Elosztott rendszerek: Alapelvek és paradigmák
Distributed Systems: Principles and Paradigms

Maarten van Steen[1] Kitlei Róbert [2]

[1]VU Amsterdam, Dept. Computer Science
[2]ELTE Informatikai Kar

6. rész: Szinkronizáció

2015. május 24.

Tartalomjegyzék

Fizikai órák

Milyen módon van szükségünk az időre?

Néha a pontos időt szeretnénk tudni, néha elég, ha megállapítható két időpont közül, melyik volt korábban. Foglalkozzunk először az első kérdéssel.

Egyezményes koordinált világidő

Az időegységeket (pl. másodperc) az atomidőből (TAI) származtatjuk.

- Az atomidő definíciója a gerjesztett céziumatom által kibocsátott sugárzás frekvenciáján alapul.
- A Föld forgásának sebessége kissé változékony, ezért a világidő (UTC) néhány (szökő)másodperccel eltér az atomidőtől.
- Az atomidő ☞kb. 420 atomóra átlagából adódik.
- Az atomórák pontosságának nagyságrendje kb. $1 ns/nap$.
- Az atomidőt műholdak sugározzák, a vétel pontossága 0.5 ms nagyságrendű, pl. az időjárás befolyásolhatja.

Fizikai órák

Fizikai idő elterjesztése

Ha a rendszerünkben van UTC-vevő, az megkapja a pontos időt. Ezt a következők figyelembe vételével terjeszthetjük el a rendszeren belül.

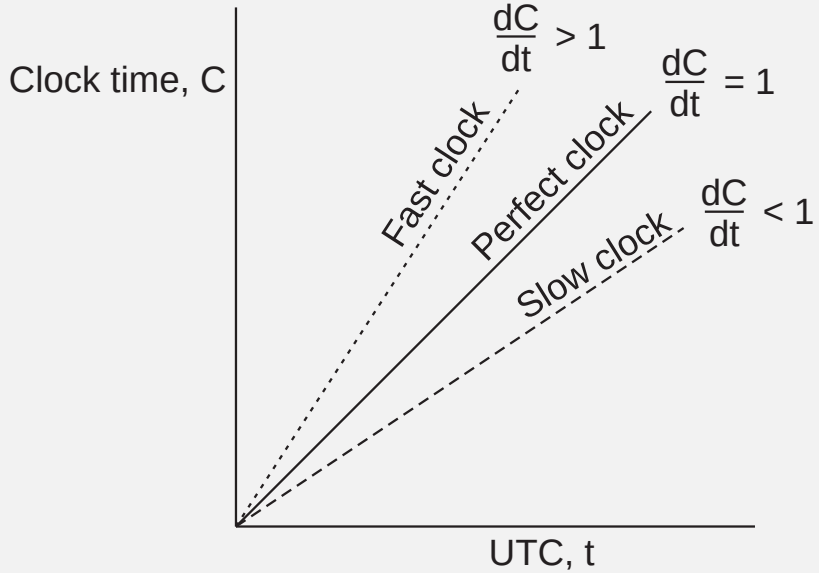
- A p gép saját órája szerint az idő t UTC-időpillanatban $C_p(t)$.
- Ideális esetben mindig pontos az idő: $C_p(t) = t$, másképpen $dC/dt = 1$.

Időszinkronizáció üteme

A valóságban p vagy túl gyors, vagy túl lassú, de viszonylag pontos:

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$$

Ha csak megadott δ eltérést akarunk megengedni, $\delta/(2\rho)$ másodpercenként szinkronizálnunk kell az időt.



Óraszinkronizálás

Cristian-algoritmus

Mindegyik gép egy központi időszervertől kéri le a pontos időt legfeljebb $\delta/(2\rho)$ másodpercenként (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.

Berkeley-algoritmus

Itt nem feltétlenül a **pontos** idő beállítása a cél, csak az, hogy minden gép ideje azonos legyen.
Az időszerver néha bekéri mindegyik gép idejét, ebből átlagot von, 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.

Az előbb-történt reláció

Az előbb-történt (happened-before) reláció

Az előbb-történt reláció az alábbi tulajdonságokkal rendelkező reláció. Annak a jelölése, hogy az a esemény előbb-történt-mint b-t: $a \rightarrow b$.

- Ha ugyanabban a folyamatban az a esemény korábban következett be b eseménynél, akkor $a \rightarrow b$.
- Ha a esemény egy üzenet küldése, és b esemény annak fogadása, akkor $a \rightarrow b$.
- A reláció tranzitív: ha $a \rightarrow b$ és $b \rightarrow c$, akkor $a \rightarrow c$

Parcialitás

A fenti reláció parciális rendezés: előfordulhat, hogy két esemény közül egyik sem előzi meg a másikat.

