Többpéldányos adatkezelés

Előadásvázlat „Szolgáltatásbiztonságra tervezés” tárgyból

Majzik István
BME Méréstechnika és Információs Rendszerek Tanszék

Tartalomjegyzék:

1 Bevezetés

Speciális feladatként jelentkezik elosztott rendszerekben (illetve redundáns adatbázisokban) a *többpéldányos adatkezelés*. Ennek lényege, hogy a hozzáférés gyorsítása illetve a hibatűréshez szükséges redundancia érdekében egy-egy adatból több példányt tárolunk fizikailag különböző hostokon. A következőket szeretnénk biztosítani:

- Konkurens összetett műveletek (tranzakciók) végrehajtása: *Read*, *Write* részműveletek, majd az összetett művelet lezárása *Commit* (ha minden részművelet sikeres volt) vagy *Abort* döntéssel.
- *Egypéldányos sorosíthatóság*: A többpéldányos esetben a részműveletek konkurens végrehajtásának eredménye megegyezzen az egypéldányos esetben egy lehetséges soros végrehajtás eredményével.
- Adatpéldány kiesésének (host kiesésének) tolerálása, egyes esetekben emellett kommunikációs hibák és partíciókra szakadás tolerálása.

Az itt tárgyalt módszerek áttekintése

- Alap módszerek (önmagukban nem használatosak):
  - *Minden példány írása* (write all)
  - *Minden elérhető példány írása* (write all available)
- Host hibák esetén alkalmazható:
  - *Elérhető példányok írása validációval* (available copies with validation)
- Kommunikációs hibák, az elosztott rendszer partíciókra szakadása esetén is alkalmazható:
  - *Hostok többsége* (site quorum)
  - *Quorum konszenzus* (quorum consensus)
  - *Virtuális partíciók* (virtual partitions)

2 Minden példány írása

Implementáció: Ha az x adat példányai $x_1, x_2, \ldots, x_n$, akkor az olvasás és írás műveletek megvalósítása a következő:

- A legközelebbi (legolcsóbban olvasható) x_i példány olvasása: $Read(x) = Read(x_i)$
  (itt az egyenlőség a *Read(x)* művelet megvalósítására utal).
- Minden adatpéldány írása (n adatpéldány esetén):
  $Write(x) = \{Write(x_1), Write(x_2), \ldots, Write(x_n)\}$

Csak hibamentes esetben működőképes módszer (hiba esetén az írás nem végrehajtható), ezért csak mint rész-algoritmus használható összetettebb algoritmusokban (ld. később).

3 Minden elérhető példány írása

Implementáció:

- A legközelebbi (legolcsóbban olvasható) példány olvasása: $Read(x) = Read(x_i)$
- Minden elérhető példány írása:
  $Write(x) = \{Write(x_1), Write(x_2), \ldots, Write(x_k)\}$, ahol $k \leq n$ elérhető példány

A meghibásodás után helyreállított hostok régi adatot tartalmazhatnak. Ez a probléma kezelhető egy ”érvénytelen” jelzéssel, amit helyreállításkor kell a hoston lévő adatpéldányokra bejegyezni, és egy-egy adatpéldány legközelebbi felülírásakor (frissítésekor) törölni.

Egyes meghibásodások esetén a sorosíthatóság nem biztosított, ld. a következő példán:

- Az a host tárolja az x_a és y_a példányokat; a b host tárolja az x_b és y_b példányokat; két összetett művelet (tranzakció) indul:

T1: Read(x); Write(y)	T2: Read(y); Write(x)
Read(x_a)	Read(y_a)
← az a host meghibásodik (kiesik) →	
y_a kiesik (*Write(y_a)* nincs)	x_a kiesik (*Write(x_a)* nincs)
Write(y_b)	Write(x_b)
Commit	Commit

- A meghibásodás miatt az a host lokális ütemezője nem detektálhatja a sorosíthatósági konfliktust; a *b* hoston pedig (itt az elérhető példányok írása miatt) csak két független változó írása látszik.
- A lezajlott műveletek eredménye soros végrehajtás esetén nem állhatna elő (az esetben vagy T1 olvasná a T2 által írt értéket, vagy viszont; itt mindkét művelet a régi értéket olvassa).

4 Elérhető példányok írása validációval

Az olvasás és az írás megvalósítása:

- A legközelebbi példány olvasása: $Read(x) = Read(x_i)$
- Minden elérhető példány írása:
  $Write(x) = \{Write(x_1), \ldots, Write(x_k)\}, k \leq n$ elérhető példány

Meghibásodás után helyreállított példányok kezelése: ”érvénytelen” jelzés a legközelebbi felülírásig.

