Adatbázisok elmélete 21. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu
http://www.cs.bme.hu/˜kiskat

2004

Holtpont

Láttuk, hogy az ütemező úgy kényszeríti ki a legális ütemezést, hogy várakoztatja a tranzakciókat. Ebből problémák lehetnek, ha a tranzakciók egymásra várnak:

Holtpont (deadlock, patt): néhány zárkérés után akkor van holtpont, ha van egy olyan részhalmaza a tranzakcióknak, akik közül egyik se tud tovább futni, mert vár egy szintén ebben a részhalmazban levő másikra (vár egy olyan zár elengedésére, amit egy másik, ebbe a részhalmazba tartozó, tranzakció tart).

Például:

$l_1(A),\ \ l_2(B),\ \ l_3(C),\ \ l_1(B),\ \ l_2(C),\ \ l_3(A)$

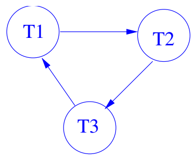
sorrendben érkező zárkérések esetén egyik tranzakció se tud tovább futni.

Az ilyen helyzeteket el kell kerülni, illetve ha már kialakultak, akkor fel kell ismerni és meg kell szüntetni.

Várakozási gráf

A felismerésben segít a zárkérések sorozatához tratozó **várakozási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha T_i vár egy olyan zár elengedésére, amit T_j tart éppen.

Például az előbbi, holtponthoz vezető zárkéréssorozat várakozási gráfja a hat zárkérés után:



Vegyük észre, hogy a várakozási gráf változik az ütemezés során, ahogy újabb zárkérések érkeznek vagy zárelengedések történnek.

Holtpont felismerése

A várakozási gráf segítségével fel lehet ismerni a holtpontot az alábbi tétel miatt:

Tétel. *Az ütemezés során egy adott pillanatban pontosan akkor nincs holtpont, ha az adott pillanathoz tartozó várakozási gráf DAG (nincs benne irányított kör).*

Bizonyítás: $\Rightarrow$: Ha van irányított kör a várakozási gráfban, akkor a körbeli tranzakciók egyike se tud lefutni, mert vár a mellette levőre. Ez holtpont.
$\Leftarrow$: Ha a gráf DAG, akkor van topológikus rendezése a tranzakcióknak (lásd Algel) és ebben a sorrendben le tudnak futni a tranzakciók. (Az első nem vár senkire, mert nem megy belőle ki él, így lefuthat; ezután már a másodikba se megy él, az is lefuthat . . .)

Példa

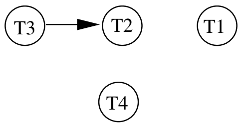
Nézzük az alábbi írásokból és olvasásokból álló ütemezést:

$$r_1(A),\quad r_2(B),\quad w_1(C),\quad r_3(D),\quad r_4(E),\quad r_3(B),\quad w_2(C),\quad w_4(A),\quad w_1(D)$$

Tegyük fel, hogy a zárkérések mindig közvetlenül megelőzik a műveletet, a zárelengedések pedig a tranzakciók végén, egyszerre történnek. Hogyan alakul a várakozási gráf ezen sorozat esetén? Lesz-e valamikor holtpont?

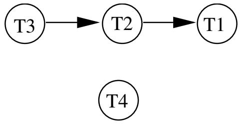
Az elején $l_1(A)$, $r_1(A)$, $l_2(B)$, $r_2(B)$, $l_1(C)$, $w_1(C)$, $l_3(D)$, $r_3(D)$, $l_4(E)$, $r_4(E)$ zárkérések és műveletek vannak, eddig még senki nem vár senkire.

Ezután $l_3(B)$ jön $r_3(B)$ miatt, de T_3-nak várnia kell T_2-re:

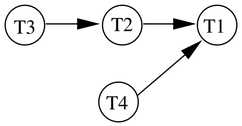


Példa

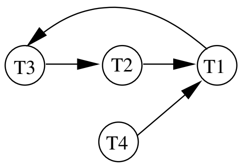
Ezután $l_2(C)$ jön $w_2(C)$ miatt, de T_2-nek is várnia kell T_1-re:



Ezután $l_4(A)$ jön $w_4(A)$ miatt, de T_4-nek is várnia kell T_1-re:



Végül $l_1(D)$ jön $w_1(D)$ miatt, de T_1-nek meg T_3-ra kell várnia:



És ez már holtpont: van irányított kör a gráfban.

Megoldások holtpont ellen

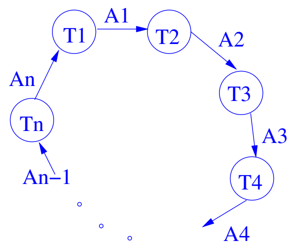
1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben.
Ez egy megengedő megoldás (optimista), hagyja az ütemező, hogy mindenki úgy kérjen zárat, ahogy csak akar, de ha baj van, akkor erőszakosan beavatkozik. Az előző példa esetében mondjuk kilövi T_2-t, ettől lefuthat T_3, majd T_1 és T_4 is.

