Osztott rendszerek jegyzet

Tóth Ákos
Eötvös Loránd Tudományegyetem Informatikai Kar
Programtervező Informatikus BSc C szakirány

Budapest, 2016

1. Definíciók

Elosztott rendszer: Önálló számítógépek olyan összessége, amely kezelői számára egyetlen koherens rendszernek tűnik.

Nyitott elosztott rendszer: A nyitott elosztott rendszer képes más nyitott rendszerek számára szolgáltatásokat nyújtani, és azok szolgáltatásait igénybe venni. Jellemző tulajdonságai, hogy jól definiált interface-el rendelkezik, valamilyen szintű hordozhatóság(*portability*) és együttműködési (*interoperability*) (más rendszerekkel), támogatása.

P2P Overlay network: Egy peer-to-peer overlay network egy számítógépes hálózat, amelyet valamilyen meglévő hálózatra építenek fel, általában az Internetre. Peer to peer overlay hálózatok előnye skálázhatóság, megbízhatóság; szemben a tradicionális kliens-szerver rendszerekkel. Legtöbbször fájl megosztás illetve valós idejű adat folyamoknál használják.

Köztes réteg: Olyan szolgáltatásokat és protokollokat lehet sorolni, amely sokfajta alkalmazáshoz lehetnek hasznosnak.

- kommunikációs protokollok
- sorosítás(serialization,marshalling)
- skálázási mechanizmusok(replikáció,cache-elés)
- biztonsági protokollok
- elnevezési protokollok, az erőforrások megosztására

Megtartó kommunikáció: A rendszer hajlandó huzamosabb ideig megtartani az üzeneteket.

Időleges kommunikáció: A kommunikációs rendszer elveti az üzenetet, ha az nem kézbesíthető.

Kliens-szerver modell: A kliens-szerver modell jellemzően *időleges(transient), szinkron* kommunikációt használ. A kliensnek és a szervernek egyidőben kell aktívnak lennie. A kliens a kérések elindítása után blokkolódik, amíg választ nem kap a szervertől. A szerver a kliensek fogadásával és kiszolgálásával foglalkozik.

Üzenetorientált köztesréteg(MOM): Message Oriented Middleware. *Megtartó(persistent), aszinkron* kommunikációs forma. Segítségével a folyamatok üzeneteket küldhetnek egymásnak. A küldő félnek nem kell válaszra várni, foglalkozhat mással. Gyakran jobb hibatűrést biztosít.
*Működési elv:*A köztesréteg sorokat(queue) tart fenn a rendszer gépein. A kliensek az alábbi műveletek használják a várakozási sorokra:

PUT: Üzenetet tesz egy várakozási sor végégre.

GET: Blokkol, amíg a sor üres, majd leveszi az első üzenetet.

POLL: Nem blokkoló módon lekérdezi van-e üzenet, ha igen leveszi az elsőt.

NOTIFY: Kezelőrutint telepít a várakozási sorhoz, amely minden beérkező üzenetre meghívódik.

Üzenetkezelő rendszerek homogenitása: Az üzenet kezelő rendszerek feltételezik, hogy a rendszer minden eleme közös protokollt használ, azaz az üzenetek szerkezete és adatábrázolása megegyező.
Üzenet közvetítő (message broker): Olyan központi komponenes, amely heterogén környezetben gondoskodik a megfelelő konverzióról: Átalakítja az üzeneteket a fogadó formátumára, illetve szerepe szerint átjáró(application level gateway, proxy), továbbá egyéb funkciókat is ellát(pl.: biztonsági); az üzenetek tartalmát is megvizsgálhatják az útválasztáshoz(Subject based vs Object based routing) $\Rightarrow$ Enterprise Application Integration.

Távoli eljáráshívás(RPC): A Remote Procedure Call vagy az alprogramok használata természetes a fejlesztés során, amelyek a jó esetben egymástól függetlenül működnek("fekete doboz"), ezáltal akár távoli gépeken is futtathatjuk. A távoli gépen futtatandó eljárás eléréséhez kommunikációra van szükség, ezt eljáráshívási mechanizmus fedi el.
RPC hívás lépései:

1. A kliensfolyamat lokálisan meghívja a klienscsonkot
2. A klienscsonk becsomagolja az eljárás azonosítóját és paramétereit, meghívja az OS-t.
3. A helyi OS átküldi a távoli OS-nek.
4. Távoli OS átadja az üzenetet a szervercsonknak.

5. A szervercsonk kicsomagolja a paramétereket és átadja a szervernek.

6. A szerver lokálisan meghívja az eljárást, majd megkapja a visszatérési értéket. Ennek visszaküldése a kliensnek hasonlóan zajlik, fordított irányban.

Mivel az elosztott rendszer résztvevőinek adatábrázolása különböző lehet, ezért az adatok sorosítása során(serializing, marshaling) közös formátumra alakítjuk az adatokat; a csonkok fordítanak a közös formátum és a helyi adatábrázolás között.
Paraméterátadás történhet érték-eredmény szerint: Minden feldolgozandó adat paraméterként kerül az eljárásba, nincsen globális hivatkozás. Nem érhető el teljes mértékű átlátszóság. Az átlátszóság növelése a *Távoli hivatkozások* segítségével történhet: A távoli adat egységesen érhető el; a hivatkozás paraméterként átadható az RPC hívásban.
Aszinkron RPC: A szerver nyugtázza a kérés beérkezését, a kliens nem várakozik az eredményre.
*Késleltetett, szinkronizált RPC:*Két aszinkron RPC hívás egymással összehangolva: Kliens elküldi a kérést, várakozik a szerver nyugtájáig. Ha a szerveren az eljárás hívás megtörtént, a szerver elküldi az adatok egy egyirányú aszinkron RPC hívással a kliensnek. További lehetőség, hogy az RPC hívás után, a kliens időnként lekérdezi a szervert a hívás állapotáról és az eredményekről.
Kliens csatlakozása a szerverhez:

1. A szolgáltatások katalógusba jegyzik be az elérhetőségüket, globálisan és lokálisan is.

2. A kliens kikeresi a szolgáltatást a katalógusból.

3. A kliens végpontot igényel a démontól a kommunikációhoz.

2. Kérdések és válaszok

2.1. Rövid kérdések

1. diasor

Egy-egy mondattal jellemezz kétfajta (nem rokon jellegű) átlátszóságot.

Fajta	Angolul	Mit rejt el az erőforrással kapcsolatban?
Hozzáférési/elérési	Access	Adatábrázolás; elérés technikai részletei
Elhelyezési	Location	Fizikai elhelyezkedés
Áthelyezési	Migration	Elhelyezési + a hely meg is változhat
Mozgatási	Relocation	Áthelyezési + használat közben is történhet az áthelyezés
Többszörözési	Replication	Az erőforrásnak több másolata is lehet a rendszerben
Egyidejűségi	Concurrency	Több versenyhelyzetű felhasználó is elérheti egyszerre
Meghibásodási	Failure	Meghibásodhat és újra üzembe állhat

Mi jellemzi a nyitott rendszereket?

- jól definiált interface-kel rendelkeznek
- Az alkalmazások hordozhatóságát minél inkább támogatják(portability)
- könnyen elérhető a rendszerek együttműködése(*interoperability*)

Mi a nyitottság implementálásának főbb jellemzői?

- Fontos, hogy a rendszer könnyen cserélhető részekből álljon
- Belső interface-k használata, nem egyetlen monolitikus rendszer
- A rendszernek minél jobban paraméterezhetőnek kell lennie

Egy-egy mondattal jellemezz kétfajta átméretezhetőséget.

- Méret szerint: Több felhasznál és/vagy folyamat (Könnyebben kezelhető például erősebb szerverekkel!)
- Földrajzi: A rendszert nagyobb területen veszik igénybe: egyetemen belüli felhasználás $\rightarrow$ világméretű felhasználóbázis
- Adminisztrációs: Biztonsági, karbantartási, együttműködési kérdések merülnek fel, ha új adminisztrációs tartományok kerülnek a rendszerbe

Ismertess egy technikát, amelynek célja az átméretezhetőség megvalósítása.

- A kommunikációs késleltetés elfedése
  - Aszinkron kommunikáció használata

  - A beérkező választ külön kezelő dolgozza fel
  - Probléma, hogy nem minden alkalmazás ültethető át ilyen formában
- Elosztás
  - A számítások egy részét a kliensoldal végzi(Java appletek)
  - Decentralizált elnevezési rendszerek(DNS)
  - Decentralizált információs rendszerek(WWW)
- Replikáció/cache-elés
  - Replikált fájlszerverek és adatbázisok $\rightarrow$ inkonzisztencia veszélye $\rightarrow$ globális szinkronizáció szükséges(Rosszul skálázható!)
  - Tükrözött weboldalak
  - Fájlok cache-elés(böngészőkben, proxy szervereken)
  - Weboldalak cache-elés(a szerver- és kliensoldalon)

Miben hasonlítanak és miben térnek el a cluster és a grid rendszerek?

Cluster	Grid
elosztott számítási rendszer	
lokális hálózat	nagyméretű hálózat
központosított vezérlés	átívelhet több szervezeti egységen
homogén számítógépek	több, kevésbé egységes számítógép

Mi az ACID (Legalább két szempont részletezve, a másik kettőnek legalább a neve)?

