteriors.
	Cap de les anteriors.

18. Els sistemes d'objectes distribuïts...:

	Solen ser escalables ja que es poden crear nous objectes en l'aplicació quan es requerisca, sense cap limitació. Encara que sí que existeix aquesta llibertat per a crear nous objectes quan es considere necessari, els sistemes basats en objectes distribuïts difícilment seran escalables. El codi que caldrà executar en cada component de l'aplicació realitzarà freqüents crides a objectes remots i això ralentirà l'execució.
	Ofereixen un model de programació asincrònic i, per tant, fàcilment escalable. Que el sistema estiga orientat a objectes no condiciona que s'adopte un model de programació asincrònic per als servidors. Pot ser multi-fil o asincrònic, o qualsevol altre (si n'hi haguera més). Són característiques independents.
	Faciliten dissenys amb alta cohesió, generant components fàcilment reutilitzables i amb baix acoblament. Sí que es proporciona, generalment, alta cohesió. No obstant això, l'acoblament també sol ser fort, ja que des de qualsevol objecte es tindrà accés a altres objectes i s'utilitzaran els seus mètodes per a modificar el seu estat. Amb això s'ofereix una imatge similar a la d'una "memòria compartida". Com això sol estar acompanyat per transparència d'ubicació... al final el sistema difícilment podrà escalar, ja que s'estaran realitzant contínuament invocacions a objectes remots.
	Minimitzen la contenció i la generació de seccions crítiques. Això també afavoreix la seua escalabilitat. Cada objecte, com podrà ser invocat des de múltiples processos, serà gestionat com un recurs compartit. Per això, si s'admet concurrència dins del procés que ho mantinga, els mètodes que modifiquen els seus atributs es convertiran en una secció crítica que caldrà protegir de manera adequada. Això redueix la seua escalabilitat.
	Totes les anteriors.
X	Cap de les anteriors.

19. Els middleware de missatgeria...:

	Proporcionen en alguns casos un model d'interacció asincrònic, altament escalable. Diversos patrons d'interacció implantats en ZeroMQ proporcionen exemples de gestió asincrònica (pub/sub, dealer, router...). L'asincronia és clau per a millorar l'escalabilitat.
	Permeten que el programador desenvolupe les seues aplicacions sense recórrer a la compartició de recursos. Això afavoreix l'escalabilitat. L'ús de memòria compartida entre les activitats d'una aplicació distribuïda comporta l'aparició de seccions crítiques i la necessitat de sincronització per a protegir-les. La intercomunicació basada en missatges permet evitar aquestes situacions.
	Pots usar-se per a la implantació de servidors replicats. Les rèpliques d'un determinat servidor necessitaran comunicar-se intercanviant missatges per a arribar a respectar algun model de consistència. Un middleware de "missatgeria" permet implantar aquests components.
	Inclouen a ZeroMQ com a exemple de middleware persistent i sense gestor ("*brokerless*"). ZeroMQ manté els missatges en memòria principal mentre no siga possible lliurar-los al seu destinatari. Amb això es garanteix cert grau de persistència en la comunicació. A més, no necessita cap procés gestor extern per a fer la intercomunicació. N'hi ha prou que els agents a intercomunicar usen la biblioteca ZeroMQ.
X	Totes les anteriors.
	Cap de les anteriors.

20. Un middleware o sistema de missatgeria es considera persistent quan...:

	No bloqueja els emissors de missatges. D'aquesta manera millora l'escalabilitat de les aplicacions. Aquesta és la definició de comunicació asincrònica. Es demanava comunicació persistent.
	Utilitza un servei de noms per a trobar els processos servidors als quals ha de lliurar els missatges. Així s'aconsegueix transparència de replicació. El servei de noms no té cap rellevància en considerar si un sistema de comunicació és persistent o no.
X	Els missatges es mantenen temporalment en buffers gestionats pel "canal". El receptor no cal que estiga actiu quan l'emissor envia el missatge. Aquesta és la definició i la propietat principal de la comunicació persistent.
	Es recolza en els nivells de xarxa i transport per a fer l'encaminament i la gestió del canal de comunicació. Els serveis d'encaminament i la gestió de connexions tampoc tenen importància a l'hora de considerar la persistència.
	Totes les anteriors.
	Cap de les anteriors.