Logikai órák

Az idő és az előbb-történt reláció

Minden e eseményhez időbélyeget rendelünk, ami egy egész szám (jelölése: $C(e)$), és megköveteljük az alábbi tulajdonságokat.

- Ha $a \to b$ egy folyamat két eseményére, akkor $C(a) < C(b)$.
- Ha a esemény egy üzenet küldése és b esemény annak fogadása, akkor $C(a) < C(b)$.

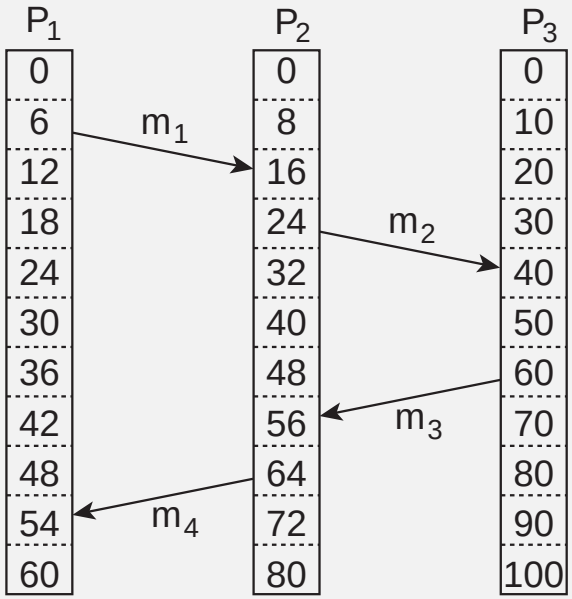
Globális óra nélkül?

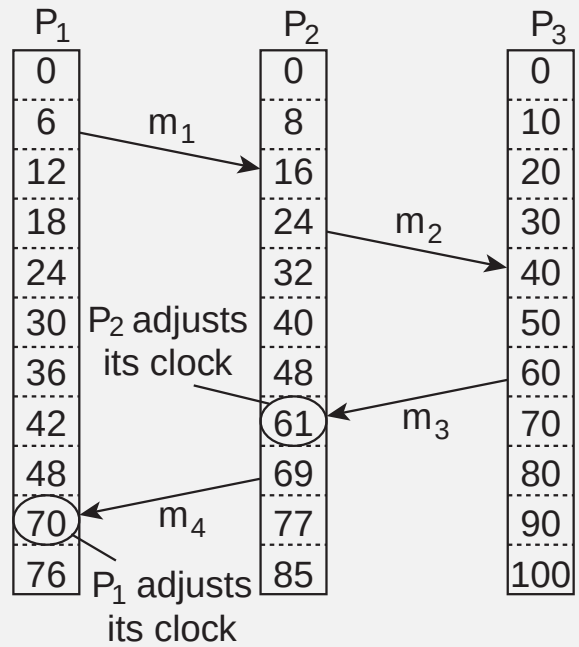
Ha a rendszerben van globális óra, azzal a fenti időbélyegek elkészíthetők. A továbbiakban azt vizsgáljuk, hogyan lehet az időbélyegeket globális óra nélkül elkészíteni.

Logikai órák: Lamport-féle időbélyegek

Minden P_i folyamat saját C_i számlálót tart nyilván az alábbiak szerint:

- P_i minden eseménye eggyel növeli a számlálót.
- Az elküldött m üzenetre ráírjuk az időbélyeget: $ts(m) = C_i$.
- Ha az m üzenet beérkezik P_j folyamathoz, ott a számláló új értéke $C_j = \max\{C_j, ts(m)\} + 1$ lesz; így az idő „nem folyik visszafelé”.
- P_i és P_j egybeeső időbélyegei közül tekintsük a P_i-belit elsőnek, ha $i < j$.

