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) végzése: Read, Write részműveletek, lezárás Commit vagy Abort döntéssel.
- *Egypéldányos sorosíthatóság*: A többpéldányos esetben a részműveletek konkurens végrehajtásának eredménye megegyezik az egypéldányos eset egy lehetséges soros végrehajtásának eredményével.
- Adatpéldány kiesés (host kiesés) tolerálása, egyes esetekben emellett kommunikációs hiba 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 esetén is alkalmazható:
  - *Hostok többsége* (site quorum)
  - *Quorum konszenzus* (quorum consensus)
  - *Virtuális partíciók* (virtual partitions)

2 Minden példány írása

Implementáció:

- A legközelebbi (leggyorsabban elérhető) példány olvasása: $Read(x) = Read(x_i)$
- 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.

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

Az adatbázis konzisztenciája nem biztosított:

- A meghibásodás után helyreállított hostok érvénytelen (régi) adatot tartalmaznak. Ez a probléma kezelhető egy ”érvénytelen” jelzéssel, amit helyreállításkor kell az adatpéldányokra bejegyezni, és egy-egy adatpéldány legközelebbi felülírásakor törölhető.

- Egypéldányos sorosíthatóság nem érvényesül, 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 tranzakció indul:

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

  - Az x adaton és az y adaton is két tranzakció dolgozik, a sorosíthatósági konfliktust a hostok lokális ütemezője venné észre.
  - A meghibásodás miatt az a host lokális ütemezője nem detektálhatja a konfliktust; az adatpéldányok kiesése miatt az egyik hoston csak olvasás, a másik hoston csak írás történik.
  - Az eredmény soros végrehajtás esetén nem állhat elő (az esetben vagy T1 olvassa a T2 által írt értéket, vagy viszont).

4 Elérhető példányok írása validációval

Az olvasás és az írás megvalósítása:

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

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

Sorosíthatósági konfliktusok kezelése: Validációs protokoll a Commit előtt:

- A hiányzó írások validációja: A meghibásodott példányok továbbra sem elérhetők:
  - $Unavailable(x_k)$ üzenet minden meghibásodott hostnak
  - ha visszajelez, hogy azóta inicializálva van, akkor Abort
- Hozzáférés validációja: Az írt példányok továbbra is elérhetőek:
  - $Available(x_k)$ üzenet minden írt példánynak
  - Commit csak akkor, ha mindegyik visszajelez

Validációs protokoll célja és megvalósítása:

- Elkerülhető az ”elérhető példányok írása” módszer esetén tapasztalt sorosíthatósági probléma
- A hiányzó írások validációjára csak akkor van szükség, ha volt hibás példány
- Hozzáférés validációja a döntési protokollal együtt megtörténhet

5 A partíciók problémája

Alapvető probléma a korábbi megoldások esetén: Az elosztott rendszer partíciókra eshet szét, az egyes partíciókban függetlenül folyhatnak a tranzakciók.

Cél: Csak egy partícióban (aktív partíció) lehessen használni az adatot. Aktív partíció lehet:

- ahol a hostok többsége megtalálható, vagy
- ahol a példányok többsége megtalálható az adott adatra nézve

Egy implementációs lehetőség:

- Súlyok vannak az egyes hostokhoz rendelve
- Aktív partíció: ahol a hostok súlyainak többsége található
- Az aktív partícióban az ”elérhető példányok írása validációval” módszer használható

Problémák:

- Sok partíció: egyik sem lehet aktív
- Ha a partíciókban vannak közös hostok (átlapolt partíciók), akkor inkonzisztens hozzáférés lehetséges:
  P1: x_a, x_b; T1: $Read(x_a), Write(x_a), Write(x_b), Commit$
  P2: x_a, x_c; T2: $Read(x_c), Write(x_a), Write(x_c), Commit$
  (x_a mindkét partícióban megtalálható; különböző adatpéldányok lesznek a két tranzakció végén)

6 Quorum konszenzus

A quorum (határozatképesség) definíciója:

- 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 példányok 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 példányok R halmaza *olvasási quorumot* képez, azaz $\sum_{x_l \in R} s_l \geq RT$
- A küszöbértékek megválasztására a következő feltételek betartása szükséges:
  - $2WT > S$
  - $WT + RT > S$
- Következmények: Minden írási quorumnak van közös eleme
  - $2WT > S$ miatt bármely másik írási quorummal (ez lehetővé teszi az egymás utáni írások adminisztrálását, ld. később)
  - $WT + RT > S$ miatt bármely olvasási quorummal (ez lehetővé teszi a legutolsó írás eredményének kiolvasását)

Implementáció az aktív partícióban:

- Verziószámok kezelése: Az új adatok írása növekvő verziószámmal történik.