Tranzakcióknál használt követelményrendszer:

- **Atomicity**: vagy a teljes tranzakció végbemegy vagy nem változik az adattár
- **Consistency**: A tranzakció konzisztens, ha érvényes állapotot állít elő. (csak a tranzakció lefutása után kell teljesülnie)
- **Isolation**: Egyszerre zajló tranzakciók nem zavarják egymást: olyan eredményt adnak, mintha egymás után sorban futottak volna le.
- **Durability**: Végrehajtás után az eredményt tartós adattárolóra mentjük, így a rendszer esetleges összeomlása után visszaállítható.

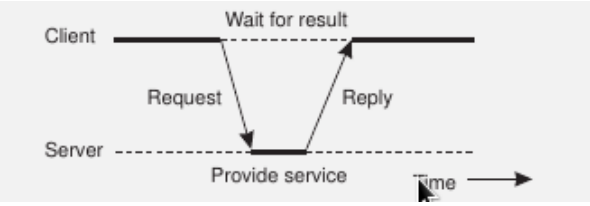
Adj meg olyan feltételezést elosztott rendszerrel kapcsolatban, amellyel kényelmes élni, de a valóságban akadályokat gördíthet elénk.

- hálózat hibamentes
- hálózat biztonságos
- hálózat homogén
- hálózati topológia nem változik
- kommunikációnak nincsen időigénye
- sávszélesség korlátlan
- kommunikációnak nincsen költsége
- csak egy adminisztrátor van

2. diasor

Írd le az egyszerű kliens-szerver modellben a kommunikáció menetét.

1. Kliens kérést küld a szervernek, majd várakozik a válaszra.
2. Szerver fogadja a kérés, válaszol.
3. Kliens megkapja a választ, folytatja a működést.



Milyen három rétegbe szokás osztani az alkalmazásokat?

- Megjelenítés - felhasználó felületet alkotó komponensek (view)
- Üzleti logika - alkalmazás működését írja le konkrét adatok nélkül (controller)
- Perzisztencia - adatok tartós tárolása (model)

Mi az "overlay"?

Számítógépes hálózat, amely egy meglévő hálózatra építkezik. Ha a hálózatokat gráffal reprezentáljuk, a szomszédos csúcsok a fizikai hálózaton lehetnek távol

egymástól, a rendszer elfedi, hogy a köztük levő kommunikáció több gépet érintve történik. A legtöbb P2P rendszer alapja.

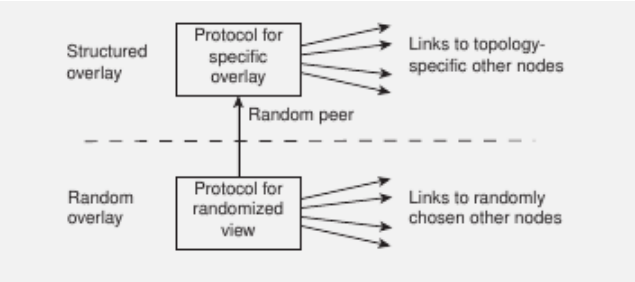
Milyen az overlay peer-to-peer hálózatok felépítése?

Különböztessünk meg két réteget:

(1) Az alsó rétegben a csúcsoknak csak részleges nézete van.

(2) A felső rétegben csak kevés csúcs kerülhet.

Az alsó réteg véletlenszerű csúcsokat ad át a felső rétegnek. A felső réteg ezek közül csak keveset tart meg.



Ismertess (nagyon) vázlatosan egy strukturált peer-to-peer rendszert.

A csúcsokat valamilyen struktúra szerint overlay hálózatba szervezzük(Pl.:logikai gyűrű), és a csúcsoktól az azonosítójuk alapján lehet szolgáltatásokat igénybe venni. Például elosztott hasítótábla megvalósítása: ebben a rendszerben kulcs-érték párokat tárolunk. Az adott értéket tároló csúcsot hatékonyan meg lehet keresni a kulcsa alapján, akármelyik csúcsra is érkezik be a kérés.

Hogy működik a struktúrálatlan peer-to-peer pletykálás?

selectPeer: A részleges nézetből kiválaszt egy szomszédot.

selectToSend: Az általa ismert szomszédok közül kiválaszt n darabot.

selectToKeep: A megkapott csúcsokat eltárolja lokálisan. Eltávolítja a többszörösen szereplő csúcsokat. A tárolt csúcsok számát m darabra csökkenti. Erre többfajta stratégia lehetséges.

Hogyan működnek a struktúrálatlan peer-to-peer rendszerek?

Ezek a rendszerek igyekeznek véletlen gráf struktúrát fenntartani. Mindegyik csúcsnak csak részleges nézete van a gráfról.

Minden P csúcs időközönként kiválaszt egy szomszédos Q csúcsot - P és Q információt cserél, valamint átküldik egymásnak az általuk ismert csúcsokat

Mi a superpeer? Egy-két mondattal jellemezz olyan rendszert, amelyben megtalálható.

Olyan kisszámú csúcs, amelyeknek külön feladata van.
- kereséshez index fenntartása
- a hálózat állapotának felügyelete
- csúcsok közötti kapcsolatok létrehozása
Példa:
- kliens-szerver + P2P: BitTorrent hibrid architektúrában a tracker, amely az egyes peer-ek, megosztásait figyeli (indexeli Pl.: feltöltési sebesség alapján).
- kliens-szerver + P2P: Edge szerver hibrid architektúrában a CDN rendszerekben az Information Broker-ek figyelemmel kísérik a hálózatot és a kliensek kéréseit, amiket a nekik legmegfelelőbb edge szerverhez irányítják.

Mi az interceptor?

Távoli objektum elérése során a vezérlés szokásos menetébe avatkozik bele. Pl.: átalakíthatja más formátumra a kérést. Jellemzően az architektúra rétegei közé illeszthető.

Milyen az önszervező rendszerek általános architektúrája?

elvárható tulajdonságok:

- önkonfiguráló
- önkezelő
- öngyógyító
- ön optimalizáló
- ön*

Mi az az edge server?

Az adatokat tároló szerver. A kliensekhez minél közelebb van elhelyezve, ahol egy nagyobb hálózat az Internetre csatlakozik.

Mi az a Content Delivery Network?

Tartalom szolgáltató hálózat. A tartalom szolgáltatás hatékonyságát növelik és költségét csökkentik.

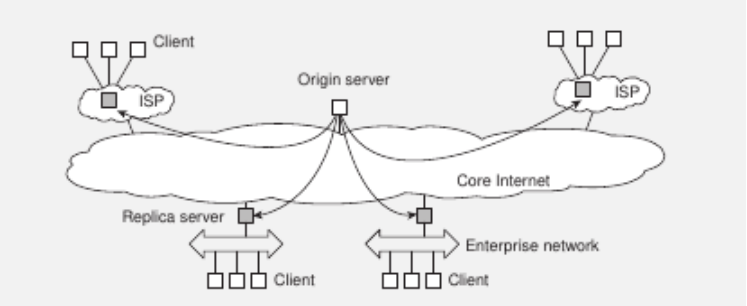
Adj módszert arra, hogyan válasszuk meg, melyik szerverek tároljanak egy adott fájlt!

Visszacsatolásos modell:Mérik, hogy a rendszer mennyire tér el a kívánt tulajdonságoktól, és szükség szerint változtatnak a beállításokon. ⇒ Például Globule

Globule (CDN)

- tartalmakat költségmodell alapján helyezi el
- A központi szerver elemzi, hogy mi történt volna, ha P oldalt az S edge szerver tárolta volna.

A számításokat különböző stratégiákra végzi el, végül a legjobbat választja ki.



3. diasor

Mi a kontextusváltás?

A másik folyamatnak/szálnak történő vezérlésátadás, illetve a megfelelő kontextusok cseréje. Így egy processzor több folyamatot/szálat is végre tud hajtani.

Milyen két fő megközelítés létezik a virtualizációra? Az egyik rövid jellemzésével.

Process VM: A virtuális program közönséges programként fut ⇒ bytecode-ot hajt végre Pl.: JVM, CLR

VM Monitor: Hardver teljes körű virtualizációja ⇒ bármely operációs rendszer futtatására alkalmas Pl.: VirtualBox

Mi a szuperszerver?

Olyan szerver, amelyik több port-on figyeli a bejövő kapcsolatokat, és amikor új kérés érkezik, új folyamatot/szálat indít annak kezelésére.

Mi a iteratív és konkurens szerver?

Iteratív: Egyszerre csak egy kapcsolatot tud kezelni

Konkurens: Párhuzamosan több kapcsolatot is tud kezelni

Mi jellemzi az állapot teljes szervereket?

Állapotot tart számon a klienstől: megjegyzi, melyik fájlokat használta a kliens, és

ezeket előre megtudja nyitni legközelebb; illetve eltárolja milyen adatokat töltött le a kliens, és frissítéseket küldhet neki

Mi a kódmigráció?

Olyan kommunikáció, amely során nem csak adatokat küldünk át

Nevezz meg legalább kétfajta feladatot, amely során kódmigráció történik, és részletezd az egyiket.