21. Exemples de middleware:

	Java RMI. Middleware d'invocació de mètodes remots propi de Java.
	ZeroMQ. Middleware de comunicacions.
	RabbitMQ Middleware de comunicacions.
	JINI. Middleware orientat a objectes, inicialment desenvolupat per Sun.
X	Totes les anteriors.
	Cap de les anteriors.

22. L'escalabilitat horitzontal...:

	Consisteix en el reemplaçament d'un component per un altre amb major capacitat de servei. Aquesta és la definició d'escalabilitat vertical.
	Implica que el sistema siga escalable i adaptable. L'adaptabilitat serà dinàmica (reaccionant a variacions de càrrega) i autònoma (sense intervenció humana). Aquesta és la definició d'elasticitat.
	Garanteix la robustesa del sistema. Pot millorar el rendiment, però no influeix gens en la seguretat, per exemple. Per tant, no garanteix la robustesa. Per a fer-ho, s'hauria de garantir alhora disponibilitat, fiabilitat, mantenibilitat i seguretat.
	Requereix un subsistema de monitoratge (de la càrrega suportada i el rendiment obtingut) i un altre d'actuació que automatitze la reconfiguració. Aquests subsistemes resulten necessaris per a construir un sistema adaptable dinàmicament (elàstic, en cas que també siga escalable).
	Totes les anteriors.
X	Cap de les anteriors.

23. Mecanismes per a incrementar l'escalabilitat de grandària d'un servei:

	Reduir o eliminar les necessitats de sincronització entre els agents que implanten el servei. Si no es necessita sincronitzar, no hi haurà bloquejos. Millora l'escalabilitat.
	Utilitzar algorismes descentralitzats. Els algorismes descentralitzats permeten prendre decisions a partir de la informació local i suporten les fallades de qualsevol component. Per tant, també redueixen la necessitat de sincronització. Millora l'escalabilitat.
	Delegar tot el còmput possible als agents clients. Això reduiria la càrrega en els servidors i permetria atendre a un major nombre de clients. Millora l'escalabilitat.
	Replicar els components servidors, utilitzant el model de consistència més relaxat que admeta el servei. En tenir els serveis replicats es podrà atendre a un nombre més alt de clients. Una consistència relaxada redueix les necessitats de sincronització entre rèpliques. Millora l'escalabilitat.
X	Totes les anteriors.
	Cap de les anteriors.

24. Poden ser causes de contenció...:

	Adoptar un model de programació asincrònica. Redueix o elimina la necessitat de sincronització. No causarà contenció.
	Utilitzar algorismes descentralitzats. Permet prendre decisions locals. No necessita sincronització. No causarà contenció.
	Emprar un model de replicació passiva. És més escalable que el model de replicació activa. D'entre els models de replicació, aquest és el més recomanable. Permet realitzar més fàcilment un equilibrat de la càrrega. Redueix la contenció en cas de sistemes replicats.
X	Usar eines de sincronització. Introduiran bloquejos. Són causes de contenció.
	Totes les anteriors.
	Cap de les anteriors.

25. L'elasticitat...:

	És una propietat de qualsevol classe d'aplicació (tant distribuïda com no distribuïda). L'elasticitat implica escalabilitat i adaptabilitat a la càrrega. La combinació d'ambdues propietats només té sentit en un sistema distribuït.
X	Requereix escalabilitat i adaptabilitat. Així va ser definida.
	Només té sentit en els sistemes IaaS. Es pot exigir a qualsevol tipus de sistema o aplicació distribuïts.
	S'obté en utilitzar un model de replicació activa. El model de replicació activa ofereix una escalabilitat molt limitada. Per això, no és la millor base per a desenvolupar un sistema elàstic.
	Totes les anteriors.
	Cap de les anteriors.

26. S'han vist tres dimensions d'escalabilitat...:

	Rendiment, Persistència i Disponibilitat. La persistència no té res a veure amb l'escalabilitat. Encara que una alta disponibilitat i un bon rendiment siguen aconsellables en un sistema distribuït escalable, no són les dimensions que determinen si un sistema distribuït és escalable.
	Consistència, Disponibilitat i Tolerància al particionat de la xarxa. Aquestes són les tres propietats analitzades en el teorema CAP. De nou, no són les dimensions d'escalabilitat.
X	Grandària, Distància i Administrativa. Aquests sí són els eixos o dimensions que determinen si un sistema distribuït pot considerar-se o no escalable.
	Vertical, Horitzontal i Obliqua. Una altra alternativa per a classificar els diferents tipus d'escalabilitat és distingir entre escalabilitat vertical (la que s'obté reemplaçant components) i l'horitzontal (obtinguda afegint nous nodes al sistema). No obstant això, en aquesta segona classificació no apareix cap "escalabilitat obliqua".
	Totes les anteriors.
	Cap de les anteriors.

