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

8. rész: Hibatűrés

2015. május 24.

Tartalomjegyzék

Megbízhatóság

Alapok

A komponensek feladata, hogy a kliensek számára szolgáltatásokat tesz elérhetővé. Ehhez sokszor maga is szolgáltatásokat vesz igénybe más komponensektől ⇒ függ tőlük.
A függéseket most főleg abból a szempontból vizsgáljuk, hogy a hivatkozott komponensek helyessége kihat a hivatkozó helyességére.

Elérhetőség	A komponens reagál a megkeresésre
Megbízhatóság	A komponens biztosítja a szolgáltatást
Biztonságosság	A komponens ritkán romlik el
Karbantarthatóság	Az elromlott komponens könnyen javítható

Terminológia

Hasonló nevű, de különböző fogalmak

- Hibajelenség (failure): a komponens nem a tőle elvártaknak megfelelően üzemel
- Hiba (error): olyan rendszerállapot, amely hibajelenséghez vezethet
- Hibaok (fault): a hiba (feltételezett) oka

Hibákkal kapcsolatos tennivalók

- Megelőzés
- Hibatűrés: a komponens legyen képes elfedni a hibákat
- Mérséklés: lehet mérsékelni a hibák kiterjedését, számát, súlyosságát
- Előrejelzés: előre becsülhető lehet a hibák száma, jövőbeli előfordulása, következményei

Lehetséges hibaokok

Lehetséges hibaokok

A hibáknak sok oka lehet, többek között az alábbiak.

- Összeomlás (crash): a komponens leáll, de előtte helyesen működik
- Kiesés (omission): a komponens nem válaszol
- Időzítési hiba (timing): a komponens helyes választ küld, de túl későn (ennek teljesítménnyel kapcsolatos oka lehet: a komponens túl lassú)
- Válaszhiba (response): a komponens hibás választ küld

  Értékhiba: a válaszként küldött érték rossz
  Állapotátmeneti hiba: a komponens helytelen állapotba kerül

- Váratlan hiba (arbitrary): véletlenszerű válaszok, gyakran véletlenszerű időzítéssel

Összeomlás

Probléma

A kliens számára nem különböztethető meg, hogy a szerver összeomlott vagy csak lassú.

Ha a kliens a szervertől adatot vár, de nem érkezik, annak oka lehet...

- időzítési vagy kieséses hiba a szerveren.
- a szerver és a kliens közötti kommunikációs csatorna meghibásodása.

Lehetséges feltételezések

- Fail-silent: a komponens összeomlott vagy kiesett egy válasz; a kliensek nem tudják megkülönböztetni a kettőt
- Fail-stop: a komponens hibajelenségeket produkál, de ezek felismerhetőek (a komponens közzéteszi, vagy időtúllépésből szűrjük le)
- Fail-safe: a komponens csak olyan hibajelenségeket produkál, amelyek nem okoznak (nagy) kárt

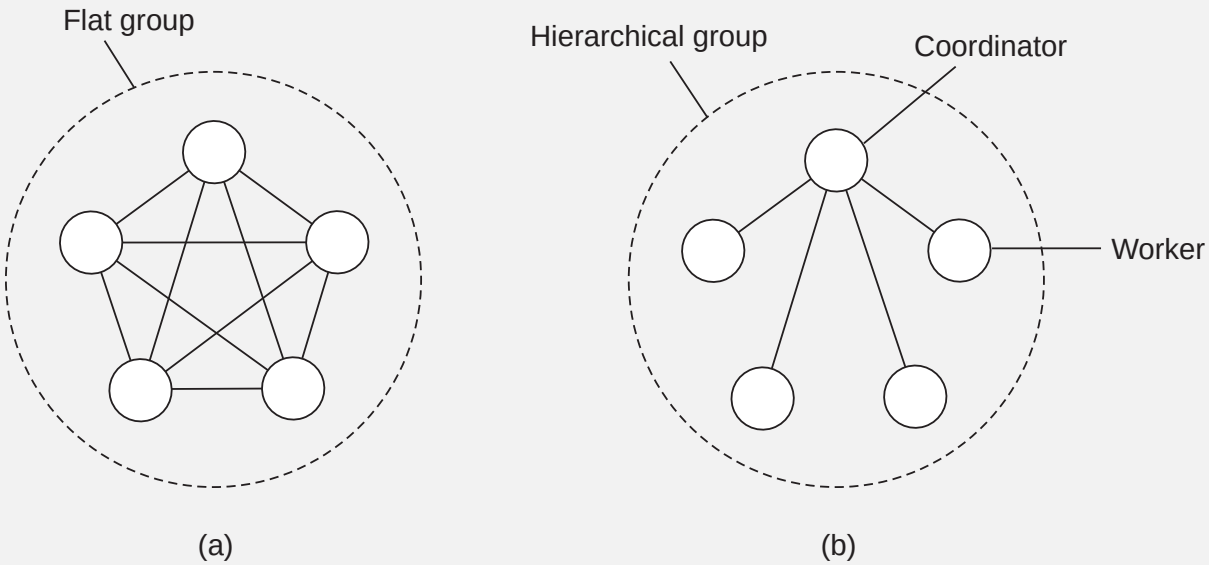
Folyamatok hibatűrése

Cél

Próbáljuk meg a hibákat redundanciával elfedni.

Egyenlő csoport: Jó hibatűrést biztosít, mert a csoport tagjai között közvetlen az információcsere. A nagymértékű többszörözöttség azonban megnehezíti az implementálást (a vezérlés teljesen elosztott).

Hierarchikus csoport: Két fél csak a koordinátoron keresztül képes kommunikálni egymással. Rossz a hibatűrése és a skálázhatósága, de könnyű implementálni.



Csoporttagság kezelése

Csoportkezelő: Egy koordinátor kezeli a csoportot (vagy csoportokat). Rosszul skálázódik.

Csoportkezelők csoportja: A csoportkezelő nem egyetlen szerver, hanem egy csoportjuk közösen. Ezt a csoportot is kezelni kell, de ezek a szerverek általában elég stabilak, így ez nem probléma: centralizált csoportkezelés elegendő.

Csoportkezelő nélkül: A belépő/kilépő folyamat minden csoporttagnak üzenetet küld.