Sorosíthatósági konfliktusok kezelése: Validációs protokoll szükséges az összetett művelet (tranzakció) lezárása előtt:

- Hozzáférés validációja: A használt adatpéldányok továbbra is elérhetőek (közben nem estek ki):
  - $Available(x_k)$ üzenet minden használt példánynak
  - Commit csak akkor, ha mindegyik visszajelez

Hasonló módon történik a hiányzó írások validációja: Meg kell győződni róla, hogy az összetett művelet közben meghibásodott példányok továbbra sem elérhetők.

Ezzel a validációs protokollal elkerülhető az ”elérhető példányok írása” módszer esetén tapasztalt sorosíthatósági probléma (a hozzáférés validációja a döntési protokollal együtt megtörténhet, a hiányzó írások validációjára csak akkor van szükség, ha volt hibás példány).

5 Hostok többsége

Kommunikációs hibák esetén az elosztott rendszer partíciókra eshet szét, az egyes partíciókban függetlenül folyhatnak a tranzakciók, ami inkonzisztens értékeket jelent újra csatlakozás esetén.

Cél: Csak egy partícióban (aktív partíció) lehessen használni az adatokat. Az aktív partíció kijelölhető például a hostok többsége által (vagy a hostokhoz rendelt súlyok többsége által).

Az aktív partícióban az ”elérhető példányok írása validációval” módszer használható.

6 Quorum konszenzus

A quorum konszenzus módszere dinamikusan jelöli ki, hogy egy adott partícióban (az elérhető hostokon) megengedhető-e egy adat olvasása illetve írása. A partíció változásaihoz gyorsan adaptálódó módszer, a gyakori változásokra van felkészülve.

Alapötletek:

- Az $x_1, x_2, ..., x_n$ adatpéldányokhoz súlyokat rendelünk: $s_1, s_2, \ldots, s_n$;
  a súlyok összege $S = \sum_{i=1}^{n} s_i$
- Küszöbértékeket definiálunk:
  - Írási küszöb WT, aminek szerepe a következő: Írást akkor szabad végrehajtani, ha az elérhető adatpéldányok W halmaza *írási quorumot* képez, azaz $\sum_{x_k \in W} s_k \geq WT$
  - Olvasási küszöb RT, aminek szerepe a következő: Olvasást akkor szabad végrehajtani, ha az elérhető adatpéldányok R halmaza *olvasási quorumot* képez, azaz $\sum_{x_l \in R} s_l \geq RT$
- A küszöbértékek megválasztására a következő feltételek betartása szükséges:
  - $2WT > S$
  - $WT + RT > S$
- A küszöbértékek megválasztásának és használatának következményei:
  - $2WT > S$ miatt bármely írási quorumnak van közös eleme bármely másik írási quorummal (ez lehetővé teszi az egymás utáni írások adminisztrálását, ld. később).
  - $WT + RT > S$ miatt bármely írási quorumnak van közös eleme bármely olvasási quorummal (ez lehetővé teszi a legutolsó írás eredményének kiolvasását).
- Nem lesz szükséges minden elérhető példány írása (elég lesz az írási quorum adatpéldányainak írása), így az egymás utáni írások eredményeit (az újabb adatértékek azonosításához) verziószámokkal látjuk el: új értékek írása növekvő verziószámmal történik.

Implementáció egy adott partícióban:

- Olvasás: $Read(x) = \{Read(x_a), Read(x_b), \ldots, Read(x_m)\}$, ahol
  - az olvasáshoz szükséges feltétel, hogy az olvasott példányok olvasási quorumot képezzenek,
  - az olvasás eredménye a legnagyobb verziószámú adatpéldány olvasásának eredménye (ezt kell kikeresni és használni).
- Írás: $Write(x) = \{Write(x_1), Write(x_2), \ldots, Write(x_k)\}$, ahol
  - az íráshoz szükséges feltétel, hogy az írt példányok írási quorumot képezzenek,
  - a verziószámok kezelése három lépésben történik:
    1. Minden példány verziószámát be kell olvasni az írási quorumból,
    2. A legnagyobb verziószámot ki kell keresni és meg kell növelni eggyel,
    3. Az új adatot ezzel a növelt verziószámmal kell írni az írási quorum minden példánya esetén.

Előnyök:

- Akár átlapolt partíciók is kezelhetők (mindig a legfrissebb adat olvasható az olvasási quorum és a verziószámok használata miatt).
- Egyszerűen változtathatók a küszöbértékek (a helyreállítás után „érvénytelen” bejegyzéssel rendelkező példányok nem számítanak a küszöbértékbe, míg a felülírás új verziószámmal meg nem történik).