Client-Server: A szokásos kliens-szerver kommunikáció, nincsen kódmigráció

Remote Evaluation: A kliens feltölti a kódot, és a szerveren futtatja

Code on Demand: A kliens letölti a kódot a szerverről, és helyben futtatja

Mobile Agent: A mobil ágens feltölti a kódját és az állapotát, és a szerveren folytatja a futását

Mik az objektum komponensek?

Kódszegmens ⇒ a programkódot tartalmazza

Adatszegmens ⇒ a futó program állapotát tartalmazza

Végrehajtási ⇒ szegmens: a futtató szál környezetét tartalmazza

Mi a gyenge mobilitás?

a kód és az adatszegmens mozgatása (⇒ a kód újraindul) - Viszonylag egyszerű megtenni ha a kód hordozható - irány szerint > feltöltés(push,ship) > letöltés(pull, fetch)

Mi az erős mobilitás?

A komponens végrehajtása szegmenssel együtt költözik - migráció ⇒ az objektum átköltözik egyik gépről másikra - klónozás ⇒ egy másolat kerül a másik gépre, mindkét gépen ugyanonnan folytatódik a futás 4. diasor

Mi az ISO/OSI modell alsó három rétegének feladata?

Fizikai réteg - a bitek átvitelének fizikai részei írja lefutása

Adatkapcsolati réteg - az üzenetet keretekre tagolja - a hibajavítás és a hálózat terhelésének korlátozása

Hálózati réteg - a hálózat távoli gépei között közvetít csomagokat útválasztás (routing) segítségével

Mi a szállítási réteg két fő protokollja, és ezeknek mik a főbb jellemzői?

TCP ⇒ kapcsolatalapú, megbízható, sorrendhelyes átvitelének

UDP ⇒ nem (teljesen) megbízható, általában kis üzenetek (datagram) átvitele

Mi a köztes réteg?

A köztes rétegbe (middleware) olyan szolgáltatásokat és protokollokat szokás sorolni, amelyek sokfajta alkalmazáshoz lehetnek hasznosak.

Mi jellemzi az időleges, szinkron kommunikációt?

A kommunikációs rendszer elveti az üzenetet, ha az nem kézbesíthető.

Mi jellemzi a megtartó, aszinkron kommunikációt?

A kommunikációs rendszer hajlandó huzamosan tárolni az üzenetet.

Milyen lépésekből áll az RPC hívás iránya? ⇒ A kliensfolyamat lokálisan meghívja a klienscsonkot ⇒ Az becsomagolja az eljárás azonosítóját és paramétereit, meghívja az OS-t. ⇒ Az átküldi az üzenetet a távoli OS-nek. ⇒ Az átadja az üzenetet a szervercsonknak. ⇒ Az kicsomagolja a paramétereket, átadja a szervernek. ⇒ A szerver lokálisan meghívja az eljárást, megkapja a visszatérési értéket. ⇒ Ennek visszaküldése a klienshez hasonlóan zajlik, fordított irányban.

Mit tud az RPC paraméter átadásról?

kliens és szervergépen eltérhet az adatábrázolás ⇒ szerializálás szükséges - rögzíteni kell a paraméterek kódolását - A két csonknak fordítania kell a közös formátumról a gépeik formátumára

Érték–eredmény szerinti paraméter átadási szemantika

Hogyan kezelhetőek a hivatkozások RPC hívás során?

Távoli hivatkozás bevezetésével növelhető az elérési átlátszóságot - A távoli adat egységesen érhető el - A távoli hivatkozásokat át lehet paraméterként adni ebben nem vagyok biztos

Milyen lépésekből áll a socket kommunikáció? (Sorrenddel.)

létrejön a socket

csatlakozik a szerverhez($\Rightarrow$ a szerver fogadja a csatlakozást)

a socket ír a szerverre ($\Rightarrow$ szerver fogadja majd válaszol)

a socket fogadja a választ

az előző két lépés tetszőleges számban ismétlődik

a socket zárja a kapcsolatot

Adatátviteli módok folyamatos média esetén

aszinkron - nem ad megkötést hogy mikor kell átvinni az adatot

szinkron - az egyes csomagoknak egy bizonyos idő alatt kell a célba érniük

izokron - alsó és felső korlátot is ad a csomagok átvitelére

Mi a folyam? (Fontosabb jellemzőkkel.)

adatfolyam - izokron kommunikációt támogató kommunikációs forma

egyirányú

legtöbbször egy forrástól(source) egy vagy több nyelő(sink) felé(gyakran közvetlenül csatlakozva hardverre)

egyszerű folyam - egyfajta adatot továbbító

összetett folyam - többfajta adatot továbbít

Miben tér el az anti-entrópia és a pletykálás alapú járványalgoritmus? (Előnyökkel, hátrányokkal.)

Anti-entrópia - Minden szerver rendszeresen kiválaszt egy másikat - kicserélik egymás között a változásokat.

Pletykálás (gossiping) - Az újonnan frissült (megfertőzött) szerver elküldi a frissítést néhány szomszédjának (megfertőzi őket).

Mi az a QoS?

Quality of Service - a folyamokkal kapcsolatban vázolt követelmények

példa - a folyam átvitelének sebessége - a folyam megindításának legnagyobb késleltetése - stb...

Biztosítása - differenciált szolgáltatás architektúra > hálózat router-i kategorizálják az áthaladó forgalmat ⇒ egyes csomagfajta elsőbbséget élvez - remegés csökkentése > a router-k pufferelhetik az adatokat 5. diasor

Mi a broadcasting?

kihirdetjük az azonosítót a hálózaton ⇒ az egyed visszaküldi a jelenlegi címét - lokális hálózatokon túl nem skálázódik - a hálózaton minden gépnek figyelni kell a beérkező kérésre

Mi a továbbító mutató?

amikor az egyed elköltözik , egy mutató marad utána az új helyre - a kliens elől rejtve van - a megtalált cím visszaküldhető ⇒ további feloldások gyorsabbak

Mi a Chord elosztott hasítótábla?

elosztott hasított táblát készítünk - csúcsok tárolnak egyedeket - N csúcsú gyűrű overlay szerkezetbe van szervezve

minden csúcshoz véletlenszerű azonosító(m bit) - mindegyik entitáshoz kulcs(m bit)

a :k kulcsú egyed felelőse az :id azonosítójú csúcs - k <= id és nincs köztük másik csúcs - jelölés > felelős csúcs a kulcs rákövetkezője > jel: succ(k)

Mit tartalmaz a Chord adatábrázolás egy finger table-je?

minden p csúcs tárol FTp "finger table" -t m bejegyzéssel - FTp[i = succ(p

2ˆ(i-) - bináris jellegű keresés ⇒ minden lépés felezi a keresési tartományt > 2ˆ(m-, 2ˆ(m-,. . . ,1

Mi az a HLS?

Hierarchical Location Service

A hálózatokat osszuk tartományokra - tartozzon minden tartományhoz egy katalógus

építsünk hierarchiát a katalógusokból

Mik a katalógus csúcsok?

az E egyed címe egy levélben található

gyökértől az E levélig vezető úton minden belső csúcsban van egy mutató a következő gyerekre

a gyökér minden út kiindulópontja - minden egyedről van információja

Jellemezd három szempont szerint, hogyan különbözik a DNS névtér három szintje.

globális szint ⇒ gyökér és felső csúcsok. A szervezetek közösen kezelik

szervezeti szint ⇒ egy egy szervezet által kezelt csúcsok szintje

kezelői szint ⇒ egy adott szervezeten belül kezelt csúcsok szempont |globális | szervezeti | kezelői ————————

————————-

————————

———————— méret | világméretű | vállalati | vállalati alegység csúcsok száma | kevés |sok |rendkívül sok keresés ideje | mp. |ezredmp. | azonnal

Hogyan működik az iteratív névfeloldás?

gyökér névszerverek egyikétől indítjuk

a névnek csak egy komponensét oldjuk fel - a megszólított névszerver az ehhez tartozó névszerver címét küldi vissza

Hogyan működik a rekurzív névfeloldás?

a névszerverek egymás között kommunikálva oldják fel a nevet - a kliens oldali név feloldóhoz rögtön a válasz érkezik

Mit mondhatunk a névfeloldás átméretezhetőségéről?

problémák » sok kérés rövid idő alatt ⇒ globális szint szerverei nagy terhelés kapnának » földrajzi távolságokat is figyelembe kell venni » egy adott csúcsot egy adott névszerver szolgál ki , földrajzilag oda kell csatlakozni

csúcsok adatai sok névszerveren - felső két szinten alig változik a gráf - legtöbb csúcs adatairól másolatok készíthetők több névszerverre

keresett adat az entitás címe - névszerverek nem alkalmasak mozgó entitások címeinek kezelésére » mert azok költözésével változna a gráf

Mi az X.500?

A katalógus szolgáltatásokban az attribútumokra érvényes megkötések egyfajta szabványa

LDAP protokollon szokás elérni

Az elnevezési rendszer fastruktúrájú - élei attribútum-érték párokkal címzettek - Az egyedekre az útjuk jellemzői vonatkoznak, és további párokat is tartalmazhatnak. 6. diasor

Hogyan működik a Cristian-algoritmus?

mindegyik gép egy központi időszervertől kéri le a pontos időt (Network Time Protocol) - Nem a megkapott időre kell állítani az órát: bele kell számítani, hogy a szerver kezelte a kérést és a válasznak vissza kellett érkeznie a hálózaton keresztül

Hogyan működik a Berkeley-algoritmus?