Hibaelfedés csoportokban

k-hibatűrő csoport

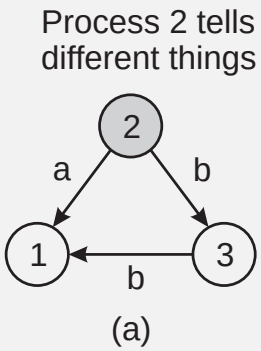
k-hibatűrő csoport: olyan csoport, amely képes elfedni k tag egyszerre történő meghibásodását.

Mekkorának kell a csoportnak lennie?

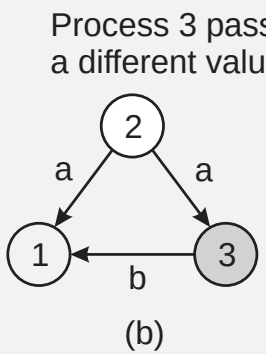
- Ha a meghibásodottak összeomlanak, csak az szükséges, hogy legyen megmaradó tag $\Rightarrow k+1$ tag
- Ha minden csoporttag egyformán működik, de a meghibásodottak rossz eredményt adnak $\Rightarrow 2k+1$ tag esetén többségi szavazással megkapjuk a helyes eredményt
- Ha a csoporttagok különbözően működnek, és mindegyikük kimenetét el kell juttatni mindenki másnak („bizánci generálisok”) $\Rightarrow 3k+1$ tag esetén a „lojális” szerverek képesek megfelelő adatokat továbbítani k „áruló” jelenlétében is

Hibaelfedés csoportokban

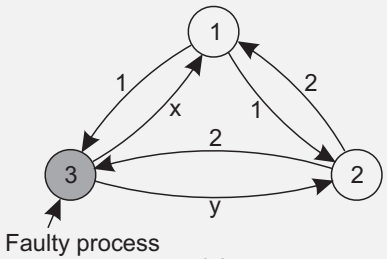
Az áruló folyamatok többféleképpen is tudnak helytelen adatokat továbbítani.



(a)



(b)



(a)

1	Got(1, 2, x)
2	Got(1, 2, y)
3	Got(1, 2, 3)

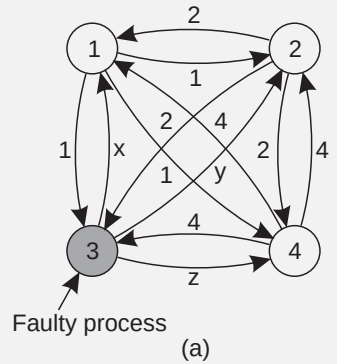
(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

Az alábbi két ábra illusztrálja $k = 1$ esetben, hogy k áruló esetén $3k + 1$ folyamat elegendő, de $3k$ még nem.

Először mindenki elküldi a saját értékét, majd minden beérkezett értéket ismét továbbítanak.



(a)

1	Got(1, 2, x, 4)
2	Got(1, 2, y, 4)
3	Got(1, 2, 3, 4)
4	Got(1, 2, z, 4)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

Hibaelfedés csoportokban

Folyamatok:	Szinkron: a folyamatok azonos ütemben lépnek-e (lockstep)?
Késleltetések:	A kommunikációs késleltetésekre van-e felső korlát?
Rendezettség:	Az üzeneteket a feladás sorrendjében kézbesítik-e?
Átvitel:	Az üzeneteket egyenként küldjük (unicast) vagy több-címűen (multicast)?

Egyezség elérése

Néhány feltételrendszer, amely fennállásával elérhető, hogy kialakuljon a közös végeredmény:

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

Hibák észlelése

Időtúllépés

Időtúllépés észlelése esetén feltételezhetjük, hogy hiba lépett fel.

- Az időkorlátok megadása alkalmazásfüggő, és nagyon nehéz
- A folyamatok hibázása nem megkülönböztethető a hálózati hibáktól
- A hibákról értesíteni kell a rendszer többi részét is
  - Pletykálással elküldjük a hiba észlelésének tényét
  - A hibajelenséget észlelő komponens maga is hibaállapotba megy át

Megbízható kommunikáció

Csatornák

A folyamatok megbízhatóságát azok csoportokba szervezésével tudtuk növelni. Mit lehet mondani az őket összekötő kommunikációs csatornák biztonságosságáról?

Hibajelenségek észlelése

- A csomagokat ellenőrző összegekkel látjuk el (felfedi a bithibákat)
- A csomagokat sorszámmal látjuk el (felfedi a kieséseket)

Hibajavítás

- A rendszer legyen annyira redundáns, hogy *automatikusan meg lehessen javítani*
- Kérhetjük a meghibásodott/elveszett utolsó N csomag újraküldését

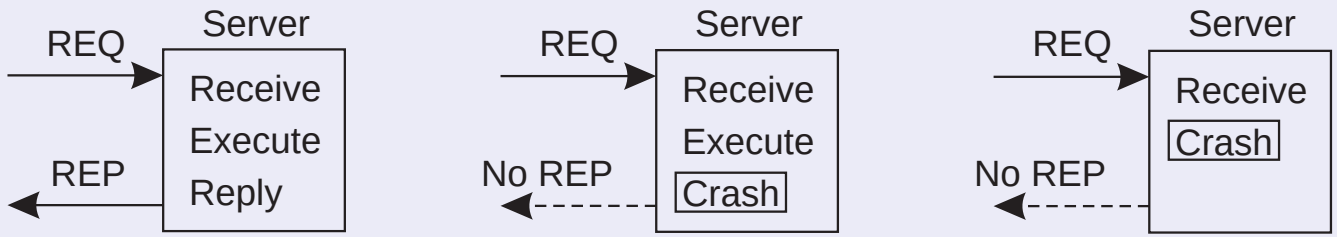
Megbízható RPC

RPC: Fellépő hibajelenségek és azok kezelése

1: A kliens nem találja a szervert. Ezt egyszerűen ki lehet jelezni a kliensnél.

2: A kliens kérése elveszett. A kliens újraküldi a kérést.

3: A szerver összeomlott. Ezt nehezebb kezelni, mert a kliens nem tudja, hogy a szerver mennyire dolgozta fel a kérést.