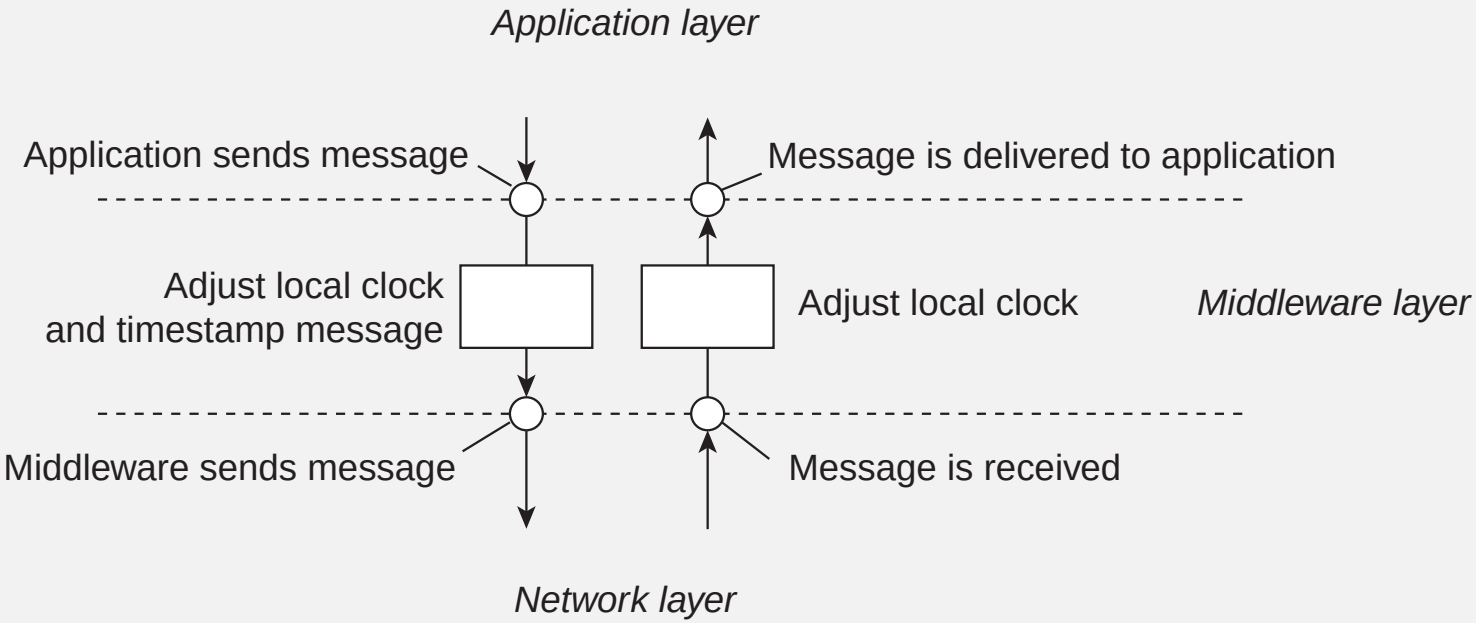




Logikai órák

Beállítás: köztesréteg

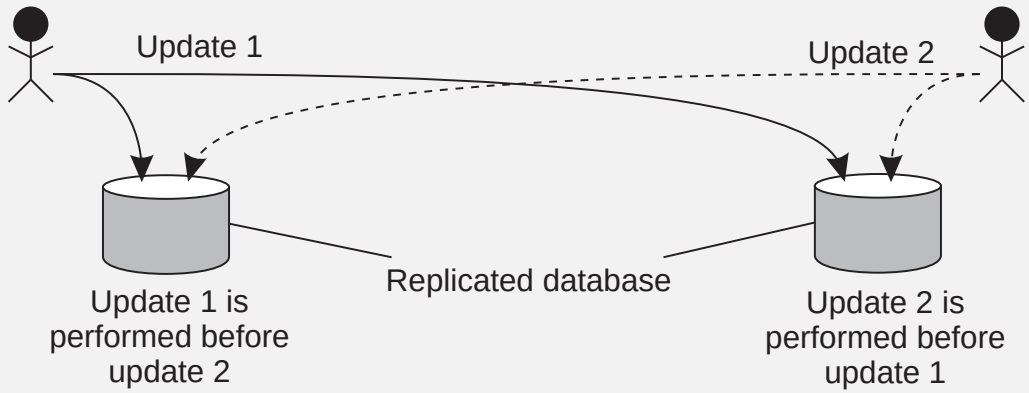
Az órák állítását és az üzenetek időbélyegeit a köztesréteg kezeli.



Logikai órák: példa

Pontosan sorbarendezett csoportcímzés

Ha replikált adatbázison konkurens műveleteket kell végezni, sokszor követelmény, hogy mindegyik másolaton ugyanolyan sorrendben hajtódjanak végre a műveletek.
Az alábbi példában két másolatunk van, a számlán kezdetben \$1000 van.
P_1 befizet \$100-t, P_2 1% kamatot helyez el.



Probléma

Ha a műveletek szinkronizációja nem megfelelő, érvénytelen eredményt kapunk: másolat$_1$ $\leftarrow$ \$1111, de másolat$_2$ $\leftarrow$ \$1110.

Példa: Pontosan sorbarendezett csoportcímzés

Pontosan sorbarendezett csoportcímzés

- A P_i folyamat minden műveletet időbélyeggel ellátott üzenetben küld el. P_i egyúttal beteszi a küldött üzenetet a saját $queue_i$ prioritásos sorába.
- A P_j folyamat a beérkező üzeneteket az ő $queue_j$ sorába teszi be az időbélyegnek megfelelő prioritással. Az üzenet érkezéséről mindegyik folyamatot értesíti.