Itt nem feltétlenül a pontos idő beállítása a cél, csak az, hogy minden gép ideje azonos legyen. - álagot von minden gép idejéből $\Rightarrow$ majd mindenkit értesít, hogy a saját óráját mennyivel kell átállítania - Az idő egyik gépnél sem folyhat visszafelé: ha vissza kellene állítani valamelyik órát, akkor ehelyett a számontartott idő mérését lelassítja a gép mindaddig, amíg a kívánt idő be nem áll.

Hogyan működik az előbb történt (happened - before) reláció?

Ha ugyanabban a folyamatban az a esemény korábban következett be beseménynél, akkor a -> b

Ha a esemény egy üzenet küldése, és b esemény annak fogadása, akkor a -> b

A reláció tranzitív: ha a -> b és b -> c, akkor a -> c

Az idő és az előbb történt reláció kapcsolata?

Minden e eseményhez időbélyeget rendelünk, ami egy egész szám (Jelölése: C(e)) és megköveteljük az alábbi tulajdonságokat: - Ha a -> b egy folyamat két eseményre, akkor C(a) < C(b) - Ha a esemény egy üzenet küldése és b esemény annak fogadása, akkor C(a) < C(b)

Mi a Lamport-féle időbélyeg?

Minden Pi folyamat saját Ci számlálót tart nyilván az alábbiak szerint: - Pi minden eseménye eggyel növeli a számlálót - Az elküldött m üzenetre ráírjuk az időbélyeget: ts(m) = Cj - Ha az m üzenet beérkezik Pj folyamathoz, ott a számláló új értéke Cj = maxCj, ts(m)

1 lesz így az idő nem folyik visszafelé - Pi és Pj időbélyegei közül tekintsük a Pi-belit elsőnek, ha i < j

Mi az időbélyeg-vektor?

A Pi most már az összes másik folyamat idejét is számon tartja egy VCi[1...n tömbben, ahol VCi[j azoknak a Pj folyamatban bekövetkezett eseményeknek a száma, amelyekről Pi tud

Az m üzenet elküldése során Pi megnöveli eggyel VCi[i értékét, és a teljes Vi időbélyegvektort ráírja az üzenetre

Amikor az m üzenet megérkezik a Pj folyamatkozhoz, amelyekre a ts(m) időbélyeg van írva, két dolog történik: - VC,[k := maxVCj[k, ts(m)[k - VCj[j megnő eggyel, vagyis az üzenet fogadása is egy eseménynek számítani

Pontosan sorbarendezett csoportcímzés definíciója?

Az időbélyeg-vektorokkal megvalósítható a pontosan sorbarendezett csoportcímzés: csak akkor kézbesítjük az üzeneteket, ha már mindegyik előzményüket kézbesítettük.

Kölcsönös kizárás feladata?

Több folyamat egyszerre szeretne hozzáférni egy adott erőforráshoz. Ezt egyszerre csak egynek engehetjük meg közülük, különben az erőforrás helytelen állapotba kerülhet

Megoldásfajták: - Központi szerver - Peer-to-peer rendszereken alapuló teljesen elosztott megoldás - Teljesen elosztott megoldás ált. gráfszerkezetre - Teljesen elosztott megoldás gyűrűben

Ismertesd a központosított kölcsönös kizárási algoritmust.

Tegyük fel, hogy az erőforrás n-szeresen többszörözött, és minden replikátumhoz tartozik egy azt kezelő koordinátor. Az erőforráshoz való hozzáférésről többségi szavazás dönt: legalább m koordinátor engedése szükséges, ahol m > n/2. Feltesszük, hogy egy esetleges összeomlás után a koordinátor hamar felépül - azonban a kiadott engedélyeket elfelejti.

Elosztott kölcsönös kizárás működési elve?

Ismét többszörözött az erőforrás, amikor a kliens hozzá szeretne férni, kérést küld mindegyik koordinátornak. Választ akkor kap, ha - a koordinátor nem igényli az erőforrást, vagy - a koordinátor is igényli az erőforrást, de kisebb az időbélyegei - különben a koordinátor nem válaszol

Ismertesd a zsetongyűrű alapú kölcsönös kizárási algoritmust.

A folyamatokat logikai gyűrűbe szervezzük. A gyűrűben egy zsetont küldünk körbe, amelyik folyamat birtokolja, az férhet hozzá az erőforráshoz

Csúcsok globális pozícionálása. Meg szeretnénk becsülni a csócsok közötti kommunikációs költségeket. Erre többek között azért van szükség, hogy hatákonyan tudjuk megválasztani, melyik gépekre helyezzünk replikátumokat az adatainkból.

Ábrázolás: A csúcsokat egy többdimenziós geometriai térben ábrázoljuk, ahol a P és Q csúcsok közötti kommunkiációs költséget a csócsok távolsága jelöli.

Zsarnok-algoritmus A folyamatoknak sorszámot adunk. A legnagyobb sorszámű folyamatot szeretnénk vezetőnek választani. - Bármelyik folyamat kezdeményezhet vezetőválasztást. Mindegyik folymatnak elküld egy választási üzenetet - Ha Pnagyobb üzenetet kap Pkisebb-től, visszaküld neki egy olan üzenetet, amellyel kiveszi Pkisebb-et a választásból - Ha P megadott időn belül nem kap letiltó üzenetet, ő lesz a vezető. Erről mindegyik másik folyamatot értesíti egy üzenettel.

Vezetőválasztás gyűrűben Logikai gyűrűnk vagy, és a folyamatoknak vannak sorszámai. A legnagyobb sorszámú folyamatot szeretnénk vezetőnek választani.

Bármelyik folyamat kezdeményezhet vezetőválasztást: elindít egy üzenetet gyűrűn körbe, amelyekre mindenki ráírja a sorszámát. Ha egy folyamat összeomlott, az kimarad

Amikor az üzenet visszajut a kezdeményezőhöz, minden aktív folyamat sorszáma szerepel rajta. Ezek közül a legnagyobb lesz a vezető.

Nem okozhat problémát, ha több folyamat is egyszerre kezdeményez választást

Superpeer-választás:

A többi csúcs alacsony késleltetéssel éri el őket

Egyenletesen vannak elosztva a hálózaton

A csúcsok megadott hányadát választjuk superpeer-nek

Egy superpeer korlátozott számú peer-t szolgál ki 7. diasor

Mi az a Conit?

egy olyan adategység amelyre közös feltételrendszer vonatkozik(consisteny unit) - amennyiben megtehetjük hogy konzisztenciafeltételeket az adatok minnél szűkebb körére írjuk fel

Mi az a soros konzisztencia?

elvárjuk hogy a végeredmény olyan legyen , mintha az összes folyamat összes művelete egy meghatározott sorrendben történt volna meg - megőrizve bármely adott folyamat saját műveleteinek sorrendjét

Mi az okozati konzisztencia?

A potenciálisan okozatilag összefüggő műveleteket kell mindegyik folyamatnak azonos sorrendben látnia. - A konkurens írásokat különböző folyamatok különböző sorrendben láthatják.

Milyen megközelítése lehetnek a szinkronizációs változóknak?

Egy rendszerszintű S változó használata - S egy elérése után garantált, hogy a korábbi elérései előtti írások megtörténtek

Változók igénylése/feloldása ⇒ kritikus területek - több rendszerszintű szinkronizációs változó használata - minden adatelemhez külön változó

Adj módszert arra, hogyan válasszuk meg sok klienst kiszolgáló szerverek helyeit (közel) optimálisan nagy hálózatban.

válasszuk meg ugy a szerverek helyét, a kliensektől vett átlagtávolsága minimális legyen. » egzakt kiszámítása költséges, heurisztika szükséges

a K legnagyobb renszerben helyezzünk el egy-egy szervert, mindig a rendszeren belül, leginkább közpoti helyre » szintén magas számítási költség

Keressük meg a K legsűrűbb részt, és oda helyezzünk szervereket(d-dimenziós ábrázolás esetén, távolság = késleltetés) » számítási költsége alacsonyabb

Milyen lehetősége vannak tartalom replikálására?

tartós másolat - a folyamat mindig rendlkezik másolattal (origin server)

szerver által kezdményezett másolat - replikátum kihelyezése egy szerverre, amikor az igényli az adatot

kliens által kezdeményezett másolat - kliensoldali gyorsítótár

Replikátum frissítése során milyen adatokat küldhetünk át a szerverek között?

Kizárólag frissítésről szóló értesítés / érvénytelenítés

Passzív replikáció - adatok átvitele egyik másolatról a másikra

Aktív replikáció - frissítés művelet átvitele

Mi a haszonbérlet?

a szerver ígéretet tesz a kliensnek, hogy elküldi neki a frissítéseket míg a haszonbérlet aktív

Mi a rugalmas haszonbérlet?

rendszer állapotától függhet a haszonbérlet időtartama - kor szerint > minnél réggebben változott az objektum, annál valószínűbb hogy az is marad ⇒ hosszabb lejárat adható - igénylés gyakorisága szerint > minél gyakrabban igényli a kliens az objektumot, annál hosszabb időtartamokra kap haszonbérletet rá - terhelés szerint - minél nagyobb a szerver terhelése, annál rövidebb haszonbérleteket ad ki

Hogyan működik az elsődleges másolaton alapuló protokoll távoli írással?