El kell dönteni, hogy milyen módon működjön a szerver.

- „Legalább egyszer” szemantika szerint: A szerver az adott kérést legalább egyszer végrehajtja (de lehet, hogy többször is).
- „Legfeljebb egyszer” szemantika szerint: A szerver az adott kérést legfeljebb egyszer hajtja végre (de lehet, hogy egyszer sem).

Megbízható RPC

RPC: Fellépő hibajelenségek és azok kezelése

4: Elveszett a szerver válasza. Nehéz felismerni, mert a szerver összeomlása is hasonló a kliens szemszögéből. Nincsen általános megoldás; ha idempotens műveleteink vannak (többszöri végrehajtás ugyanazt az eredményt adja), megpróbálhatjuk újra végrehajtani őket.

5: Összeomlik a kliens. Ilyenkor a szerver feleslegesen foglalja az erőforrásokat: árva[a] feladatok keletkeznek.

- A kliens felépülése után az árva feladatokat szükség szerint leállítja/visszagörgeti a szerver.
- A kliens felépülése után új korszak kezdődik: a szerver leállítja az árva feladatokat.
- A feladatokra időkorlát adott. A túl sokáig futó feladatokat a szerver leállítja.

[a] Más fordításban: fattyú.

Megbízható csoportcímzés

Adott egy többcímű átviteli csatorna; egyes folyamatok üzeneteket küldenek rá és/vagy fogadnak róla.

Megbízhatóság: Ha m üzenet csatornára küldésekor egy folyamat a fogadók közé tartozik, és köztük is marad, akkor m üzenetet kézbesíteni kell a folyamathoz. Sokszor a sorrendtartás is követelmény.

Atomi csoportcímzés: Cél: elérni azt, hogy csak akkor kézbesítsük az üzenetet, ha garantáltan mindegyik fogadónak kézbesíthető.

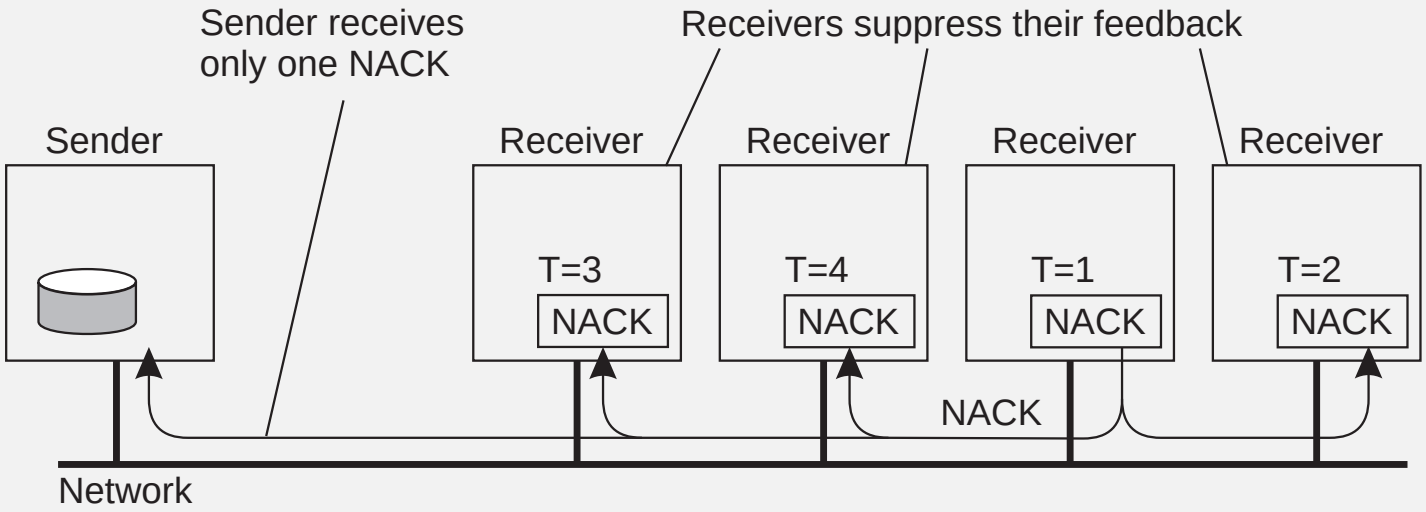
Egyszerű megoldás lokális hálózaton

A küldő számon tartja az elküldött üzeneteket. Minden fogadó nyugtázza (ACK) az üzenet fogadását, vagy kéri az üzenet újraküldését (NACK), ha észreveszi, hogy elveszett. Ha minden fogadótól megérkezett a nyugta, a küldő törli az üzenetet a történetből.
Ez a megoldás nem jól skálázódik:

- Ha sok a fogadó, a küldőhöz túl sok ACK és NACK érkezik.
- A küldőnek minden fogadót ismernie kell.