27. L'escalabilitat vertical...:

X	Incrementa la capacitat de servei reemplaçant components. Aquesta és la definició enunciada en les classes de teoria.
	Millora la disponibilitat dels serveis. L'escalabilitat vertical permet millorar el rendiment d'una sola màquina. Com no s'afigen altres nodes al sistema, la disponibilitat dels serveis executats no s'incrementa. Si falla el node on s'execute aquest servei, el servei queda indisponible.
	Incrementa la capacitat de servei afegint nous nodes, recursos o components. Això ocorre en l'escalabilitat horitzontal, no en la vertical.
	Millora la seguretat dels serveis. Per a millorar la seguretat en un sistema distribuït cal emprar altres mecanismes. L'escalabilitat vertical no influeix per a res en la seguretat.
	Totes les anteriors.
	Cap de les anteriors.

28. Un sistema escalable...:

	Haurà d'eliminar els seus punts de contenció. Si hi haguera punts de contenció que introduïren "colls de botella" el sistema deixaria d'incrementar el seu rendiment i capacitat de servei en augmentar la càrrega que estiga suportant. Si això succeïra, no seria escalable.
	Haurà de ser robust. De res serviria tenir un sistema capaç d'incrementar la seua capacitat de servei en funció de la càrrega introduïda si no fóra fiable, es garantira la seua disponibilitat, es recuperara el més prompte possible de les situacions de fallada (mantenibilitat) i fora segur.
	Es podrà augmentar la seua capacitat de servei quan siga necessari. Aquesta és la seua característica principal.
	No sempre tindrà una consistència forta. Quants més nodes participen en el sistema distribuït (i el nombre serà cada vegada major si s'empra escalabilitat horitzontal i la càrrega que suporta el sistema segueix creixent) més difícil serà mantenir una consistència forta entre les rèpliques de cada element gestionat pel sistema.
X	Totes les anteriors.
	Cap de les anteriors.

29. El "*sharding*" o particionat d'una base de dades distribuïda (utilitzat, per exemple, en MongoDB)...:

	És una implantació del mecanisme de repartiment de dades que potencia l'escalabilitat de grandària.
	És una implantació del mecanisme de repartiment de tasques que potencia l'escalabilitat horitzontal.
	Aconsegueix augmentar el grau de concurrència sense necessitar mecanismes de sincronització.
	No sempre exigeix mantenir múltiples rèpliques per a un mateix element de la base de dades.
X	Totes les anteriors. En el "*sharding*" es reparteixen les dades mantingudes en la BD entre múltiples ordinadors (primer apartat), amb el que les tasques de gestió d'aquestes dades també estaran repartides entre múltiples processos (segon apartat). Ambdues facetes milloren l'escalabilitat horitzontal (escalabilitat de grandària). En efectuar aquest repartiment cada procés podrà atendre, pel seu compte, una petició diferent. Així s'augmenta el grau de concurrència sense necessitar mecanismes de sincronització ja que no hi ha recursos compartits entre els diferents processos servidors (tercer apartat). Finalment, el "*sharding*" permet que hi haja un sol servidor en cada fragment de la base de dades (quart apartat), encara que generalment n'hi ha més d'un (utilitzant replicació) per a millorar la disponibilitat.
	Cap de les anteriors.