P_j akkor adja át a msg_i üzenetet feldolgozásra, ha:

(1) msg_i a $queue_j$ elején található (azaz az ő időbélyege a legkisebb)
(2) a $queue_j$ sorban minden P_k ($k \neq i$) folyamatnak megtalálható legalább egy üzenete, amelynek msg_i-nél későbbi az időbélyege

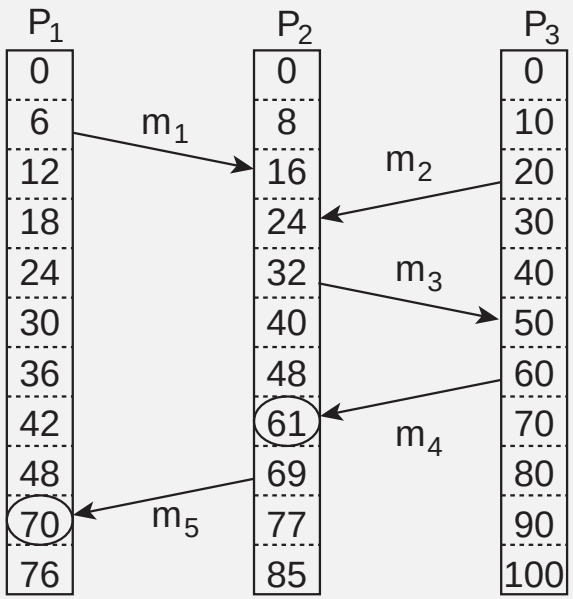
Feltételek

Feltételezzük, hogy a kommunikáció a folyamatok között megbízható és FIFO sorrendű.

Időbélyeg-vektor

Okság

Arra is szükségünk lehet, hogy megállapíthassuk két eseményről, hogy az egyik okoz(hat)ta-e a másikat – illetve fordítva, függetlenek-e egymástól.
Az eddigi megközelítésünk erre nem alkalmas: abból, hogy $C(a) < C(b)$, nem vonható le az a következtetés, hogy az a esemény okságilag megelőzi a b eseményt.



A példában szereplő adatok

a esemény: m_1 beérkezett $T = 16$ időbélyeggel;
b esemény: m_2 elindult $T = 20$ időbélyeggel.
Bár $16 < 20$, a és b nem függenek össze okságilag.

Időbélyeg-vektor

Időbélyeg-vektor

- A P_i most már az összes másik folyamat idejét is számon tartja egy $VC_i[1..n]$ tömbben, ahol $VC_i[j]$ azoknak a P_j folyamatban bekövetkezett eseményeknek a száma, amelyekről P_i tud.
- Az m üzenet elküldése során P_i megnöveli eggyel $VC_i[i]$ értékét (vagyis az üzenetküldés egy eseménynek számít), és a teljes V_i időbélyeg-vektort ráírja az üzenetre.
- Amikor az m üzenet megérkezik a P_j folyamathoz, amelyre a $ts(m)$ időbélyeg van írva, két dolog történik:

  (1) $VC_j[k] := \max\{VC_j[k], ts(m)[k]\}$
  (2) $VC_j[j]$ megnő eggyel, vagyis az üzenet fogadása is egy eseménynek számít

Pontosan sorbarendezett csoportcímzés

Időbélyeg-vektor alkalmazása

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

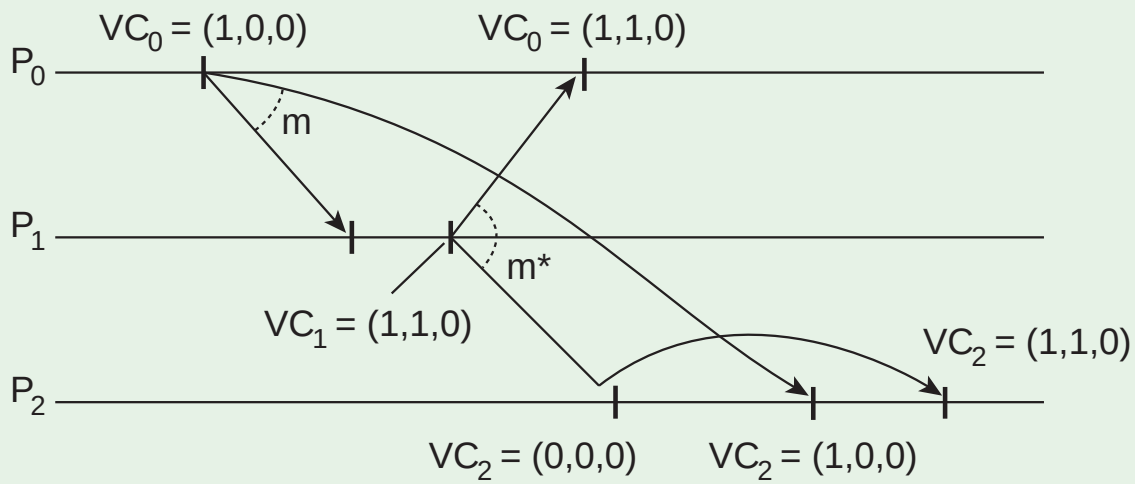
Ehhez annyit változtatunk az előbb leírt időbélyeg-vektorok működésén, hogy amikor P_j fogad egy üzenetet, akkor nem növeljük meg $VC_j[j]$ értékét.

