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.

2. Kérdések és válaszok

2.1. Rövid kérdések

1. 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

2. Mi jellemzi a nyitott rendszereket?

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

3. 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

4. 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
- Adminisztrációs: Biztonsági, karbantartási, együttműködési kérdések

5. Ismertess egy technikát, amelynek célja az átméretezhetőség megvalósítása.
   - A kommunikációs késleltetés elfedése
     - Aszinkron kommunikáció használata
     - A beérkező választ külön kezelő dolgozza fel
     - Probléma » nem minden alkalmazás ültethető át ilyen formában
   - elosztás
     - a számítások egy részét a kliensoldal végzi
     - decentralizált elnevezési/információs rendszerek
   - replikáció/cache
     - replikált fájlszerverek és adatbázisok » inkonzisztencia veszélye » globális szinkronizáció szükséges(Rosszul skálázható!)
     - tükrözött weboldalak
     - fájlok/weboldalak cache-elés
6. Miben hasonlítanak és miben térnek el a cluster és a grid rendszerek?
7. mindkettő elosztott számítási rendszer fajta.
8. különbségeik: Cluster <————————————> Grid ———————————— lokális hálózat | nagyméretű hálózat központosított vezérlés | átívelhet több szervezeti egységen homogén számítógépek | több, kevésbé egységes számítógép
9. Mi az ACID? (Legalább két szempont részletezve, a másik kettőnek legalább a neve.)
10. 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.
11. Adj meg olyan feltételezést elosztott rendszerrel kapcsolatban, amellyel kényelmes élni, de a valóságban akadályokat gördíthet elénk.

12. a hálózat hibamentes

13. a hálózat biztonságos

14. a hálózat homogén

15. a hálózati topológia nem változik

16. a kommunikációnak nincsen időigénye

17. a sávszélesség korlátlan

18. a kommunikációnak nincsen költsége

19. csak egy adminisztrátor van 2. diasor

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

21. a kliens kérést küld a szervernek ⇒ a szerver fogadja a kérés, válaszol | eközben a kliens várakozik ⇒ ⇒ a kliens megkapja a választ , folytatja a működést

22. így veszi igénybe a kliens a szolgáltatást

23. Milyen három rétegbe szokás osztani az alkalmazásokat?

24. Megjelenítés - felhasználó felületet alkotó komponensek (view)

25. Üzleti logika - alkalmazás működését írja le konkrét adatok nélkül (controller)

26. Perzisztencia - adatok tartós tárolása (model)

27. Mi az "overlay"?

28. A gráfban szomszédos csúcsok a fizikai hálózaton lehetnek távol egymástól, a rendszer elfedi, hogy a köztük levő kommunikáció több gépet érintve történik.

29. legtöbb P2P rendszer alapja

30. Milyen az overlay peer-to-peer hálózatok felépítése?

31. Nem találtam diában

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

33. A csúcsokat valamilyen struktúra szerint overlay hálózatba szervezzük(pl̇logikai gyűrű), és a csúcsoktól az azonosítójuk alapján lehet szolgáltatásokat igénybe venni.

34. Példa: elosztott hasítótábla 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.

35. Hogy működik a struktúrált peer-to-peer pletykálás?

36. selectPeer: - A részleges nézetből kiválaszt egy szomszédot.

37. selectToSend: - Az általa ismert szomszédok közül kiválaszt n darabot.

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

39. Hogyan működnek a strukturálatlan peer-to-peer rendszerek?

40. ezek a rendszerek igyekeznek véletlen gráfstruktúrátfenntartani.

41. Mindegyik csúcsnak csak részleges nézete van a gráfról

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

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

44. olyan kisszámú csúcs, amelyeknek külön feladata van - kereséshez index fenntartása - a hálózat állapotának felügyelete - csúcsok közötti kapcsolatok létrehozása

45. Példa: nem találtam diában

46. Mi az interceptor?

47. Távoli objektum elérése során a vezérlés szokásos menetébe avatkozik bele - pl. átalakíthatja más formátumra a kérést

48. Jellemzően az architektúra rétegei közé illeszthető.

49. Milyen az önszervező rendszerek általános architektúrája?

50. elvárható tulajdonságok - önkonfiguráló - önkezelő - öngyógyító - ön optimalizáló - ön*

51. Mi az az edge server?

52. az adatokat tároló szerver

53. a kliensekhez minél közelebb van elhelyezve

54. ahol egy nagyobb hálózat az Internetre csatlakozik

55. Mi az a Content Delivery Network?

56. a tartalom szolgáltatás hatékonyságát növelik és költségét csökkentik.