30. Amb el model actiu de replicació...:

	Es garanteix la robustesa d'un servei. La replicació (tant si és passiva com a activa) millora la disponibilitat i la mantenibilitat, però no fa res per a millorar la seguretat. Per tant, la robustesa no està garantida. La fiabilitat dependrà dels components utilitzats per a implantar el sistema o servei.
X	Es pot suportar el model de fallades d'omissió general. Per a suportar els models de fallades més severes (omissió general, arbitrari) se sol recórrer a una comparativa entre les eixides generades per les diferents rèpliques, seleccionant la resposta majoritària. El model actiu permet aquest tipus de gestió ja que múltiples rèpliques gestionen directament cada petició i responen totes al client, on el seu "stub client" realitza la comparativa abans de retornar el control al procés client.
	No existirà risc d'inconsistència quan s'executen operacions no deterministes. Al contrari, si les operacions foren no deterministes cada rèplica podria generar un resultat diferent. Si aquest resultat afecta a l'estat mantingut en cada rèplica, s'arribaria a tenir un estat diferent en cada rèplica. És a dir, es generarien inconsistències entre l'estat de les rèpliques.
	No es necessita cap tipus de sincronització entre les rèpliques per a obtenir una consistència seqüencial entre elles. Encara que les rèpliques no es comuniquen entre si una vegada han rebut cada petició, sí que es necessita sincronitzar-les per a arribar a una consistència seqüencial. Aquesta sincronització consisteix a utilitzar un protocol de difusió dels missatges que emeten els clients perquè siguen lliurats en el mateix ordre en totes les rèpliques destinatàries (difusió d'ordre total FIFO). Un protocol d'aquest tipus requereix que els diferents processos decidisquen en quin ordre han de lliurar-se els missatges "concurrents". Amb aquesta fi es necessita intercanviar un bon nombre de missatges entre tots els participants. Això demana ample de banda i temps, ralentint l'execució.
	Totes les anteriors.
	Cap de les anteriors.

31. En el model passiu de replicació...:

	Totes les rèpliques d'un servei tenen un mateix rol. No és així. Hi ha una rèplica primària i totes les altres són rèpliques secundàries. Són dos rols diferents.
	Es pot suportar el model de fallades arbitràries (o fallades "bizantines"). Com només existeix una rèplica primària que gestiona directament cada petició, no es pot "votar" per un resultat. Així no pot suportar-se un model de fallades arbitràries.
	Es garanteix la seguretat (en les seues dues accepcions: "security"/"safety") del servei oferit. La seguretat no pot garantir-se emprant replicació. Els mecanismes a utilitzar són uns altres (xifrat, ús de certificats, polítiques de seguretat uniformes i ben implantades...)
X	Es requereix menys còmput que en el model actiu quan les operacions impliquen un servei perllongat però modifiquen poc estat. Aquest és el seu avantatge principal en comparar-ho amb el model actiu. Encara que l'operació requerirà molt de temps, aquest temps solament s'inverteix en la rèplica primària. En modificar poc estat, costarà poc transferir l'estat a les rèpliques secundàries i aplicar-ho allí. En el model actiu, totes les rèpliques haurien d'executar localment l'operació i els hauria costat més a totes elles. Així, utilitzant replicació activa un sol servei podria saturar a cert conjunt de N nodes. Per contra, amb replicació passiva és possible desplegar N serveis en el mateix conjunt de nodes, situant la rèplica primària de cadascun en cada node. Un exemple d'operacions del tipus comentat en el primer paràgraf és l'execució de sentències en una base de dades relacional. Sobretot si només afecten a uns pocs registres i requereixen l'accés a disc per a recuperar-los i tornar-los a guardar. Cada sentència pot executar-se localment en la rèplica primària. Solament es transmetran les modificacions a les rèpliques secundàries quan la transacció que les englobe finalitze satisfactòriament. El missatge transmès serà xicotet i el temps necessari per a aplicar aquests canvis en les rèpliques secundàries podrà ser entre 10 i 200 vegades menor a l'execució local.
	Totes les anteriors.
	Cap de les anteriors.

32. **Seleccionar, d'entre totes les alternatives propostes, aquella que oferisca la llista més extensa de models de consistència satisfets en la següent execució. No s'admetrà cap alternativa amb algun model de consistència no respectat per l'execució:**
P1:W(x)1, P2:W(x)2, P3:R(x)1, P3:W(x)3, P2:W(x)4, P4:R(x)2, P4:R(x)3, P4:R(x)1, P5:R(x)2, P4:R(x)4, P5:R(x)3, P5:R(x)1, P5:R(x)4