P_j csak akkor kézbesíti az m üzenetet, amikor:

- $ts(m)[i] = VC_j[i] + 1$, azaz a P_j folyamatban P_i minden korábbi üzenetét kézbesítettük
- $ts(m)[k] \leq VC_j[k]$ for $k \neq i$, azaz az üzenet „nem a jövőből jött”

Pontosan sorbarendezett csoportcímzés

Példa



Kölcsönös kizárás

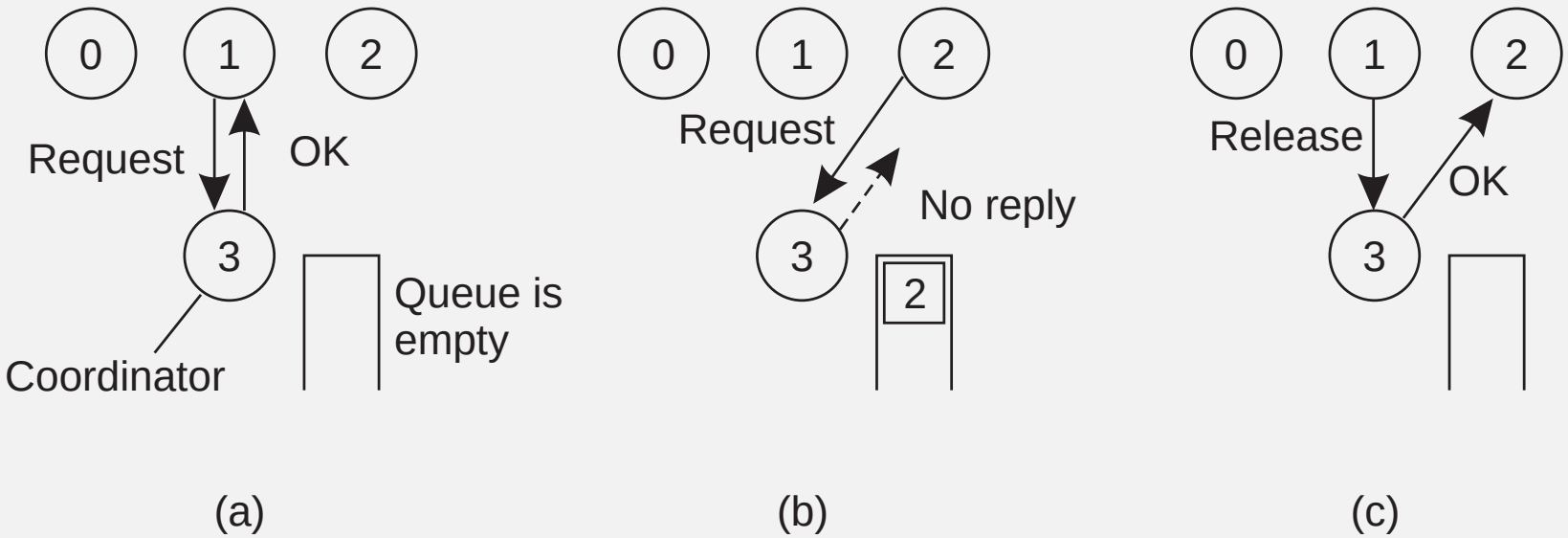
Kölcsönös kizárás: a feladat

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

Megoldásfajták

- Központi szerver használata.
- Peer-to-peer rendszeren alapuló teljesen elosztott megoldás.
- Teljesen elosztott megoldás általános gráfszerkezetre.
- Teljesen elosztott megoldás (logikai) gyűrűben.

Kölcsönös kizárás: központosított



Kölcsönös kizárás: decentralizált

Kölcsönös kizárás: decentralizált

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 engedélye szükséges, ahol $m > n/2$.
Feltesszük, hogy egy esetleges összeomlás után a koordinátor hamar felépül – azonban a kiadott engedélyeket elfelejti.

Példa: hatékonyság

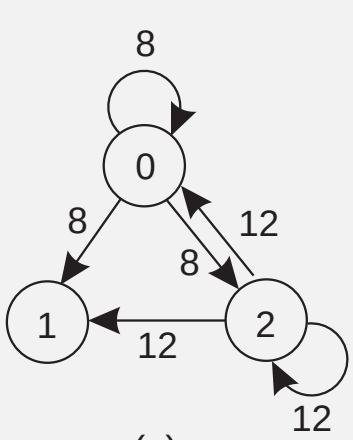
Tegyük fel, hogy a koordinátorok rendelkezésre állásának valószínűsége 99.9% („három kilences”), 32-szeresen replikált az erőforrásunk, és a koordinátorok háromnegyedének engedélyére van szükségünk ($m = 0.75n$).
Ekkor annak a valószínűsége, hogy túl sok koordinátor omlik össze, igen alacsony: kevesebb mint 10^{-40}.