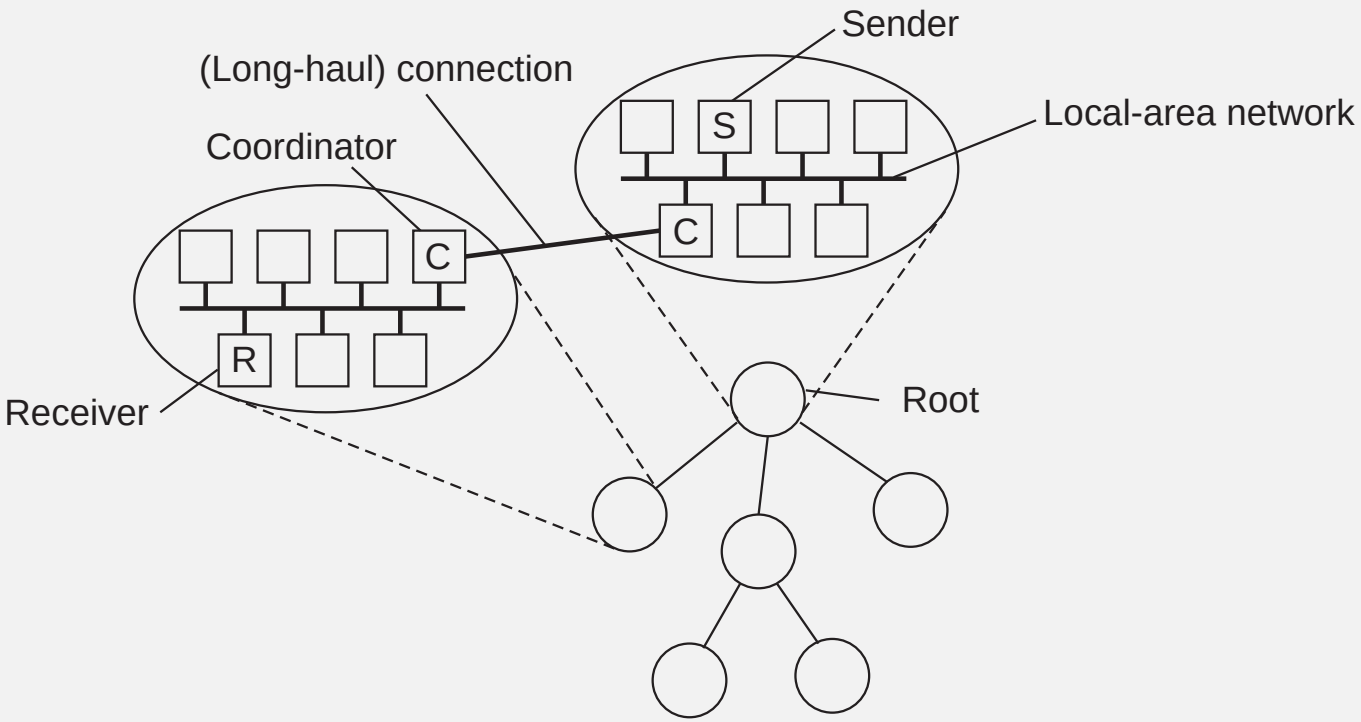
Megbízható csoportcímzés: visszajelzés-elfojtás

Hagyjuk el az ACK üzeneteket, csak a kimaradó üzenetek esetén küldjenek a fogadók NACK üzeneteket. Tegyük fel, hogy a NACK üzenetek közös csatornán utaznak, és mindenki látja őket. A *P* fogadó a NACK küldése előtt véletlen ideig vár; ha közben egy másik fogadótól NACK érkezik, akkor *P* nem küld NACK-t (elfojtás), ezzel csökkenti az üzenetek számát. A NACK hatására a küldő mindegyik fogadóhoz újra eljuttatja az üzenetet, így *P*-hez is.



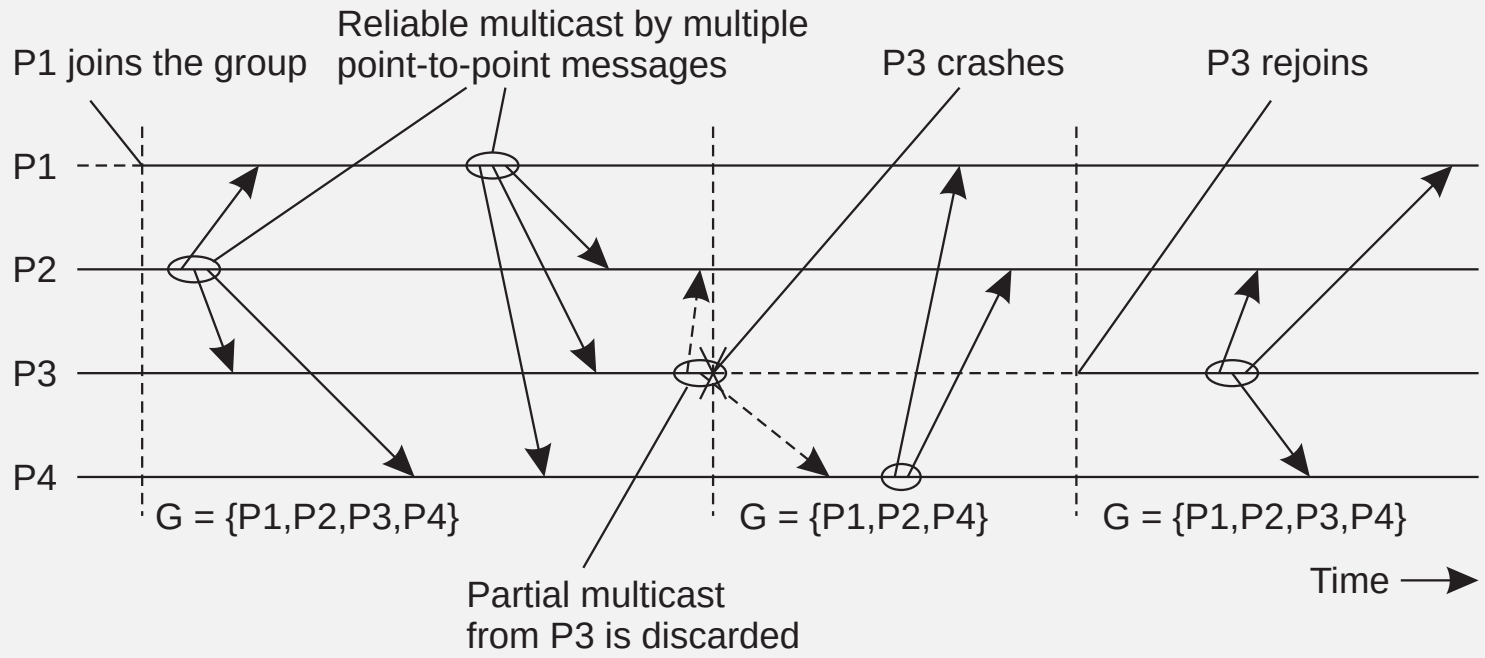
Megbízható csoportcímzés: hierarchikus visszajelzés-vezérlés

Az előző módszer lokális hálózatban működik hatékonyan. Az átméretezhetőség javításához a lokális hálózatokból építsünk fát; az üzenetek a fában felfelé, a NACK üzenetek felfelé utaznak.
Probléma: a fát dinamikusan kell építeni, a meglévő hálózatok átkonfigurálása nehéz lehet.