	Seqüencial, "*cache*", processador, causal, FIFO. En aquesta execució s'observa que: 1) P2 escriu primer el valor 2 i després el valor 4. No hi ha cap procés més que escriga dos valors. 2) P3 llig el valor 1 abans d'escriure el valor 3. Si les execucions mantingueren l'ordre FIFO, en tots els lectors apareixeria abans el valor 2 que el 4. Si les execucions mantingueren l'ordre causal, el valor 1 hauria d'aparèixer abans que el 3. En aquesta execució els processos 4 i 5 actuen com a lectors. Tots dos obtenen la mateixa seqüència: 2, 3, 1, 4. Per tant, en compartir tots la mateixa seqüència sobre la mateixa variable (solament s'utilitza la "x" en aquest exemple), la consistència és "*cache*". També es respecta que siga FIFO. La unió de "*cache*" i FIFO genera la consistència "processador". No obstant això, el valor 3 és llegit abans que el valor 1. Per tant, no es respecta la consistència causal. Per no ser causal, tampoc podrà ser seqüencial. Resum: les consistències que es compleixen són ***"cache"*, FIFO i processador.**
	"Cache".
	Causal, FIFO.
	Processador, FIFO.
X	Processador, *"cache"*, FIFO.
	FIFO.

33. **Seleccionar, d'entre totes les alternatives propostes, aquella que oferisca la llista més extensa de models de consistència satisfets en la següent execució. No s'admetrà cap alternativa on aparega un model de consistència no respectat per l'execució: P1:W(x)1, P2:W(x)2, P3:R(x)1, P3:W(x)3, P2:W(x)4, P4:R(x)1, P4:R(x)3, P4:R(x)2, P5:R(x)2, P4:R(x)4, P5:R(x)1, P5:R(x)3, P5:R(x)4**

	FIFO. Els cinc primers accessos d'aquesta execució són idèntics als vists en la qüestió 32. No obstant això, hi ha diferències en els processos lectors. P4 llig 1, 3, 2 i 4. P5 llig 2, 1, 3 i 4. Ambdues seqüències de lectura respecten la condició FIFO (2 abans que 4) i causal (1 davant que 4) explicades en la solució de l'activitat 32. Per no coincidir les seqüències de lectura en els processos 4 i 5, l'execució no serà *"cache"*, ni podrà ser seqüencial. Si no és *"cache"*, tampoc podrà ser "processador". Per tant, la solució correcta consistia a citar les consistències **FIFO i causal**.
	Processador, FIFO, *"cache"*.
X	Causal, FIFO.
	Causal, *"cache"*.
	Processador, causal.
	Processador, causal, *"cache"*, FIFO.

34. **Seleccionar, d'entre totes les alternatives propostes, aquella que oferisca la llista més extensa de models de consistència satisfets en la següent execució. No s'admetrà cap alternativa on aparega un model de consistència no respectat per l'execució: P1:W(x)1, P2:W(x)2, P3:W(x)3, P2:W(x)4, P4:R(x)3, P4:R(x)1, P4:R(x)2, P5:R(x)3, P4:R(x)4, P5:R(x)1, P5:R(x)2, P5:R(x)4**

	"Cache". Cap dels processos escriptors arriba a llegir els valors escrits pels altres processos. Només P2 escriu dos valors, en ordre 2, 4. Per tant, les execucions seran FIFO si en elles apareix un 2 abans que un 4. En aquest cas, també seran causals. P4 i P5 actuen com a processos lectors. Obtenen tots dos la seqüència 3, 1, 2, 4. Per ser la mateixa seqüència en tots els lectors, es compleix la consistència *"cache"*. Per ser FIFO i causal, serà també seqüencial. En complir-se tant la *"cache"* com la FIFO, també es compleix la consistència "processador". Resum: es compleixen les consistències ***"cache"*, FIFO, processador, causal i seqüencial**.
	FIFO.
	Processador, FIFO, *"cache"*.
	Causal, FIFO.
X	Seqüencial, causal, processador, FIFO, *"cache"*.
	Causal, *"cache"*.