Kölcsönös kizárás: elosztott

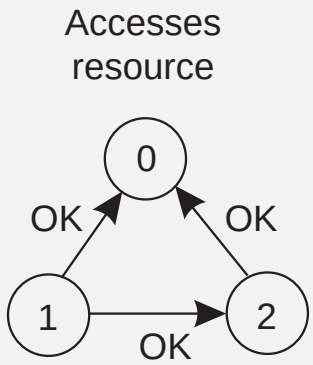
Működési elv

Ismét többszörözött az erőforrás, amikor a kliens hozzá szeretne férni, kérést küld mindegyik koordinátornak (időbélyeggel).
Választ (hozzáférési engedélyt) akkor kap, ha

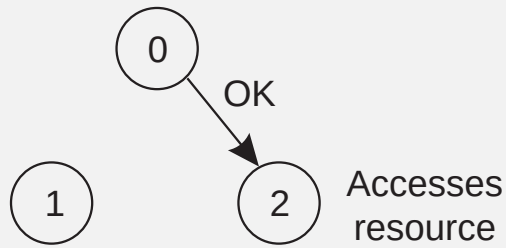
- a koordinátor nem igényli az erőforást, vagy
- a koordinátor is igényli az erőforrást, de kisebb az időbélyege.
- Különben a koordinátor (átmenetileg) nem válaszol.



(a)



(b)

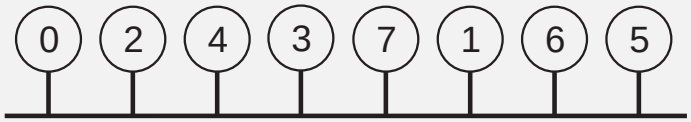


(c)

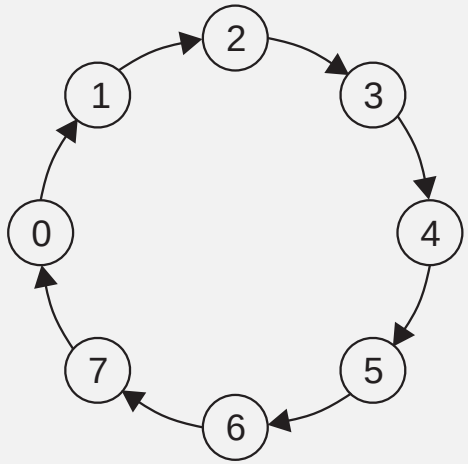
Kölcsönös kizárás: zsetongyűrű

Essence

A folyamatokat logikai gyűrűbe szervezzük (fizikailag lehetnek pl. egy lokális hálózaton).
A gyűrűben egy zsetont küldünk körbe, amelyik folyamat birtokolja, az férhet hozzá az erőforráshoz.



(a)



(b)

Kölcsönös kizárás: összehasonlítás

Algoritmus	Be+kilépési üzenetszám	Belépés előtti késleltetés (üzenetidőben)	Problémák
Központosított	3	2	Ha összeomlik a koordinátor
Decentralizált	2mk + m	2mk	Kiéheztetés, rossz hatékonyság
Elosztott	2 (n – 1)	2 (n – 1)	Bármely folyamat összeomlása
Zsetongyűrű	1 .. ∞	0 .. n – 1	A zseton elvész, a birtokló folyamat összeomlik

Csúcsok globális pozícionálása

Feladat

Meg szeretnénk becsülni a csúcsok közötti kommunikációs költségeket. Erre többek között azért van szükség, hogy hatékonyan tudjuk megválasztani, melyik gépekre helyezzünk replikátumokat az adatainkból.