Példa - a másolatok gyakran egy lokális hálózatra kerülnek - kapcsolat nélküli munka, időnként szinkronizál a rendszerrel

Hogyan működik a testületalapú protokoll? Milyen feltételek kikötésével lehet garantálni a helyességet?

Többszörözött írás - az írási műveletet több szerveren hajtjuk végre.

Testület (quorum): - egy művelet végrehajtása előtt meghatározott számú szervertől kell engedélyt kérni. 8. diasor

Komponens helyessége

elérhetőség ⇒ a komponens reagál a megkeresésre

megbízhatóság ⇒ a komponens biztosítja a szolgáltatást

biztonságosság ⇒ a komponens ritkán romlik el

karbantarthatóság ⇒ az elromlott komponens könnyen javítható

mi a hibajelenség?

a komponens nem tőle a tőle elvártaknak megfelelően üzemel

mi a hiba?

olyan rendszerállapot ami hibajelenséghez vezethet

mi a hibaok?

a hiba (feltételezett) oka

Hibákkal kapcsolatos tennivalók

Megelőzés

Hibatűrés - a komponens legyen képes elfedni a hibát

Mérséklés - lehet mérsékelni a hibák kitejedését, számát , súlyosságát

Előrejelzés - előre becsülhető lehet a hibák száma, következményei

Milyen lehetséges hibaokok vannak?

Összeomlás - a komponens leáll de előtte helyesen működik » Probléma: Nem különbö-
tethető meg hogy összeomlott vagy csak lassú

kiesés - a komponens nem válaszol

időzítési hiba - a komponens helyes választ küld , de túl lassan

válaszhiba - a komponens hibás választ küld > értékhiba > állapotátmeneti hiba

váratlan hiba - véletlenszerű válaszok , véletlenszerű időzítéssel

Milyen csoportak vannak folyamatok esetén?

egyenlő csoport - jó hibatűrés > csoport tagjai között közvetlen az információ csere -
nehéz implementáció > a vezérlés teljesen elosztott

hierarchikus csoport - két fél csak a koordinátoron keresztül kommunikál » Rossz a
hibatűrése és a skálázhatósága - könnyű implementálni

Csoport tagság kezelése

csoportkezelő - egy koordinátor kezeli a csoportot » rosszul skálázódik

csoportkezelők csoportja - a csoportkezelő nem egyetlen szerver > ezt is kezelni kell , viszont ezek a szerverek elég stabilak

csoportkezelő nélkül - a belépő / kilépő folyamat minden csoporttagnak üzenetet küld

Mit kell tudni a csoportkban történő hibaelfedésről?

k-hibatűrő csoport - olyan csoport amely képes elfedni k tag egyszerre történő meghibásodását

folyamatok - a folyamatok azonos ütemben lépnek-e

késleltetések - a kommunikációs késleltetésekre van-e felső korlát

rendezettség - az üzeneteket a feladás sorrendjében kézbesítik-e

átvitel - egyenként (unicast) - többcíműen (multicast)

lehetséges feltételrendszerek a közös végredményhez - Ha az ütemezés szinkron és a késleltetés korlátozott. - Ha az üzeneteket sorrendtartó módon, többcíműen továbbítjuk. - Ha az ütemezés szinkron és a kommunikáció sorrendtartó.

Hogyan észlelhetők a hibák?

időtúllépés észlelése - időkorlát megadása alkalmazásfüggő » nehéz - folyamat / hálózat hiba nem megkülönböztethető - hibákról értesíteni kell a rendszer többi részét > pletykálás > az észlelő komponens is hibaállapotba megy

Milyen hibajelenségek lehetnek RPC esetén?

kliens nem találja szervert $\Rightarrow$ egyszerű kijelezni a kliensnél

a kliens kérése elveszett $\Rightarrow$ a kliens újraküldi a kérést

a szerver összeomlott » nehéz kezelni » a kliens nem tudja mennyire lett feldolgozva a kérés

elveszett a szerver válasza » nehéz felismerni » kliens szemszögéből szerver összeomláshoz hasonlít

összeomlik a kliens » szerver feleslegesen foglal erőforrásokat » árva feladatok ⇒ feladatokra időkorlát ⇒ szerver leállítja/visszagörgeti az árva feladatokat

Hogyan működik a 2PC?

two-phase commit - koordinátor K > számítást kezdeményező kliens - résztvevő R

működés - 1/K > megkérdez mindenkit, enged e commitolni(vote-request) - 1/R > igen(vote-commit) vagy nem(vote-abort) válasz ⇒ utóbbinál eldobja a kiszámított értéket - 2/K > ha minden válasz igen ⇒ global-commit üzenet > különben ⇒ global-abort - 2/R > végrehajtja a globális utasítást

blokkolódhat - ha a koordinátor és legalább egy résztvevő összeomlik - szinte sosem fordul elő gyakorlatban

Hogyan működik a 3PC?

működés - 1/K > vote-request - 1/R > vote-commit vagy vote-abort - 2/K > van vote-abort ⇒ mindenkinek global-abort > minden vote-commit ⇒ mindenkinek prepare-commit - 2/R > prepare -commit esetén ⇒ ready-commit válasz > különben leáll - 3/K > összegyüjti a prepare-commit üzeneteket ⇒ mindenkinek global-commit - 3/R > fogadja global-commit-ot , elmenti az adatokat

nem blokkolódhat - koordinátor és résztvevők legfeljebb 1 távol - ha minden aktív résztvevő READY ⇒ a kiesett érsztvevő nem lehet COMMIT fázisban

mik lehetnek a felépülés irányai?

előrehaladó felépülés - új állapotba hozzuk a rendszert ahonnan működhet

visszatérő felépülés - egy korábbi érvényes állapotra térünk vissza > ehhez ellenőrzőpontokat veszünk fel

Mi a konzisztens metszet?

olyan ellenőrzőpont-gyűjtemény, amelyben minden beérkezett üzenethez a küldési esemény is el van tárolva

Mi a felépülési vonal?

a lehető legkésőbb készült konzisztens metszet

Mi az a dominóeffektus?

vissza akarjuk kresni a legutolsó konzisztens metszetet ⇒ minden folyamatban visszatérés egy korábbi ellenőrzőponthoz » elküldetlen üzenetek további üzeneteket tesznek nem elküldötté

hátha ez is kellde nem biztos (87- jelölések üzenetek naplózására?

HDR[m - m az üzenet fejléce

COPY[m - azok a folyamatok amelekhez HDR[m megérkezett, de még nem tárolták

DEP[m - azok a folyamatok, amelyekhez megérkezett HDR[m' > ahol m' okozatilag függ m től

Mikor stabil egy üzenet?

ha a fejléce már biztosan nem veszeht el

Mi a pesszimista naplózóprotokoll?

ha m nem stabil akkor megköveteljük hogy legfeljebb egy folyamat függjön tőle > |DEP[m| <= 1

Mi az optimista naplózóprotokoll?

tfh C hibás folyamatok halmaza

nem stabil m üzenetekre igaz hogy COPY[m rész C -nek > ekkor egy idő után teljesüljön DEP[m része C -nek is 10. diasor

Távoli elosztott objektumok

Objektum: műveleteket és adatokat zár egységbe (enkapszuláció)

A műveleteket metódusok implementálják, ezeket interfészekbe csoportosítjuk

Az objektumokat csak az interfészükön keresztül érhetik el a kliensek

Az objektumokat objektumszerverek tárolják

A kliensoldali helyettes (proxy) megvalósítja az interfészt

A szerveroldalon a váz kezeli a beérkező kéréseket

Objektumok létrehozása

ideje alpján: - Fordítási időben létrejövő objektumok: A helyettest és a vázat a fordítóprogram készíti el, összeszerkeszti a kliens és a szerver kódjával - Futási időben létrejövő objektumok: Tetszőleges nyelven valósítható meg, de objektumadapterre van szükség a szerveroldalon a használatához

élettartam alapján: - Átmeneti (tranziens) objektum: Élettartama csak addig tart, amíg be van töltve a szerverbe. Ha a szerver kilép, az objektum is megsemmisül - Tartós (perzisztens) objektum: Az objektum állapotát és kódját lemezre írjuk, így a szerver kilépése után is megmarad.

Enterprise Java Beans (EJB)

Az objektumokat alkalmazásszerverek tárolják (pl. GlassFish), amelyek lehetővé teszik az objektumok különböző módokon való elérését

Fajtái: - Statless session bean: Tranziens objektum, egyszer hívják meg, miután elvégezte a feladatát megszűnik. Példa: egy SQL lekérdezés végrehajtása és az eredmény átadása a kliensnek - Stateful session bean: Tranziens objektum, de a klienssel egy munkameneten keresztül tartja a kapcsolatot, ezalatt állapotot is tart fenn. Példa: bevásárlókosár - Entity bean: Perzisztens, állapottal rendelkező objektum, amely több munkamenetet is ki tud szolgálni Példa: olyan objektum, amely az utolsó néhány kapcsolódó kliensről tárol adatokat - Message-driven bean: Különböző fajta üzenetekre reagálni képes objektum. A publish/subscribe kommunikációs modell szerint működik

Globe elosztott objektumok

A Globe rendszerekben az objektumok fizikailag több gépen helyezkednek el: elosztott közös objektum(distributed shared object, DSO).