35. Relació entre defectes, errors i fallades:

	Podrà haver-hi fallades que no estiguen causats per cap defecte. Perquè hi haja una fallada ha d'haver-se donat prèviament algun error. Al seu torn, els errors no es generen si no estan precedits per algun defecte. Per tant, el primer pas és l'ocurrència d'un defecte. Si no existeix cap tractament, aquest generarà un error. Si tampoc hi ha una gestió que els resolga, causaran fallades. Transitivament, totes les fallades estan causats per un o més defectes.
X	Es generarà una fallada si hi ha errors i no existeix redundància en el component que ha patit els errors.
	La replicació resol totes les situacions d'error, perquè no es convertisquen en fallades. La replicació és la gestió recomanada perquè no hi haja fallades. No obstant això, no totes les situacions d'error es podran resoldre mitjançant replicació, si aquesta no és suficientment acurada. La replicació ha d'estar acompanyada per mecanismes de diagnòstic adequats i per certs protocols que asseguren la correcció de les rèpliques que hagen superat la situació d'error. Per exemple, quan no s'haja desplegat un nombre suficient de rèpliques o totes elles depenguen de la mateixa font d'errors, els errors seguiran provocant fallades malgrat la replicació. En aquesta situació, un error podria afectar totes les rèpliques d'un determinat servei i cap d'elles continuaria amb el seu treball, deixant el servei indisponible.
	Amb un mecanisme de diagnòstic adequat i un tractament correcte es pot evitar que les situacions de fallada provoquen errors. Ocorre el contrari. Amb aquestes gestions es podria evitar que les situacions d'error es convertisquen en fallades.
	Totes les anteriors.
	Cap de les anteriors.

36. Exemples de model de fallades:

	Actiu, passiu. No són exemples de models de fallades, sinó de models de replicació.
	Canal i enllaç, xarxa, transport. Els quatre termes esmentats guarden relació amb les comunicacions però no són models de fallades. "Canal i enllaç" són dos termes sinònims. "Caiguda i enllaç" és el nom d'un model de fallades.
X	Parada, caiguda, omissió general. Aquests són tres dels models explicats. Hi ha uns altres: omissió de recepcions, omissió d'enviaments, arbitraris…
	Mantenibilitat, partició primària, consistència seqüencial. No són exemples de models de fallades. La "mantenibilitat" és una propietat o atribut que han de complir els sistemes robusts. La "partició primària" és un tipus de gestió per a afrontar les situacions de particionat de la xarxa. La "consistència seqüencial" és un model de consistència aplicable a sistemes replicats.
	Totes les anteriors.
	Cap de les anteriors.

37. Atributs que defineixen un sistema robust:

X	Fiabilitat, mantenibilitat, seguretat, disponibilitat. Aquests són els quatre atributs que apareixen en la seua definició. "Seguretat" tenia dues accepcions: evitació de desastres, d'una banda, i confidencialitat i protecció, per una altra.
	Consistència, atomicitat, aïllament, durabilitat. Aquestes són les quatre garanties que ha d'oferir tota transacció acceptada.
	Escalabilitat, elasticitat, adaptabilitat, eficiència. Aquests són atributs que guarden relació amb el rendiment d'un sistema, però no defineixen la seua robustesa.
	Correcció, eficàcia, facilitat d'ús, vivacitat. Són quatre exemples més d'atributs aconsellables per a qualsevol sistema informàtic, però no defineixen la seua robustesa.
	Totes les anteriors.
	Cap de les anteriors.

A continuació presentem el mòdul nodejs, shared.js.

```
// shared memory module
//
var zmq = require('zmq');
var id = "";
var pb = zmq.socket('pub');
var sb = zmq.socket('sub');
var local = {};

sb.subscribe("");
pb.bind('tcp://*:8888');

function join(nodes) {
        nodes.forEach(function (ep) {
          sb.connect(ep);
        });
}
function myID(nid) {
        id = nid;
}

sb.on('message', function (name, value) {
        local[name] = value;
});

// initialization
exports.init = function(nodes, id) {
   join(nodes);
   myID(id);
}
// Write function
exports.W = function (name, value) {
        pb.send([name, value]);
        local[name] = value;
        console.log("W" + id + "(" + name + ")" + value + ":");
};
// Read function
exports.R = function (name) {
        console.log("R" + id + "(" + name + ")" + local[name] + ":");
        return local[name];
};
```

Aquest mòdul implementa operacions *Read* (**R)** i *Write* (**W**) sobre un conjunt de variables compartides per un grup de processos.
Quan un procés vol integrar-se en el grup de compartició, realitza un require('shared.js'), i ho inicialitza amb les URLs dels endpoints dels altres processos en el grup, i un identificador per a si mateix.
shared.js imprimeix un log amb les operacions **R**/**W** realitzades, incloent els valors. Això representa la seqüència d'esdeveniments R/W en el procés en l'ordre en què són produïts.
Suposem que llancem quatre processos, amb ids 1,2,3,4, compartint la variable "X", i els llancem sobre la mateixa consola de ordres. Els esdeveniments de tots els processos, a mesura que són impresos per shared.js, es van a barrejar en l'eixida de la consola. Suposar que cap missatge pub es perd.
Respondre a les següents dues preguntes en el context anterior.