Megbízható csoportcímzés: elemi csoportcímzés

A folyamatok számon tartják, hogy kik a csoport tagjai: nézet (view). Egy üzenetet csak akkor kézbesítenek, ha az eljutott az adott nézet minden tagjához. Mivel a nézetek időben változhatnak, ez ún. látszólagos szinkronizáltságot (virtual synchrony) ad.



Elosztott véglegesítés: 2PC

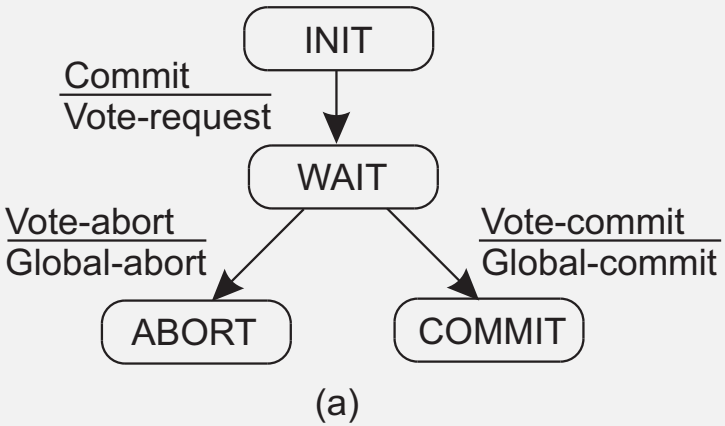
2PC

Egy elosztott számítás végén szeretnénk elérni, hogy vagy minden folyamat véglegesítse az eredményt, vagy egyik sem. (atomiság)
Ezt általában kétfázisú véglegesítéssel (two-phase commit, 2PC) szokás elérni. A számítást az azt kezdeményező kliens koordinátorként vezérli. Az egyes fázisokban a koordinátor (K) és egy-egy résztvevő (R) az alábbiak szerint cselekszik.

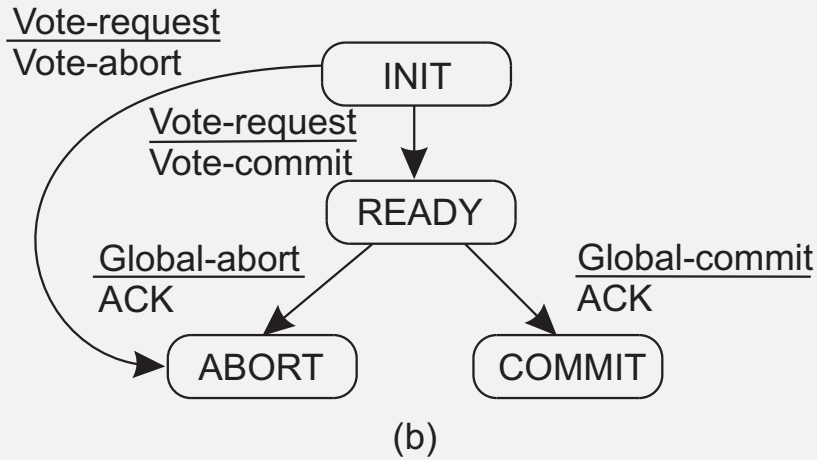
- 1/K: Megkérdez mindenkit, enged-e véglegesíteni: vote-request.
- 1/R: Dönt: igennel (vote-commit) vagy nem (vote-abort), utóbbi esetben rögtön el is veti a kiszámított értéket.
- 2/K: Ha minden válasz vote-commit, mindenkinek global-commit üzenetet küld, különben global-abort üzenetet.
- 2/R: Végrehajtja a kapott globális utasítást: elmenti a kiszámított adatokat (eddig csak átmenetileg tárolta őket lokálisan), illetve törli őket.

Elosztott véglegesítés: 2PC

A vonalak fölött a kapott, alattuk a küldött üzenetek nevei láthatóak.



A 2PC koordinátor állapotgépe



Egy 2PC résztvevő állapotgépe

Elosztott véglegesítés: 2PC, résztvevő összeomlása

Résztvevő összeomlása

Egy résztvevő összeomlott, majd felépült. Mit tegyen adott állapotban?

- `INIT`: A résztvevő nem is tudott a véglegesítésről, döntés: `ABORT`.
- `ABORT`: Az eredmények törlendők. Az állapot marad `ABORT`.
- `COMMIT`: Az eredmények elmentendők. Az állapot marad `COMMIT`.
- `READY`: A résztvevő a koordinátor 2/K döntésére vár, amit megkérdez tőle, vagy a koordinátor összeomlása esetén a többi résztvevőtől. Azok vagy már tudnak róla (`ABORT` vagy `COMMIT`), vagy még `INIT` állapotban vannak (ekkor `ABORT` mellett dönthetnek). Ha mindegyik résztvevő `READY` állapotban van, akkor megoldható a helyzet, azonban...

A 2PC protokoll blokkolódhat

Ha a koordinátor és legalább egy résztvevő összeomlott, és a többi résztvevő a `READY` állapotban van, akkor a protokoll beragad, ugyanis nem ismert, hogy a kiesett résztvevő kapott-e az összeomlása előtt valamilyen utasítást a koordinátortól. Ez szerencsére szinte sohasem fordul elő a gyakorlatban.