Ábrázolás

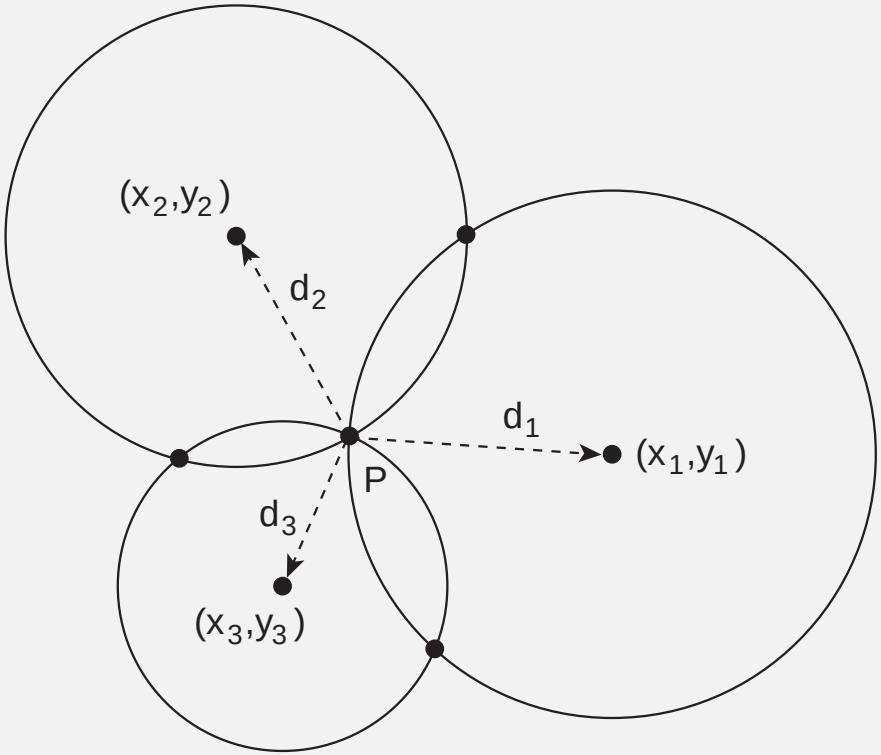
A csúcsokat egy többdimenziós geometriai térben ábrázoljuk, ahol a P és Q csúcsok közötti kommunikációs költséget a csúcsok távolsága jelöli. Így a feladatot visszavezettük távolságok becslésére.

A tér dimenziószáma minél nagyobb, annál pontosabb lesz a becslésünk, de annál költségesebb is.

A pozíció kiszámítása

A becsléshez szükséges csúcsok száma

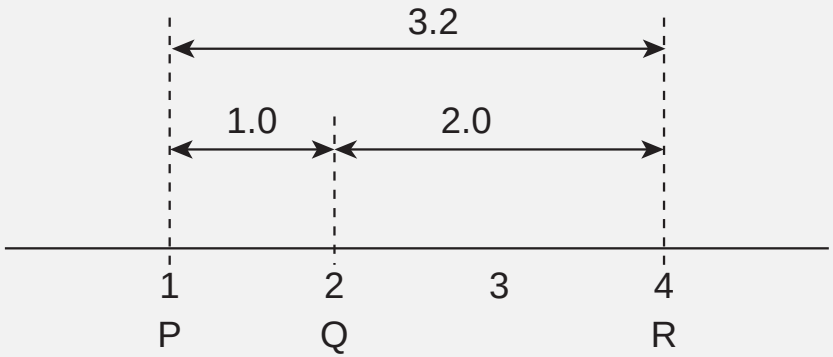
Egy pont pozíciója meghatározható a tér dimenziószámánál eggyel nagyobb számú másik pontból a tőlük vett távolságok alapján.



A pozíció kiszámítása

Nehézségek

- a késleltetések mért értékei ingadoznak
- nem egyszerűen összeadódnak a távolságok $\longrightarrow$



Megoldás

Válasszunk L darab csúcsot, amelyek pozícióját tegyük fel, hogy nagyon pontosan meghatároztuk.

Egy P csúcsot ezekhez viszonyítva helyezünk el: megmérjük az összestől mért késleltetését, majd úgy választjuk meg P pozícióját, hogy az össz-hiba (a mért késleltetések és a megválasztott pozícióból geometriailag adódó késleltetés eltérése) a legkisebb legyen.