38. Seleccionar la seqüència de reads/writes que pot observar-se en la consola:

	R1(X)2:W1(X)4:R2(X)4:W3(X)67:R4(X)undefined: S'assumeix que la variable X no estava inicialitzada. Per això, si la primera operació en una execució fóra una lectura, com en aquest cas, el procés corresponent obtindria el valor "undefined". Això no ocorre en aquesta execució. Per tant, no seria possible observar aquesta seqüència. No té sentit obtenir un 2 abans que siga escrit per algun procés. De fet, cap d'ells arriba a escriure aquest valor.
X	R1(X)undefined:W1(X)4:R2(X)4:W2(X)67:R4(X)4: En aquest cas la primera lectura sí que obté un valor justificable ja que no s'havia fet encara una escriptura. Al seu torn, el procés 4 llig un valor 4 després que el procés 2 haja escrit el valor 67. Això també és admissible ja que pot ser que l'enviament realitzat pel procés 2 mitjançant el socket "pub" utilitzat encara no haja arribat al procés 4.
	R2(X)2:W2(X)4:R1(X)4:R3(X)67:R4(X)undefined: Es dóna una paradoxa similar a la de la primera execució: llegir un valor abans que aquest haja sigut escrit, assumint que arribarà a escriure's en algun moment futur.
	R1(X)2:W2(X)2:R1(X)4:W3(X)4:R4(X)5: Es dóna una paradoxa similar a la de la primera execució: llegir un valor abans que aquest haja sigut escrit.
	Totes les anteriors.
	Cap de les anteriors.

39. El mòdul shared.js implementa la consistència…:

	Seqüencial Perquè fóra seqüencial, el mecanisme de difusió de les escriptures hauria de preocupar-se per arribar a un acord sobre un mateix ordre de lliurament en tots els processos destinataris. A més, aquest ordre hauria de respectar l'ordre en què cada procés haja fet les seues escriptures localment (ordre FIFO) i l'ordre causal. Els sockets "pub/sub" de ZeroMQ no es preocupen per garantir un ordre total en els missatges propagats.
	Processador La consistència processador requereix lliurament en ordre total (consistència "*cache*") i que aquest ordre total respecte també l'ordre FIFO, encara que no el causal. Com ZeroMQ no proporciona ordre total en els sockets "pub/sub", la consistència processador no està garantida.
	Causal Per a garantir la consistència causal sol ser necessari un etiquetatge dels missatges difosos amb els rellotges vectorials associats als seus esdeveniments d'enviament. En aquest cas no s'ha arribat a realitzar aquesta gestió.
X	FIFO La consistència FIFO pot garantir-se si els missatges es lliuren en cada destinatari en l'ordre en què els va enviar el seu procés emissor. Hi ha llibertat per a intercalar de diferents maneres els missatges difosos per diferents processos. Això pot arribar a respectar-se amb sockets "pub/sub".
	Totes les anteriors.
	Cap de les anteriors.

40. Objectius generals de la replicació:

	Augmentar la disponibilitat. Quantes més rèpliques hi haja, més difícil serà que un servei no estiga disponible quan hi haja errors.
	Augmentar el rendiment. Si s'incorporen més rèpliques a un servei es podrà repartir la càrrega entre totes elles i millorarà el rendiment general.
	Disminuir el temps de servei de les peticions. En repartir la càrrega entre totes les rèpliques i augmentar el nombre de rèpliques, cada procés servidor rebrà un percentatge menor de la càrrega global i podrà acabar més ràpidament el servei de cada petició (ja que el grau de concurrència serà menor i apareixeran menys conflictes entre les activitats que s'executen en una mateixa rèplica).
	Implantar l'escalabilitat horitzontal. Com a conseqüència de tot l'explicat en els tres apartats anteriors, el servei replicat millora la seua escalabilitat. L'escalabilitat obtinguda mitjançant replicació es coneix com escalabilitat horitzontal.
X	Totes les anteriors.
	Cap de les anteriors.