Elosztott véglegesítés: 3PC

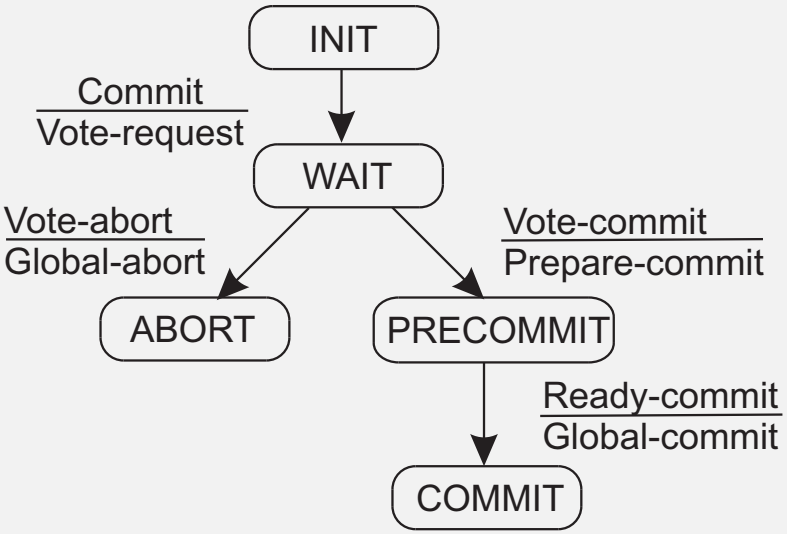
3PC

- 1/K és 1/R: Mint korábban: megkezdődik a szavazás (vote-request), és a résztvevők válaszolnak (vote-commit vagy vote-abort)
- 2/K: Ha van vote-abort) szavazat, minden résztvevőnek global-abort üzenetet küld, majd leáll. Ha minden szavazat vote-commit, prepare-commit üzenetet küld mindenkinek.
- 2/R: global-abort esetén leáll; prepare-commit üzenetre ready-commit üzenettel válaszol.
- 3/K: Összegyűjti a prepare-commit üzeneteket; küldi: global-commit.
- 3/R: Fogadja a global-commit üzenetet, és elmenti az adatokat.

A 3PC protokoll nem blokkolódhat

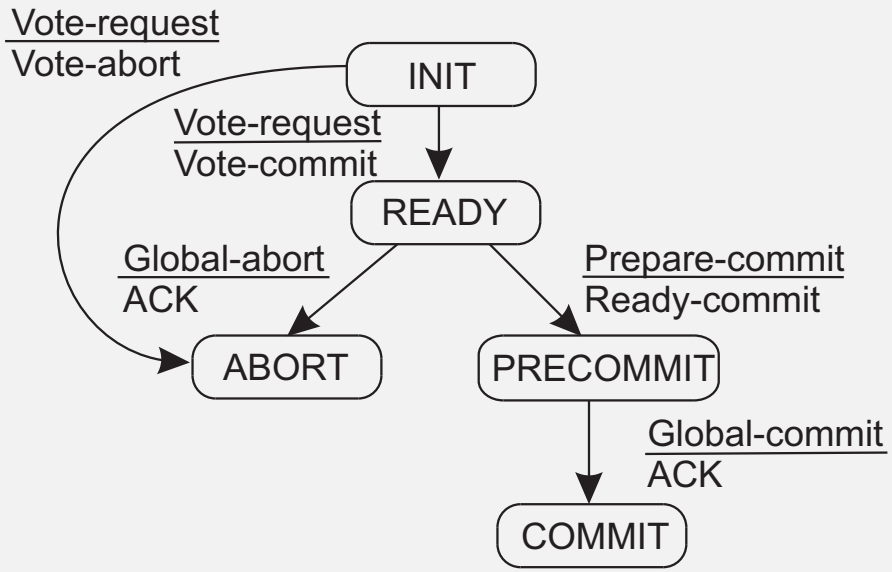
Mivel a koordinátor és a résztvevők mindig legfeljebb „egy távolságban vannak” az állapotaikat tekintve, ha mindegyik aktív résztvevő `READY` állapotban van, akkor a kiesett résztvevő még nem járhat a `COMMIT` fázisban. Megjegyzés: a `PRECOMMIT` állapotból csak véglegesíteni lehet.

Elosztott véglegesítés: 3PC



(a)

A 3PC koordinátor állapotgépe



(b)

Egy 3PC résztvevő állapotgépe

Felépülés

Alapok

Ha hibajelenséget érzékelünk, minél hamarabb hibamentes állapotba szeretnénk hozni a rendszert.

- Előrehaladó felépülés: olyan új állapotba hozzuk a rendszert, ahonnan az folytathatja a működését
- Visszatérő felépülés: egy korábbi érvényes állapotba térünk vissza

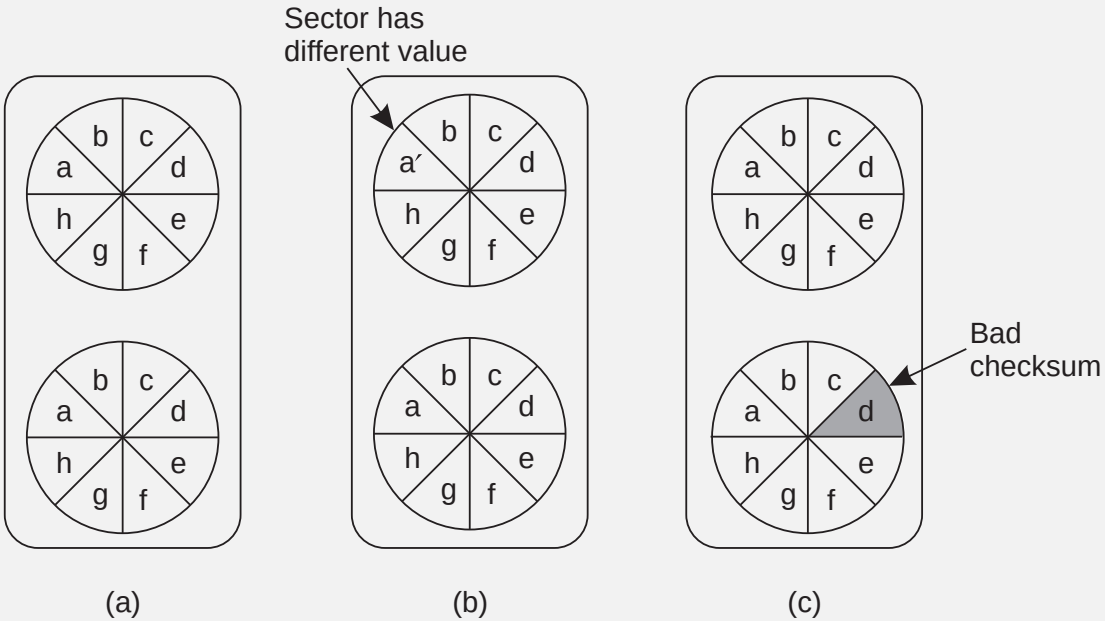
Jellemző választás

Visszatérő felépülést alkalmazunk, ehhez ellenőrzőpontokat (checkpoint) veszünk fel.