- Olvasás: $Read(x) = \{Read(x_a), Read(x_b), \ldots, Read(x_m)\}$, ahol
  - az olvasott példányok olvasási quorumot képeznek (egyébként Abort, ha nincs ilyen),
  - 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 írt példányok írási quorumot képeznek (egyébként Abort, ha nincs ilyen)
  - a verziószámok kezelése három lépésben történik:
    1. Minden példány verziószámát be kell olvasni az írási quorumból,
    2. A legnagyobb verziószámot meg kell növelni,
    3. Az új adatot ezzel a növelt verziószámmal kell írni minden példány esetén az írási quorumban.

Előnyök:

- Átlapolt partíciók kezelése lehetséges (mindig a legfrissebb adat olvasható az olvasási quorum és a verziószámok használata miatt)
- Automatikus helyreállítás lehetséges (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
- Hibatűrés nem igazán hatékony (hasonló a többségi szavazáshoz)
- Új példány integrálása esetén a küszöbértékeket újra kell számolni

Egy példa:

- 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: $\{x_1, x_2, x_3\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_1\}$

Az előző algoritmusban szereplő problémás eset kezelése: van közös elem az írási-olvasási quorumokban.

7 Virtuális partíciók

A partíciók helyett itt a *nézet* (view) definícióját használják:

- A nézet definíciója:
  - egy A host $v(A)$ nézete: Az A-ból elérhető hostok halmaza (egy host csak egy nézethez tartozhat)
  - egy T tranzakcióhoz tartozó $v(T)$ nézet: a T-t indító host nézete

- Egy $v(a)$ nézet változásai, amelyek a nézet frissítéséhez vezetnek:
  - Egy eddig a nézetben szereplő b host elérhetetlenné válik (ennek oka lehet az a és b közötti összeköttetés megszakadása vagy a b host meghibásodása).
    Detektálható: írás vagy olvasás művelet sikertelen lesz, tranzakció Abort
  - Egy eddig a nézetben nem szereplő c host elérhetővé válik (az összeköttetés a és c között helyreáll, vagy c újraindul).
    Detektálható: kérés érkezik c-től.

Az írás és olvasás implementációja: "*Minden példány írása*" algoritmus szerint a $v(T)$ nézetben, ahol

- Quorum definíciója: mint a quorum konszenzus esetén (súlyok, küszöbértékek használatával)
- Olvasás: $Read(x) = Read(x_i)$ azaz egy példány olvasása elegendő
  - akkor végrehajtható, ha van olvasási quorum $v(T)$-ben;
- Írás: $Write(x) = \{Write(x_1), Write(x_2), \ldots, Write(x_k)\}$, azaz a nézet minden példányát írni kell (ha ez nem biztosítható, akkor Abort és nézet frissítés szükséges).
  - akkor végrehajtható, ha van írási quorum $v(T)$-ben;
  - verziószámok kezelése a $Write(x)$ során történik: egy példány olvasása, a hozzá tartozó verziószám növelése és az új értékek írása ezzel a verziószámmal

A $v(t)$ nézet változásainak kezelése: *Nézet frissítés*

- Alapfeladatok:
  - minden helyen frissíteni kell a nézetet, új azonosítóval (VID a nézet azonosítója)
  - minden *adatpéldányt frissíteni kell* az új nézetben (pl. ha host elérhetővé válik).
- Az a host indítja, amely a nézet változását detektálja (ld. fentebb)
- A protokoll lépései:
  1. Ha egy a host detektálja a nézet változását, akkor egy új $newVID > VID$ nézet azonosítót generál
  2. $Join\_View(newVID, a)$ üzenetet küld az elérhető hostoknak
  3. Egy b host elfogadja ezt az üzenetet, ha $VID_b < newVID$; egyébként elutasítja
  4. Ha az a host elutasítást kap, akkor nagyobb $newVID$-vel újrakezdi a protokollt
  5. Ha nincs elutasítás, akkor az a host kialakítja a new_view új nézetet (az elfogadó hostok listája), és elküldi az új nézetet ezeknek a hostoknak
  6. Minden host beállítja az új nézetet és annak új azonosítóját
  7. Legfontosabb lépés: Minden adatra, amire olvasási quorum van az új partícióban:
     be kell olvasni az adatpéldányokat és a legnagyobb verziószám alapján frissíteni ezeket.

Tulajdonságok:

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