Hátrányok:

- $Read(x)$ esetén több példány olvasása szükséges
- $Write(x)$ esetén a verziószám kezelése olvasásokat is igényel

Egy példa a quorumok alakulására:

- Példányok: x_1, x_2, x_3, súlyok: $s_1 = 3, x_2 = 2, x_3 = 1, S = 6$
- Küszöbértékek: $WT = 4\ (2WT > S), RT = 3\ (RT + WT > S)$
- Írási quorumok: $\{x_1, x_2, x_3\}, \{x_1, x_2\}, \{x_1, x_3\}$
- Olvasási quorumok: $\{x_1, x_2, x_3\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_1\}$
- Van közös elem az írási-írási, illetve írási-olvasási quorumokban.

7 Virtuális partíciók

A virtuális partícióknak nevezett módszer esetén egyszerűsödik az írási és olvasási műveletek végrehajtása a quorum konszenzushoz képest. Ennek az az ára, hogy a partíció (hosztok egy elérhető halmaza) változásakor az adatpéldányok költséges frissítésére van szükség a további működéshez. Akkor jól használható a módszer, ha erre ritkán kerül sor.

Az aktuális partícióról, azaz az elérhető hostokról szükséges a tagsági kép nyilvántartása minden host esetén. A partíció változására a következők miatt kerülhet sor:

- Egy eddig a partícióban szereplő b host elérhetetlenné válik (ennek oka lehet kommunikációs hiba vagy a b host meghibásodása).
  Detektálható, ha az *b* hostra vonatkozó írás vagy olvasás művelet sikertelen lesz. Ilyenkor az összetett műveletet meg kell szakítani (Abort) és el kell végezni a partíció frissítését.
- Egy eddig a partícióban nem szereplő c host elérhetővé válik (pl. az összeköttetés helyreáll, vagy c host hiba utáni helyreállítása megtörténik).
  Detektálható, ha kérés érkezik c-től. Ilyenkor is el kell végezni a partíció frissítését.

Az írás és olvasás implementációja: ”*Minden példány írása*” algoritmus szerint történik a partícióban:

- Ugyanúgy használandók a küszöbértékek és quorumok, mint a quorum konszenzus módszer esetén (szükséges feltételt adnak meg az íráshoz és olvasáshoz).
- Olvasás: $Read(x) = Read(x_i)$ azaz egy példány olvasása elegendő
  - akkor végrehajtható, ha van olvasási quorum a partícióban;
- Írás: $Write(x) = \{Write(x_1), Write(x_2), \ldots, Write(x_k)\}$, azaz a partíció minden példányát írni kell (ha ez nem biztosítható, akkor partíció frissítés szükséges).
  - akkor végrehajtható, ha van írási quorum a partícióban;
  - verziószámok kezelése a $Write(x)$ során: itt az írás előtt *egy* példány verziószámának kiolvasása és növelése szükséges, majd az új értékek írása ezzel a növelt verziószámmal.

A partíció frissítése:

- A partíció megváltozását detektáló host indítja az új partíció (tagsági kép) kialakítását, azaz felszólítja a hostokat az általa kezdeményezett partícióhoz való csatlakozásra. Egy host csak egy partícióhoz csatlakozhat.
    - Jellegzetes megvalósítás: A partíciókat azonosítószámmal látják el. A kezdeményező új (a nála lévőnél nagyobb) azonosítószámmal kezdeményez. A hostok a náluk tároltho képest növelt azonosítószámmal kezdeményezett partícióhoz csatlakozhatnak, különben elutasítják a felkérést. Ha a kezdeményező elutasítást kap, növelt azonosítószámmal újra kezdheti a partíció kialakítását.
    - A kialakuló új partícióról konszenzus szükséges a csatlakozó hostok között.
- Minden *adatpéldányt frissíteni kell* az új partícióban, amire olvasási quorum van:
    - be kell olvasni minden adatpéldányt és a legnagyobb verziószámút kikeresni,
    - a legnagyobb verziószámú alapján frissíteni az összes adatpéldány értékét és verziószámát.

Tulajdonságok:

- $Read(x)$ esetén egy példány olvasása elég;
  ennek ára, hogy új partíció kialakításakor szükséges minden példány frissítése a legnagyobb verziószám alapján (viszont erre csak viszonylag ritkán, a partíció változása esetén van szükség).
- $Write(x)$ verziószámokat kezel, és továbbra is minden példányt írni kell az írási quorumban.