Konzisztens állapot szükséges

Az elosztott rendszerek visszaállítását nehezíti, hogy ellenőrzőpontot a rendszer egy konzisztens állapotára szeretnénk elhelyezni, ennek megtalálásához pedig a folyamatok együttműködése szükséges.

Stabil tárolás (stable storage)

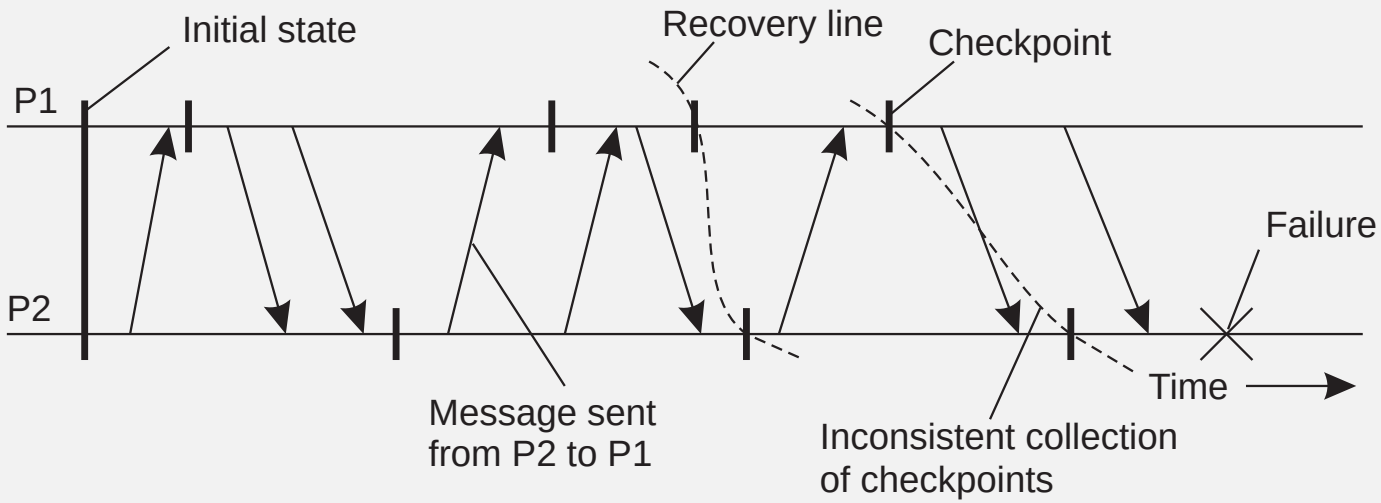


Az ellenőrzőpontokat minél megbízhatóbban kell tárolnunk. Tegyük fel, hogy fizikailag szinte elpusztíthatatlan tárunk van (stable storage), és két másolatot használunk; mit kell tenni összeomlás után?

- Ha mindkét lemez tartalma azonos, minden rendben.
- Ha az ellenőrzőösszegek alapján kiderül, melyik a helyes, azt választjuk.
- Ha helyesek, de eltérnek, az első lemezt választjuk, mert oda írunk először.
- Ha egyik sem helyes, nehezen eldönthető, melyiket kell választani.

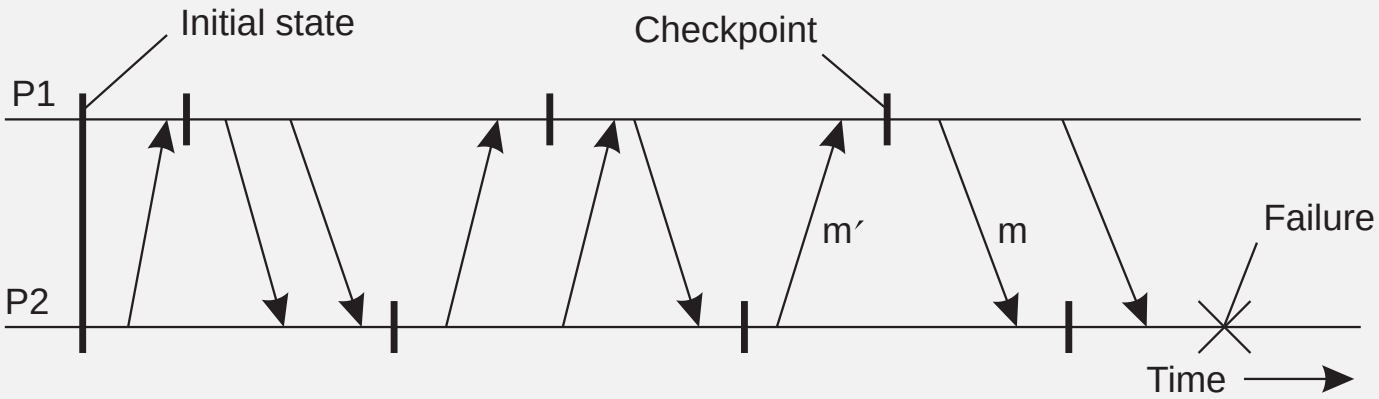
Konzisztens rendszerállapot

A folyamatok pillanatfelvételeket készítenek az állapotukról.
konzisztens metszet: olyan ellenőrzőpont-gyűjtemény, amelyben minden beérkezett üzenethez a küldési esemény is el van tárolva
felépülési vonal: a lehető legkésőbb készült (legfrissebb) konzisztens metszet



Dominóeffektus

Vissza kell keresnünk a legutolsó konzisztens metszetet. Ezért minden folyamatban vissza kell térni egy korábbi ellenőrzőponthoz, ha „elküldetlen” fogadott üzenetet találunk. Így viszont további üzenetek válnak nem elküldötté, ami továbbgyűrűző visszagörgetéshez (cascaded rollback) vezethet.



Független ellenőrzőpontok készítése

Egy lehetőség, ha mindegyik folyamat egymástól függetlenül készít ellenőrzőpontokat (independent checkpointing).