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

58. visszacsatolásos modell - mérik, hogy a rendszer mennyire tér el a kívánt tulajdonságoktól, és szükség szerint változtatnak a beállításokon. ⇒ Például Globule

59. 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

60. Mi a kontextusváltás?

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

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

63. Process VM - a virtuális program közönséges programként fut ⇒ bytecode-ot hajt végre - példa : JVM , CLR

64. VM Monitor - Hardver teljes körű virtualizációja ⇒ bármely operációs rendszer futtatására alkalmas - pl VirtualBox

65. Mi a szuperszerver?

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

67. Mi a iteratív és konkurens szerver?

68. iteratív ⇒ egyszerre csak egy kapcsolatot tud kezelni

69. konkurens ⇒ párhuzamosan több kapcsolatot is tud kezelni

70. Mi jellemzi az állapotteljes szervereket?

71. Állapotot tart számon a klienstől - Megjegyzi, melyik fájlokat használta a kliens, és ezeket előre megtudja nyitni legközelebb - Megjegyzi, milyen adatokat töltött le a kliens, és frissítéseket küldhet neki

72. Mi a kódmigráció?

73. olyan kommunikáció, amely során nem csak adatokat küldünk át

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

75. Client-Server - a szokásos kliens-szerver kommunikáció, nincsen kódmigráció

76. Remote Evaluation - a kliens feltölti a kódot, és a szerveren futtatja

77. Code on Demand - a kliens letölti a kódot a szerverről, és helyben futtatja

78. Mobile Agent - a mobil ágens feltölti a kódját és az állapotát, és a szerveren folytatja a futását

79. Mik az objektum komponensek?

80. Kódszegmens ⇒ a programkódot tartalmazza

81. Adatszegmens ⇒ a futó program állapotát tartalmazza

82. Végrehajtási ⇒ szegmens: a futtató szál környezetét tartalmazza

83. Mi a gyenge mobilitás?

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

85. Mi az erős mobilitás?

86. 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

87. Mi az ISO/OSI modell alsó három rétegének feladata?

88. Fizikai réteg - a bitek átvitelének fizikai részei írja lefutása

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

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

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

92. TCP ⇒ kapcsolatalapú, megbízható, sorrendhelyes átvitelének

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

94. Mi a köztes réteg?

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

96. Mi jellemzi az időleges, szinkron kommunikációt?

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

98. Mi jellemzi a megtartó, aszinkron kommunikációt?

99. A kommunikációs rendszer hajlandó huzamosan tárolni az üzenetet.

100. 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.

101. Mit tud az RPC paraméter átadásról?

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

103. Érték–eredmény szerinti paraméter átadási szemantika

104. Hogyan kezelhetőek a hivatkozások RPC hívás során?

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

106. Milyen lépésekből áll a socket kommunikáció? (Sorrenddel.)

107. létrejön a socket

108. csatlakozik a szerverhez(⇒ a szerver fogadja a csatlakozást)

109. a socket ír a szerverre (⇒ szerver fogadja majd válaszol)

110. a socket fogadja a választ

111. az előző két lépés tetszőleges számban ismétlődik

112. a socket zárja a kapcsolatot

113. Adatátviteli módok folyamatos média esetén

114. aszinkron - nem ad megkötést hogy mikor kell átvinni az adatot

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

116. izokron - alsó és felső korlátot is ad a csomagok átvitelére

117. Mi a folyam? (Fontosabb jellemzőkkel.)

118. adatfolyam - izokron kommunikációt támogató kommunikációs forma

119. egyirányú

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

121. egyszerű folyam - egyfajta adatot továbbító

122. összetett folyam - többfajta adatot továbbít

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

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

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

126. Mi az a QoS?

127. Quality of Service - a folyamokkal kapcsolatban vázolt követelmények

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

129. 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

130. Mi a broadcasting?

131. 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

132. Mi a továbbító mutató?

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

134. Mi a Chord elosztott hasítótábla?

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

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

137. 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)