Objektumszerverek

A rendszer részei a kiszolgálók, a vázak és az adapterek

A kiszolgálót, amely az objektum működését biztosítja, több paradigma szerint lehet implementálni: - Függvények gyűjteménye, amelyik adatbázistáblákat, rekordokat stb. manipulálasznak - Osztályok

A váz szerveroldali hálózati kapcsolatokat kezeli: - Kicsomagolja a beérkező kéréseket, lokálisan meghívja az objektumot, becsomagolja és visszaküldi a választ - Az interfész specifikációja alapján hozzák létre

Az objektumadapter feladata az objektumok egy csoportjának kezelése: - Elsőként fogadja a kéréseket és azonosítja a releváns kiszolgálót - Aktivációs házirend (policy) szerint aktiválja a megfelelő vázat - Az adapter generálja az objektumhivatkozásokat

Távoli metódushívás (RMI)

Tegyük fel, hogy a helyttes és a váz rendelkezésre áll a kliensnél/szervernél - A kliens meghívja helyettest - A helyettes becsomagolja a hívás adatait és elküldi a szervernek - A szerver biztosítja, hogy a hivatkozott objektum aktív - Az objektum váza kicsomagoja a kérést és a metódus meghívodik - Ha paraméterként objektumhivatkozást kaptunk, ezt szintén távoli metódushívással ér el a szerver, ebben a szerver kliensként vesz részt - A választ hasonló úton küldjük vissza, a helyettes kicsomagolja és visszatér vele a klienshez

RMI paraméterátadás

Hivatkozás szerinti paraméterátadás: - A szerver egyszerűen távoli, metódushívással éri el az objektumot - Ha már nincsen szüksége rá, megszünteti a csatolást (unbind)

Érték szerinti paraméterátadás: - Szerializálni kell az objektumot: - Az állapotát - A metódusait, vagy hivatkozását olyan helyre, ahol alérhető az implementációjuk - Amikor a szerver kicsomagolja az objektumot, ezzel másolat készül az eredetiról - Ez az automatikus másolódás többféle problémát okoz Példa: néha túl átlátszó

Konzisztencia

Az objektumok a belépő konzisztencia megvalósításának természetesen adódó eszközei - Az adatok egységbe vannak zárva, és szinkronizációs változóval védjük őket - A szinkronizációs változókat soros konzisztencia szerint érjük el - Az adatokat kezelő műveletek összessége pont az objektum interfésze lesz

Milyen probléma merül fel, ha többszörözött objektumok hívják egymást, és hogyan lehet megoldani?

Replikált objektumok: Nemcsak a kéréseknek kell sorrendben beérkezniük a replikátumokhoz, a vonatkozó szálak ütemezésének determinisztikusnak kell lennie

Replikált hívások: - Aktív replikáció: Ha a replikált objektum hívás során maga is meghív más objektumot, az a kérést többszörözve kapná meg Mo.: mind a szerver-, mint a kliensobjektumon válasszunk koordinátort és csak a koordinátorok küldhessenek kéréseket és válaszokat 11. diasor

Hogyan működik a Network File System (NFS)?

Network File System - elosztott fájlok távoli elérése - alkamazások a hely Virtual File System réteget érik el(távoli elérés átlátszósága)

működés System call layer ⇒ VFS ⇒ RPC client stub ⇒ RPC server stub ⇒ VFS

Cluster alapú fájlrendszerek.

kliens - szerver megközelítés nem elég jó

fájlok felbontésa csíkokra , részek párhuzamos elérése - rendszer gyorsítása - biztonságossá tétele

Példa: Google File System - központ csak azt tárolja , melyik szerver melyik részek felelőse - elsődleges másolaton alapuló protokoll > központot nem terheli

Milyen vonatkozásban jelenik meg az RPC fájlrendszerekben?

Egy lehetőség távoli fájlrendszerek megvalósítására, ha távoli eljáráshívások segítségével végezzük a fájlműveleteket.

Példa - NFSv4 ⇒ támogatja műveletek összekombinálását egy hívásba

Mit lehet mondani a fájlmegosztás szemantikájáról?

Ha egyszerre több kliens is hozzáférhet egy fájlhoz, a konkurens írási és olvasási műveletek lehetséges végrehajtási sorrendjeit és a kijöhető eredmények

Fajtái - Megváltoztathatatlan fájlok > fájlok tartalmát nem lehet módosítani létrehozás után > ritkán alkalmazható - UNIX szemantika > olvasási eredmények mindig a legutolsó írási művelet eredményét adják - Tranzakciós szemantika > a rendszer minden fájlra külön biztosít tranzakciókat - Munkamenet szemantika > ha a kliens megnyitja a fájl , amíg vissza nem írja az írási és olvasási műveletek csak saját maga számára látszanak

Mit lehet tudni a Coda fájlrendszerről

kliensek cache-elhetik a fájlokat » replikált fájlok esetén költséges a kérések sorrendjének kikényszerítése » ha összeomlik egy kliens, hosszu idő után jöhet a következő

munkamenetei tranzakciós szemantikát valósítanak meg - A megnyitott fájl tartalma átmásolódik a kliensre > ha más módosítja a fájlt, arról a kliens értesítést kap. - Ha a kliens csak olvas akkor folytathatja a működését > úgy tekintjük, hogy a tranzakció, amelyet végez, már korábban lezárult.

Mi a kliensoldali gyorsítótárazás célja?

főleg a teljesítmény növelés

szerver kiiértesítő őket ha megvonják ezt a jogot

Mi a szerveroldali gyorsítótárazás célja?

hibatűrés biztosítása

Hogyan növelhető P2P rendszerek rendelkezésre állása?

decentralizált fájlrendszer - probléma lehet ha túl gyorsan változik a tagság » kiléphet akár egy fájlt tartalmazó összes csúcs ⇒ replikálhatjuk fájljainkat (aránya r_rep) ⇒(erasure coding) F fájlt bontsuk m részre , majd minden szerverre tegyünk n részt (n > m) (aráyna r_ec = n/m) > változékony rendszerekben(r_ec < r_rep) 12. diasor

Elosztott webalapú rendszerek

A WWW (World Wide Web) olyan szerverek összessége, amelyek HTTP protokollon keresztül különféle tartalmakat szolgálnak ki A dokumentumokat hiperhivatkozások kapcsolják össze - Sok dokumentum szövegalapú: szövegfálj, HTML, XML - Egyéb fajták: képek, audio, videó, dokumentum - A tartalmak lehetnek a kliensoldalon végrehajthatóak (Javascript)

Webszolgáltatások.

Felmerült az is, hogy felhasználó < - > weboldal interakció mellett az oldalak is igénybe vehetnek szolgáltatásokat más oldalakról – > fontos, hogy a szolgáltatások szabványosak legyenek

Webszerverek.

A szerver szerkezetét a tartalmak kiszolgálásának menete szabja meg. A szerverekbe beépülő modulok telepíthetők, amelyek a kiszolgálás egyes fázisaiban aktivizálódnak

Szerverfürtök

A teljesítmény és a rendelkezésre állás növelésének érdekében a szerverek sokszor többszörözve vannak. A kapcsolattartó szűk keresztmetszetté válhat, ennek elkerülésére több lehetőség van: - TCP átadás: Valamilyen metrika alapján kiválasztunk egy szervert és a kliens kiszolgálását az a szerver folytatja - Taralomérzékeny kéréselosztás: A HTTP kérés tartalmát is figyelmebe vesszük a szerver kiválasztásánál. Ez megnö-

veli a kpacsolattartó terhelését, de sok előnye van: segítségével hatékonyabban lehet a szerveroldali cache-elés, és lehetnek bizonyos feladatokra dedikált szervereink

Webhelyettes A kimenő kapcsolatok kezelésére webhelyetteseket (web proxy) telepíthetünk. Ezek cache-elik a kiszolgált tartalmakat, csak akkor fordulnak a szerverekhez, ha sem náluk, sem a többi helyettesnél nincsen meg a kért tartalom

Replikáció webkiszolgálókban

A replikáció célja a teljesítmény növelése. A rendszer paraméterei változóak lehetnek, ezeket célszerű önszabályozással beállítani

Szerveroldali replikáció.