- Jelölje $CP[i](m)$ a P_i folyamat m-edik ellenőrzőpontját.
- Jelölje $INT[i](m)$ a $CP[i](m-1)$ és $CP[i](m)$ közötti időintervallumot.
- Amikor P_i üzenetet küld a $INT[i](m)$ intervallumban, hozzácsatolja i és m értékét.
- Amikor P_j fogadja az üzenetet a $INT[j](n)$ intervallumban, rögzíti, hogy új függőség keletkezett: $INT[i](m) \xrightarrow{\text{függ}} INT[j](n)$, és ezt $CP[j](n)$ készítésekor a stabil tárba menti
- Ha P_i-nek vissza kell állítania $CP[i](m-1)$-et, akkor P_j-t is vissza kell görgetni $CP[i](m-1)$-ig. A függőségek egy gráfot adnak ki, ennek a vizsgálatával derül ki, melyik folyamatot pontosan melyik ellenőrzőpontig kell visszagörgetni, legrosszabb esetben akár a rendszer kezdőállapotáig.

Koordinált ellenőrzőpontkészítés

A dominóeffektus elkerülése végett most egy koordinátor vezérli az ellenőrzőpont készítését.
Kétfázisú protokollt használunk:

- Az első fázisban a koordinátor minden folyamatot pillanatfelvétel készítésére szólít fel.
- Amikor ezt megkapja egy folyamat, elmenti az állapotát, és ezt nyugtázza a koordinátor felé. A folyamat továbbá felfüggeszti az üzenetek küldését.
- Amikor a koordinátor minden nyugtát megkapott, minden folyamatnak engedélyezi az üzenetek küldését.

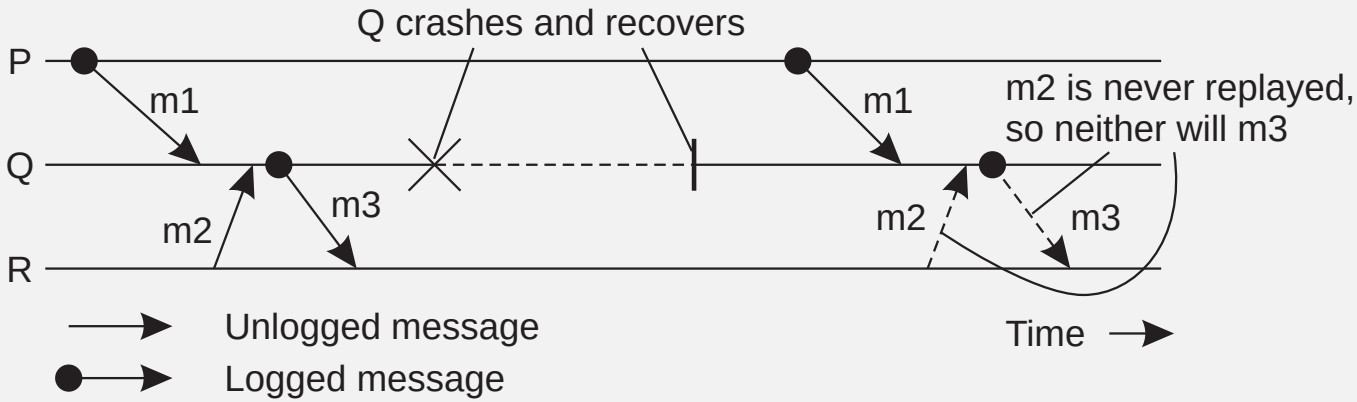
Felépülés naplózással

Ahelyett, hogy pillanatfelvételt készítenénk, most megpróbáljuk újra lejátszani a kommunikációs lépéseket az utolsó ellenőrzőponttól.
Ehhez mindegyik folyamat lokális naplót vezet.
A végrehajtási modellről feltételezzük, hogy szakaszosan determinisztikus (piecewise deterministic):

- Mindegyik folyamat végrehajtását egymás után következő szakaszok sorozata
- Mindegyik szakasz egy nemdeterminisztikus eseménnyel kezdődik (pl. egy üzenet fogadása, szignál kezelése)
- A szakaszon belül a végrehajtás determinisztikus

A naplóba elég a nemdeterminisztikus eseményeket felvenni, a többi lépés belőlük meghatározható.

Üzenetek konzisztens naplózása



Úgy szeretnénk naplózni, hogy elkerüljük árva folyamatok kialakulását.

- Q folyamat fogadta m_1-et és m_2-t, és elküldte m_3-at.
- Tegyük fel, hogy m_2 üzenetet se R, se Q nem naplózta
- Q összeomlása után a visszajátszás során ekkor senkinek sem tűnik fel, hogy m_2 kimaradt
- Mivel Q kihagyta m_2-t, ezért a tőle (lehet, hogy) függő m_3-at sem küldi el

Üzenetek konzisztens naplózása

Jelölések

HDR[m]: Az *m* üzenet fejléce, tartalmazza a küldő és a folyamat azonosítóját és az üzenet sorszámát.
Egy üzenet stabil, ha a fejléce már biztosan nem veszhet el (pl. mert stabil tárolóra írták).

COPY[m]: Azok a folyamatok, amelyekhez *HDR[m]* már megérkezett, de még nem tárolták el. Ezek a folyamatok képesek *HDR[m]* reprodukálására.

DEP[m]: Azok a folyamatok, amelyekhez megérkezett *HDR[m']*, ahol m' okozatilag függ *m*-től.

Ha *C* összeomlott folyamatok halmaza, akkor $Q \notin C$ árva, ha függ olyan *m* üzenettől, amelyet csak az összeomlottak tudnának előállítani:

$$Q \in DEP[m] \text{ és } COPY[m] \subseteq C.$$

Ha $DEP[m] \subseteq COPY[m]$, akkor nem lehetnek árva folyamataink.

Üzenetek konzisztens naplózása

Pesszimista naplózóprotokoll

Ha m nem stabil, akkor megköveteljük, hogy legfeljebb egy folyamat függjön tőle: $|DEP[m]| \leq 1$.

Implementáció: minden nem-stabil üzenetet stabilizálni kell továbbküldés előtt.

Optimistista naplózóprotokoll

Tegyük fel, hogy C a hibás folyamatok halmaza. A nem-stabil m üzenetekre, ha $COPY[m] \subseteq C$, akkor azt szeretnénk elérni, hogy egy idő után $DEP[m] \subseteq C$ is teljesüljön.

Implementáció: minden árva folyamatot visszagörgetünk olyan ellenőrzőpontig, ahol a függőség már nem áll fenn.