138. Mit tartalmaz a Chord adatábrázolás egy finger table-je?

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

140. 2$\hat{}$(i-) - bináris jellegű keresés ⇒ minden lépés felezi a keresési tartományt > 2$\hat{}$(m-, 2$\hat{}$(m-,. . . ,1

141. Mi az a HLS?

142. Hierarchical Location Service

143. A hálózatokat osszuk tartományokra - tartozzon minden tartományhoz egy katalógus

144. építsünk hierarchiát a katalógusokból

145. Mik a katalógus csúcsok?

146. az E egyed címe egy levélben található

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

148. a gyökér minden út kiindulópontja - minden egyedről van információja

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

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

151. szervezeti szint ⇒ egy egy szervezet által kezelt csúcsok szintje

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

153. ————-

154. ————

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

156. Hogyan működik az iteratív névfeloldás?

157. gyökér névszerverek egyikétől indítjuk

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

159. Hogyan működik a rekurzív névfeloldás?

160. a névszerverek egymás között kommunkálva oldják fel a nevet - a kliensoldali névfeloldóhoz rögtön a válasz érkezik

161. Mit mondhatunk a névfeloldás átméretezhetőségéről?

162. 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

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

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

165. Mi az X.500?

166. A katalógusszolgáltatásokban az attribútumokra érvényes megkötések egyfajta szabványa

167. LDAP protokollon szokás elérni

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

169. Hogyan működik a Cristian-algoritmus?

170. 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

171. Hogyan működik a Berkeley-algoritmus?

172. Itt nem feltétlenül a pontos idő beállítása a cél, csak az, hogy minden gép ideje azonos legyen. - álagot von minden gép idejéből ⇒ 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.

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

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

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

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

177. Az idő és az előbb történt reláció kapcsolata?

178. 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)

179. Mi a Lamport-féle időbélyeg?

180. 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)

181. 1 lesz így az idő nem folyik visszafelé - Pi és Pj időbélyegei közül tekintsük a Pi-belit elsőnek, ha i $<$ j

182. Mi az időbélyeg-vektor?

183. 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

184. Az m üzenet elküldése során Pi megnöveli eggyel VCi[i értékét, és a teljes Vi időbélyeg-vektort ráírja az üzenetre

185. 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

186. Pontosan sorbarendezett csoportcímzés definíciója?

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

188. Kölcsönös kizárás feladata?

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

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

191. Ismertesd a központosított kölcsönös kizárási algoritmust.

192. 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.

193. Elosztott kölcsönös kizárás működési elve?

194. 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

195. Ismertesd a zsetongyűrű alapú kölcsönös kizárási algoritmust.

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

197. 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.

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

199. 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.

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

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

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

203. Nem okozhat problémát, ha több folyamat is egyszerre kezdeményez választást

204. Superpeer-választás:

205. A többi csúcs alacsony késleltetéssel éri el őket

206. Egyenletesen vannak elosztva a hálózaton

207. A csúcsok megadott hányadát választjuk superpeer-nek

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

209. Mi az a Conit?

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

211. Mi az a soros konzisztencia?

212. 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

213. Mi az okozati konzisztencia?

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

215. Milyen megközelítése lehetnek a szinkronizációs változóknak?

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

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

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

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

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

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

222. Milyen lehetősége vannak tartalom replikálására?

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

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

225. kliens által kezdeményezett másolat - kliensoldali gyorsítótár

226. Replikátum frissítése során milyen adatokat küldhetünk át a szerverek között?

227. Kizárólag frissítésről szóló értesítés / érvénytelenítés

228. Passzív replikáció - adatok átvitele egyik másolatról a másikra

229. Aktív replikáció - frissítés művelet átvitele

230. Mi a haszonbérlet?

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

232. Mi a rugalmas haszonbérlet?

233. rendszer állapotától függhet a haszonbérlet időtartama - kor szerint $>$ minnél réggebben változott az objektum, annál valószínűbb hogy az is marad $\Rightarrow$ 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

234. Hogyan működik az elsődleges másolaton alapuló protokoll távoli írással?

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

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

237. Többszörözött írás - az írási műveletet több szerveren hajtjuk végre.

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

239. Komponens helyessége

240. elérhetőség $\Rightarrow$ a komponens reagál a megkeresésre

241. megbízhatóság $\Rightarrow$ a komponens biztosítja a szolgáltatást

242. biztonságosság $\Rightarrow$ a komponens ritkán romlik el

243. karbantarthatóság $\Rightarrow$ az elromlott komponens könnyen javítható

244. mi a hibajelenség?