A tartalomkézbestő hálózatok (CDN) nagy teljesítményű és rendelkezésre állású elosztott rendszerek, amelyeknek célja dokumentumok hatékony kiszolgálása

Replikáció webalkalmazásokban

Ha a CDN tárolt adataiban változás következik be, ez először az eredetszerveren jelenik meg. A változásokat el kell juttatni a CDN szerverekhez, ennek a célszerű módja a rendszer jellegétől függ - Teljes replikáció: sok olvasás, kevés írás, összetett lekérdezések - Részleges replikáció: sok olv., kevés írás, egyszerű lekérdezések - Tartalom szerinti gyorsítótárazás: Az adatbázist az edge szerver módosított, a lekérdezésekhez illeszkedő alakban tárolja helyben, és feliratkozik a szerveren a frissítésekre. Jól működik intervallumokra vonakozó, összetett lekérdezésekre - Eredmények gyorsítótárazása: Az edge szerver a korábbi lekérdezések eredményeit tárolja el. Jól működik egyszerű lekérdezések, amelyek egyedi adatokra vonatkoznak

Ha az írások számaránya megnő, akkor a replikáció akár ronthatja is a rendszer teljesítményét

Hogyan történik a nyitottság implementálása?

- Fontos, hogy a rendszer könnyen cserélhető részekből álljon
- Belső interfészek használata, nem egyetlen monolitikus rendszer
- A rendszerknek minél jobban paraméterezhetőnek kell lennie.
- Egyetlen komponens megváltoztatása/cseréje lehetőleg minél kevésbé hasson a rendszer más részeire.

Melyek az elosztott rendszer céljai?

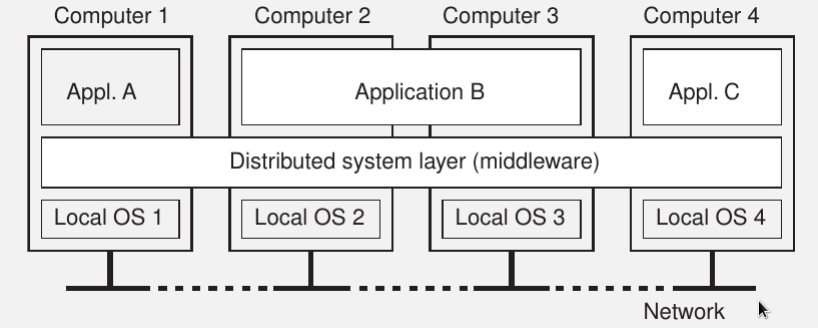
- Távoli erőforrások elérhetővé tétele
- Átlátszóság (distribution transparency)
- Nyitottság (openness)
- Skálázhatóság (scalability)

Mely szempontok alapján épül fel egy elosztott rendszer?

- Független számítógépek
- Egyetlen rendszer $\Rightarrow$ köztes réteg (middleware)

Melyek az elosztott rendszerek fajtái:

- számítási
- információs
- beágyazott/átható



2.2. Kifejtős kérdések

Milyen átfogó célokat fogalmazhatunk meg elosztott rendszerekkel kapcsolatban? Milyen fő szempontok tartoznak a célokhoz? Mennyire megvalósíthatóak a célok, milyen akadályok merülnek fel? Célok:

Távoli erőforrások elérhetővé tétele

Átlátszóság

Nyitottság

Skálázhatóság Átlátszóság: (! a törekvés általában túl erős)

A felhasználók különböző kontinenseken is lehetnek

Meghibásodások elfedése lehetetlen - Összeomlott a szerver vagy lassan válaszol? - Összeomlás előtt feldolgoztat-e az adatot

nagyméretű átlátszóság ⇒ hatékonyság romlása - webes gyorsítótárak tökéletesen frissen tartása - minden azonnal lemezre írása Nyitottság:

a rendszer legyen képes más nyitott rendszerekkel együtt dolgozni - jól definiált interface-k - alkalmazások hordozhatóságának támogatása - könnyen eléhető a rendszerek együttműködése

legyen alkalmazható heterogén(különböző) környezetben - hardvereken - platformokon - programozási nyelveken

implementálása - A rendszer könnyen cserélhető részekből álljon - Belső interface-k használata, nem egyetlen monolitikus rendszer - a rendszernek minnél jobban paraméterezhetőnek kell lennie - egyetlen komponens megváltoztatása/cseréje minnél kevésbé hasson a a rendszer más részeire

probléma - kozisztencia - adatok megbízhatósága Skálázhatóság:

jellege - méret szerint ⇒ több felhasznál és/vagy folyamat (! könnyebben kezelhető például erősebb szerverekkel) - földrajzi ⇒ a rendszert nagyobb területen veszik igénybe - adminisztrációs ⇒ biztonsági, karbantartási, együttműködési kérdések

megvalósítások és problémák - kommunikációs késletetés elfedése > aszinkron kommunikáció » nem minden alkalmazás ültethető át ilyen megközelítésre - elosztás > a számítások egy részét a kliensoldal végzi > decentralizált elnevezési/információs rendszerek - replikáció/cache > replikált fájlszerverek és adatbázisok » inkozisztencia veszélye » globális szinkronizáció szükséges(! rosszul skálázható) > tükrözött weboldalak > fájlok/weboldalak cache-elése

Milyen főbb fajtái vannak az elosztott rendszereknek? Milyen feladatok megoldására alkalmasak, milyen szerveződésűek? Elosztott számítási rendszerek(⇒ számítások végzése nagy teljesítménnyel)

Cluster - lokális hálózatra kapcsolt számítógépek összessége - homogén (ugyanaz az os, hardveresen nem vagy alig térnek el) - a vezérlés központosítva van általában egyetlen gépre

Grid - több gép, kevésbé egységesek - átívelhet több szervezeti egységen - nagyméretű hálózatokra terjedhet ki

Cloud(többrétegű architectúra) - Hardver - Infrastruktúra(⇒ virtuális hardvert tesz elérhetővé) - Platform - Alkalmazás Elosztott információs rendszerek(⇒ adatok kezelése, már meglévő ilyen rendszerek elérése)

A tranzakció adatok összességén végzett művelet az alábbi tulajdonságokkal - atomicity - consistency - isolation - durability

A tranzakciókat esetenként több szerver végzi ezeket egy TP monitor vezérli » Probléma : A TP monitor nem választja el az alkalmazásokat az adatbázisoktól, az alkalmazásoknak egymással is kommunikálniuk kell. Elosztott átható rendszerek(⇒ sok kicsi mobil elemből áll)

A környzet változhat ⇒ a rendszernek ezt követnie kell

Ad hoc szerveződés ⇒ komponensek különbözően használhatók ⇒ könnyen konifurálhatóság szükséges

Megosztott szolgáltatások ⇒ változékobny rendszer az adatoknak könnyen kell áramolni ⇒ egyszerű szerkezetű elemek

Mik a szálak és mik a folyamatok? Hogyan viszonyulnak egymáshoz? Mikor melyiket érdemes alkalmazni? Hogyan jelennek meg kliens-szerver kapcsolatokban? Szál:

a processor egyfajta szoftveres megfelelője, minimális kontextussal

a kontextus elmenthető, és később visszatölthető továbbfuttatáshoz Folyamat:

egy vagy több szálat összefogó nagyobb egységen

egy folyamat szálai közös memóritaterületen dolgoznak

különböző folyamatok nem látják egymás memóriaterületét Kontextus:

Szál - nem sokkal bővebb a processorkontextusnál - szálak közötti váltáshoz nem kell os támogatása

Folyamat - tartalom jórészét az MMU kezeli - folyamatok létrehozása / törlése / váltása költséges Szálak elhelyezkedése:

Folyamaton belül(szálkönyvtár) - előny > minden műveletet egyetlen folyamaton belül kezelünk, ez hatékony - hátrány > minden művelet a gazdafolyamattól » ha a kernel blokkolja a szálat, a folyamat is blokkolódik > ha a kernel nem látja a szálakat, hogy közvetit neki szignálokat?

Folyamaton kívül(kernelszintű szálak) - előny > A szálak blokkolása nem okoz problémát > A szignálokat a kernel a megfelelő szálhoz tudja irányítani - hátrány > Mivel minden művelet a kernelt érinti, ez a hatékonyság rovására megy

Solaris szálakat - könnyűsúlyú folyamat ⇒ kernelszintű szálak , amelyek felhasználói szintű szálkezelőket futtatnak Kliens oldalon:

Példa: többszáló webkliens (⇒ hálózati késé elfedése) - A böngésző letöltött egy oldalt, ami több másik tartalomra hivatkozik. - Mindegyik tartalmat külön szálon tölti le, amíg a HTTP kéréseket kiszolgálják, ezek blokkolódnak. - Amikor egy-egy fájl megérkezik, a blokkolás megszűnik, és a böngésző megjeleníti a tartalmat. Szerver oldalon:

Cél: hatékonyság növelése - Szálak olcsóbbak mint a folyamatok - többprocesszoros rendszerek kapacitását csak többszálú szerverek képesek kihasználni - hálózat késleltetését lehet elfedni

Cél: program szerkezetének javítása - A program jobban kezelhető lehet, ha sok egyszerű, blokkoló hívást alkalmaz, mint más szerkezet esetén. (» némi teljesítményvesztés) - kisebbek és könnyebben érthetőek

Hasonlítsd össze távoli szolgáltatások igénybe vételének különböző modelljeit: kliens-szerver, RPC, RMI, MOM. Kliens-szerver