Vezetőválasztás: zsarnok-algoritmus

Vezetőválasztás: feladat

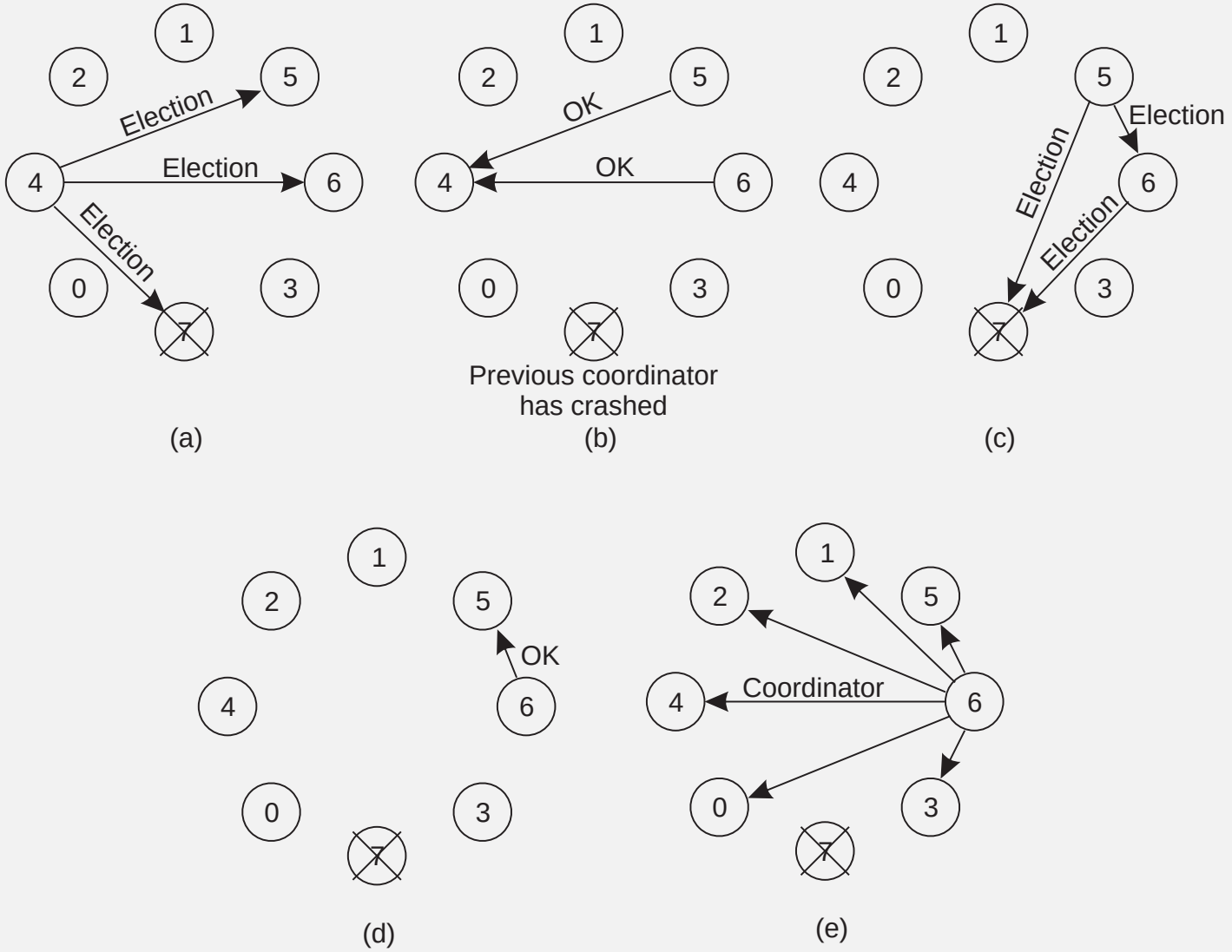
Sok algoritmusnak szüksége van arra, hogy kijelöljön egy folyamatot, amely aztán a további lépéseket koordinálja.
Ezt a folyamatot dinamikusan szeretnénk kiválasztani.

Zsarnok-algoritmus

A folyamatoknak sorszámot adunk. A legnagyobb sorszámú folyamatot szeretnénk vezetőnek választani.

- Bármelyik folyamat kezdeményezhet vezetőválasztást. Mindegyik folyamatnak (amelyről nem ismert, hogy kisebb lenne a küldőnél a sorszáma) elküld egy választási üzenetet.
- Ha $P_{nagyobb}$ üzenetet kap P_{kisebb}-től, visszaküld neki egy olyan üzenetet, amellyel kiveszi P_{kisebb}-et a választásból.
- Ha P megadott időn belül nem kap letiltó üzenetet, ő lesz a vezető. Erről mindegyik másik folyamatot értesíti egy üzenettel.

Zsarnok-algoritmus



Vezetőválasztás gyűrűben

Vezetőválasztás gyűrűben

Ismét logikai gyűrűnk van, és a folyamatoknak vannak sorszámai. A legnagyobb sorszámú folyamatot szeretnénk vezetőnek választani.

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

Amikor az üzenet visszajut a kezdeményezőhöz, minden aktív folyamat sorszáma szerepel rajta. Ezek közül a legnagyobb lesz a vezető; ezt egy másik üzenet körbeküldése tudatja mindenkivel.

Nem okozhat problémát, ha több folyamat is egyszerre kezdeményez választást, mert ugyanaz az eredmény adódik. Ha pedig az üzenetek valahol elvesznének (összeomlik az éppen őket tároló folyamat), akkor újrakezdhető a választás.

Superpeer-választás

Szempontok

A superpeer-eket úgy szeretnénk megválasztani, hogy teljesüljön rájuk:

- 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

Megvalósítás DHT használata esetén

Az azonosítók terének egy részét fenntartjuk a superpeer-ek számára.
Példa: ha m-bites azonosítókat használunk, és S superpeer-re van szükségünk, a $k = \lceil \log_2 S \rceil$ felső bitet foglaljuk le a superpeer-ek számára. Így N csúcs esetén kb. $2^{k-m}N$ darab superpeer lesz.

A p kulcshoz tartozó superpeer: a p AND $\underbrace{11\cdots11}_{k}\underbrace{00\cdots00}_{m-k}$ kulcs felelőse az.