245. a komponens nem tőle a tőle elvártaknak megfelelően üzemel

246. mi a hiba?

247. olyan rendszerállapot ami hibajelenséghez vezethet

248. mi a hibaok?

249. a hiba (feltételezett) oka

250. Hibákkal kapcsolatos tennivalók

251. Megelőzés

252. Hibatűrés - a komponens legyen képes elfedni a hibát

253. Mérséklés - lehet mérsékelni a hibák kitejedését, számát , súlyosságát

254. Előrejelzés - előre becsülhető lehet a hibák száma, következményei

255. Milyen lehetséges hibaokok vannak?

256. Összeomlás - a komponens leáll de előtte helyesen működik » Probléma: Nem különbötethető meg hogy összeomlott vagy csak lassú

257. kiesés - a komponens nem válaszol

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

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

260. váratlan hiba - véletlenszerű válaszok , véletlenszerű időzítéssel

261. Milyen csoportak vannak folyamatok esetén?

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

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

264. Csoport tagság kezelése

265. csoportkezelő - egy koordinátor kezeli a csoportot » rosszul skálázódik

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

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

268. Mit kell tudni a csoportkban történő hibaelfedésről?

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

270. folyamatok - a folyamatok azonos ütemben lépnek-e

271. késleltetések - a kommunikációs késleltetésekre van-e felső korlát

272. rendezettség - az üzeneteket a feladás sorrendjében kézbesítik-e

273. átvitel - egyenként (unicast) - többcíműen (multicast)

274. 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.

275. Hogyan észlelhetők a hibák?

276. 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

277. Milyen hibajelenségek lehetnek RPC esetén?

278. kliens nem találja szervert ⇒ egyszerű kijelezni a kliensnél

279. a kliens kérése elveszett ⇒ a kliens újraküldi a kérést

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

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

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

283. Hogyan működik a 2PC?

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

285. 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

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

287. Hogyan működik a 3PC?

288. 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

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

290. mik lehetnek a felépülés irányai?

291. előrehaladó felépülés - új állapotba hozzuk a rendszert ahonnan működhet

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

293. Mi a konzisztens metszet?

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

295. Mi a felépülési vonal?

296. a lehető legkésőbb készült konzisztens metszet

297. Mi az a dominóeffektus?

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

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

300. HDR[m - m az üzenet fejléce

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

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

303. Mikor stabil egy üzenet?

304. ha a fejléce már biztosan nem veszeht el

305. Mi a pesszimista naplózóprotokoll?

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

307. Mi az optimista naplózóprotokoll?

308. tfh C hibás folyamatok halmaza

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

310. Távoli elosztott objektumok

311. Objektum: műveleteket és adatokat zár egységbe (enkapszuláció)

312. A műveleteket metódusok implementálják, ezeket interfészekbe csoportosítjuk

313. Az objektumokat csak az interfészükön keresztül érhetik el a kliensek

314. Az objektumokat objektumszerverek tárolják

315. A kliensoldali helyettes (proxy) megvalósítja az interfészt

316. A szerveroldalon a váz kezeli a beérkező kéréseket

317. Objektumok létrehozása

318. 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

319. 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.

320. Enterprise Java Beans (EJB)

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

322. 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

323. Globe elosztott objektumok

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

325. Objektumszerverek

326. A rendszer részei a kiszolgálók, a vázak és az adapterek

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

328. 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

329. 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

330. Távoli metódushívás (RMI)

331. 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

332. RMI paraméterátadás

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

334. 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

335. Konzisztencia

336. 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

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

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

339. 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

340. Hogyan működik a Network File System (NFS)?

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

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

343. Cluster alapú fájlrendszerek.

344. kliens - szerver megközelítés nem elég jó

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

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

347. Milyen vonatkozásban jelenik meg az RPC fájlrendszerekben?

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

349. Példa - NFSv4 ⇒ támogatja műveletek összekombinálását egy hívásba

350. Mit lehet mondani a fájlmegosztás szemantikájáról?

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

352. 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

353. Mit lehet tudni a Coda fájlrendszerről

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

355. 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.