általános jellemzők - jellemzően szinkron kommunikáció - kliensnek és szervernek egyidőben kell aktívnak lennie - a kliens blokkolódik amíg a válasz meg nem érkezik - a szerver csak a kliensek fogadásával és a kérések kiszolgálásával foglalkozik

szinkron kommunikáció hátrányai - a kliens nem dolgozik amíg a válaszra vár - a hibákat rögtön kezelni kell vagy feltartjuk a klienst - bizonyos feladatokhoz(levelezés) nem jól illeszkedik RPC(Remote Procedure Call)

alapötlet - alprogramok használata természetes fejlesztés során - az alprogramok jó esetben egymástól függetlenül működnek - ... így akár távoli gépen is végrehajthatóak

hálózati kommunikációra van szükség (⇒ ezt eljáráshívási mechanizmus fedi el)

lépései ⇒ lásd

paraméterek sorosítása - kliens-szerver eltérhet az adatábrázolásban ⇒ serializálás(⇒ közös bájtsorozat) > a két csonknak fordítania kell a közös formátumról a gépeik formátumára

paraméterátadás szemantikája - Érték–eredmény szerinti paraméterátadási szemantika(<= mivel a hivatkozások csak az egyik oldalon látszanak) - Minden feldolgozandó adat paraméterként kerül átadásra

távoli hivatkozás - távoli adat egységesen elérhető - paraméterként átadhatóak

speciális megvalósítások - aszinkron RPC > a szerver nyugtázza az üzenet megérkezését, választ nem vár - késleltetett szinkronizált RPC > két aszinkron RPC egymással összehangolva - további > a kliens elküldheti a kérését majd időnként lekérdezheti a szervertől kész-e már a válasz.

kliens csatlakozása - a szolgáltatásokat katalógusba jegyzik, hogy melyik gépen érhetők el > globálisan és lokálisan is - a kliens kikeresi a szolgáltatást a katalógusból - a kliens végpontot igényel a démontól a kommunikációhoz Üzenet orientált köztes réteg (MOM)

tulajdonságok - aszinkron kommunikációs architektúra - folyamatok üzeneteket küldhetnek egymásnak - a küldő félnek nem kell válaszra várni, foglalkozhat mással - gyakran hibatűrést biztosít

működési elv - a köztes réteg várakozási sorokat tart fenn a rendszer gépein(queue) - műveletek > PUT ⇒ üzenetet tesz a várakozási sor végére > GET ⇒ blokkol amíg a sor üres, majd leveszi az első üzenetet > POLL ⇒ nem blokkolva, lekérdezi van-e üzenet, ha igen, leveszi az elsőt > Notify ⇒ kezelőrutint hozzáadása a várakozási sorhoz , amely minden érkező üzenetre meghívódik

Üzenetsor kezelő rendszer homogenitás - feltételezzük hogy a rendszer minden eleme közös protokollt használ ⇒ az üzenetek szerkezete és adat ábrázolása megegyező

Üzenet közvetítő - olyan központi komponens, amely heterogén környezetben gondoskodik a megfelelő koverziókról > átalakítja az üzeneteket a megfelelő formátumra > szerepe szerint gyakran átjáró(application-level gateway,proxy) ⇒ biztonsági funkciókat is ellát > az üzenetek tartalmát is megvizsgálhatja az útválasztáshoz (Enterprise Application Integration)

Milyen módokon lehet információt elterjeszteni a rendszerben? (multicast, járvány) Alkalmazásszintű multicasting

multicast - a hálozat minden csúcsának üzenetküldés ⇒ hierarchikus overlay hálózat kell

Chord struktúrában tárolt fa - multicast hálózatunkhoz generálunk egy azonosítót ⇒ több hálózat is lehet a rendszerben - tfh. az azonosító egyértelműen kijelöl egy csúcsot a

rendszerben ⇒ gyökere - terv > a küldendő üzeneteket a gyökérnek küldik ⇒ onnan terjed lefele - csatlakozás a multicast hálózathoz > P csúcs csatlkaozási kérést küld gyökér fel > P csúcstól gyökérek egyértelmű útvonal ⇒ fa részévé tesszük ⇒ P elérhetővé válik a gyökérből

költségek - kapcsolatok terhelése > mivel overlay hálózat, előfordulhat hogy többször igénybeveszi ugyanazt a fizikai kapcsolatot - Stretch > az overlay-t követő és az alacsonyszintű üzenetküldés kölcségének hányadosa Járványalapó algoritmusok

alapötlet - valamelyik szerveren frissítési művelet történt ⇒ szeretnénk hogy edlterjedjen a rendszerben minden szervehez - minden szerver elküldi a változtatást néhány szomszédjának lusta módon - tfh, nincs olvasás-írás konfliktus

két alkategória - Anti-entrópia > Minden szerver rendszeresen kiválaszt egy másikat > kicserélik egymás között a változásokat. - Pletykálás (gossiping) > Az újonnan frissült (megfertőzött) szerver elküldi a frissítést néhány szomszédjának (megfertőzi őket).

anti-entrópia - frissítések cseréje > P csúcs Q csúcsot választotta ki > küldés ⇒ P elküldi a nála lévő frisstéseket Q nak > rendelés ⇒ P bekéri a Q nál lévő frissítéseket > küldés-rendelés ⇒ P és Q kicseréli az adatokat - hatékonyság > küldő-rendelő megközelítés esetén o(log(n)) nagyságrendű forduló után végbemegy a terjesztés > egy forduló ha minden csúcs megtett egy lépést

pletykálás - működési elv > ha S szerver frisstést észlel > felveszi a kapcsolatot más szerverekkel > elküldi számukra a frissítést > ha olyan szerverre kapcsolodik , ahol már jelen van a frissítés 1/k valószínűséggel abbahagyja a terjesztést -hatékonyság - kellően sok szerver esetén exponenciálisan csökken a a tudatlanságban lévő szerverek száma » Probléma viszont az hogy nem garantálható hogy minden szerverhez eljut a frissítés

Értékek törlése

a törlési művelet nem terjeszthető » a még terjedő frissítések újra létrehoznák az adatot

megoldás - speciális frissítés : hallotti bizonyítvány - egy halotti bizonyítvány törölhető > szemétgyüjtő jellegű megközelítés ⇒ globális ellenőrzése annak hogy mindenhova eljutott > elavuló bizonyítvány ⇒ kibocsátás után adott idővel elavul » probléma: nem garantálható hogy mindenhova elér

Hasonlítsd össze a tanult elnevezési rendszereket. Elnevezési rendszerek

elosztott rendszerek entitásai kapcsolódási pontokon keresztül érhetők el - távolról címük azonosítja - célszerű őket kapcsolodási pontjaiktól függetlenül is elnevezni > egyszerű nevekknek nincs szerkezete > tartalmuk véletlen szöveg > csak összehasonlításra használhatóak - azonosító > egy-egy kapcsolat > maradandó hozzárendelés ⇒ a név nem hivatkozhat másra később sem Egyszerű megoldások

Broadcasting - kihirdetjük az azonosítót a hálózaton ⇒ az egyed visszaküldi a jelenlegi címét > lokális hálózatokon túl nem skálázódik > a hálózaton minden gépnek figyelni kell a beérkező kérésre

Továbbítómutató - amikor az egyed elköltözik , egy mutató marad utána az új helyre > a kliens elől rejtve van > a megtalált cím visszaküldhető ⇒ további feloldások gyorsabbak - problémák > hosszú láncok nem hibatűrőek > feloldás időigényes > láncok röviditésére külön mechanizmus Otthonalapú megközelítések

Egyrétegű rendszer - egy egyedhez tartozik otthon > ez tartja számon a jelenlegi címet - az otthoni cím be van jegyezve egy névszolgáltatásba - a kliens az otthonhoz kapcsolódik, onnan kapja meg az aktuális címet

Kétrétegű rendszer - feljegyezzük a környéken lévő egyedeket - a névfeloldás ezt a jegyzéket vizsgálja először > ha a keresett egyed nincs a környéken ⇒ otthonhoz kapcsolódás

Problémák » legalább az egyed élettartamán fenn kell tartani az otthont » az otthon helye rögzített » költséges az egyed költözése » rossz földrajzi skálázódás » az egyed közelebb lehet a klienshez az otthonnál

Milyen kérdések merülhetnek fel elosztott rendszer konzisztenciájával kapcsolatban? Milyen algoritmusok/módszerek alkalmazhatók a konzisztencia biztosítására? Kliensközpontú konzisztencia

Probléma: (A szerver után B szerverre csatlakozva) - A-ra feltöltött frissítések még lehet nem jutottak el B-hez - B-n lehet, hogy újabb adatok vannak mint A-n - B -re töltött frissítések ütközhetnek A-ra feltöltöttekkel

Cél - az A szerveren kezelt adatok ugyanolyan állapotban legyenek látható B-n ⇒ ekkor az adatbázis konzisztens a kliens számára

Monoton olvasás - ha a kliens kiolvasott egy értéket x-ből , minden ezután következő olvasás ezt adja, vagy ennél frissebbet

Monoton írás - a kliens csak akkor írhatja x-et , ha a kliens korábbi írásai x-re már befejeződtek.

Olvasd az írásodat - ha a kliens olvassa x-et , a saját legutolsó írásának eredményét kapja, vagy frissebbet

Írás olvasás után - Ha a kliens kiolvasott egy értéket x-ből, minden ezután kiadott frissítési művelete x legalább ennyire friss értékét módosítja.