356. Mi a kliensoldali gyorsítótárazás célja?

357. főleg a teljesítmény növelés

358. szerver kiiértesítő őket ha megvonják ezt a jogot

359. Mi a szerveroldali gyorsítótárazás célja?

360. hibatűrés biztosítása

361. Hogyan növelhető P2P rendszerek rendelkezésre állása?

362. 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

363. Elosztott webalapú rendszerek

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

365. Webszolgáltatások.

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

367. Webszerverek.

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

369. Szerverfürtök

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

371. 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

372. Replikáció webkiszolgálókban

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

374. Szerveroldali replikáció.

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

376. Replikáció webalkalmazásokban

377. 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, öszetett 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

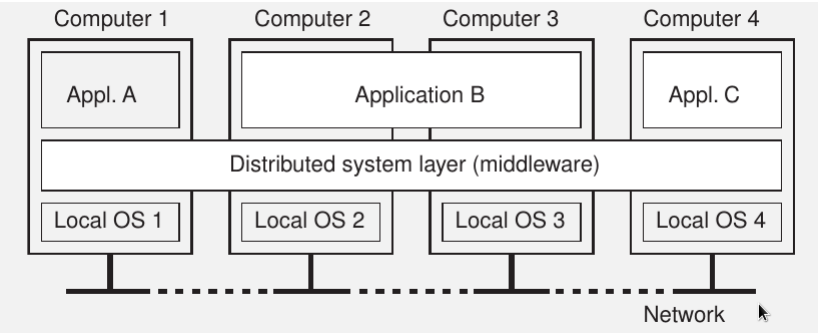
378. Ha az írások számaránya megnő, akkor a replikáció akár ronthatja is a rendszer teljesítményét

379. Melyek az elosztott rendszer céljai?

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

380. Mely szempontok alapján épül fel egy elosztott rendszer?

- Független számítógépek
- Egyetlen rendszer ⇒ köztes réteg (middleware)



2.2. Kifejtős kérdések

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

2. Távoli erőforrások elérhetővé tétele

3. Átlátszóság

4. Nyitottság

5. Skálázhatóság Átlátszóság: (! a törekvés általában túl erős)

6. A felhasználók különböző kontinenseken is lehetnek

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

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

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

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

11. 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

12. probléma - kozisztencia - adatok megbízhatósága Skálázhatóság:

13. 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

14. 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

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

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

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

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

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

20. 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)

21. A környzet változhat ⇒ a rendszernek ezt követnie kell

22. Ad hoc szerveződés ⇒ komponensek különbözően használhatók ⇒ könnyen konifurálhatóság szükséges

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

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

25. a processor egyfajta szoftveres megfelelője, minimális kontextussal

26. a kontextus elmenthető, és később visszatölthető továbbfuttatáshoz Folyamat:

27. egy vagy több szálat összefogó nagyobb egységen

28. egy folyamat szálai közös memóritaterületen dolgoznak

29. különböző folyamatok nem látják egymás memóriaterületét Kontextus:

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

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

32. 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?

33. 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

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

35. 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:

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

37. 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

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

39. 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

40. 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)

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

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

43. lépései ⇒ lásd

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

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

46. távoli hivatkozás - távoli adat egységesen elérhető - paraméterként átadhatóak

47. 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.

48. 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)

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

50. 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

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

52. 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)

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

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

55. 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

56. 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

57. 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

58. 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).

59. 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

60. 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

61. Értékek törlése

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

63. 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

64. Hasonlítsd össze a tanult elnevezési rendszereket. Elnevezési rendszerek

65. 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

66. 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

67. 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

68. 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

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

70. 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

71. Milyen kérdések merülhetnek fel elosztott rendszer konzisztenciájával kapcsolatban? Milyen algoritmusok/módszerek alkalmazhatók a konzisztencia biztosítására? Kliens-központú konzisztencia

72. 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

73. Cél - az A szerveren kezelt adatok ugyanolyan állapotban legyenek látható B-n $\Rightarrow$ ekkor az adatbázis konzisztens a kliens számára

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

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

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

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