2. Pesszimista hozzáállás: ha hagyjuk, hogy mindenki össze-vissza kérjen zárat, abból baj lehet. Előzzük inkább meg a holtpont kialakulását valahogyan. Lehetőségek:

(a) Minden egyes tranzakció előre elkéri az összes zárat, ami neki kelleni fog. Ha nem kapja meg az összeset, akkor egyet se kér el, el se indul.
Ilyenkor biztos nem lesz holtpont, mert ha valaki megkap egy zárat, akkor le is tud futni, nem akad el. Az csak a baj ezzel, hogy előre kell mindent tudni.

(b) Feltesszük, hogy van egy sorrend az adategységeken és minden egyes tranzakció csak eszerint a sorrend szerint növőleg kérhet újabb zárakat. Itt lehet, hogy lesz várakozás, de holtpont biztos nem lesz.

Bizonyítás: Ha valamely pillanatban lenne irányított kör a várakozási gráfban:



ahol T_i vár T_{i+1}-re az A_i adategység miatt, akkor $A_1 < A_2 < A_3 < \ldots A_n < A_1$ áll fenn abban az esetben, ha mindegyik tranzakció betartotta, hogy növőleg kér zárat. Ez azonban ellentmondás.
Tehát ez a protokoll is megelőzi a holtpontot, de itt is előre kell tudni, hogy milyen zárakat fog kérni egy tranzakció.

Még egy módszer, ami szintén optimista, mint az első:
Időkorlát alkalmazása: ha egy tranzakció kezdete óta túl sok idő telt el: ABORT.
Ehhez az kell, hogy ezt az időkorlátot jól tudjuk megválasztani.

Éhezés, zárak és sorosíthatóság

Másik probléma, ami zárakkal kapcsolatban előfordulhat: **éhezés**: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.
Megoldás: adategységenként FIFO listában tartani a várakozókat

Eddig azt láttuk csak, hogy mennyi baj lehet a zárak alkalmazásával (holtpont, éhezés). Most nézzük, hogy mire jók a zárak.

A zárak segítségével el lehet majd érni, hogy az ütemezések sorosíthatók legyenek, de pusztán az, hogy használjuk a zárakat, még nem ad sorosítható ütemezést.

Példa olyan legális, zárakat használó ütemezésre, ami nem sorosítható: a korábbi, nem sorosítható, írásokból és olvasásokból álló ütemezésbe zárakat rakunk:

$$l_1(A),\ r_1(A),\ w_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A),\ u_2(A),$$
$$l_2(B),\ r_2(B),\ w_2(B),\ u_2(B),\ l_1(B),\ r_1(B),\ w_1(B),\ u_1(B)$$

Sorosíthatóság és zárak

Zárakat használunk, figyelünk arra, hogy legális legyen az ütemezés, és még figyelünk valamire, ami biztosítja a sorosíthatóságot.

Egyszerű tranzakció modellben vagyunk (egy fajta zár van csak és a korábbi három feltevés mindig fennáll, azaz a zárkérés legális) és még valamit felteszünk:
A sorosíthatóságról pusztán a zárkérések alapján fogunk dönteni, nem nézzük azt, hogy ezeken kívül milyen műveletek (írások/olvasások) vannak. Pontosabban:

Nem foglalkozunk azzal, hogy $LOCK_i(A)$ és $UNLOCK_i(A)$ között mi történik, feltesszük hogy valami teljesen egyedi írás és olvasás is. Ez hasonlít ahhoz a helyzethez, mint amikor a konkrét számolásokat elhanyagoltuk: feltesszük, hogy a lehető legrosszabb történik azalatt, amíg a tranzakciónál van a zár.

Így persze megint igaz lesz az, hogy olyan ütemezéseket is rossznak minősítünk, amik igazából sorosíthatók lennének, ha megnéznénk, hogy írások vagy olvasások történnek, de ez nem baj, mert szigorúbbak lehetünk, csak az a fontos, hogy olyan ne legyen sorosíthatónak minősítve, aki nem az.

Sorosítási gráf az egyszerű tranzakciómodellben

Az előbbiek értelmében tehát egy olyan legális ütemezésről akarjuk eldönteni, hogy sorosítható-e, amiben csak zárkérések és zárelengedések vannak.

Mikor lesz egy ilyen ütemezés sorosítható, függetlenül attól, hogy milyen írások és olvasások történnek valójában?

Ennek megválaszolásában segít a **sorosítási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha az ütemezésben van olyan $u_i(A)\ \ldots\ l_j(A)$ rész, ahol $u_i(A)$ (T_i elengedi A zárját) és $l_j(A)$ (T_j megkapja A zárját) között A-ra senki se kap zárat.

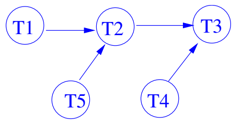
Ekkor minden olyan soros ütemezésben, ami ekvivalens lehet a miénkkel, biztos, hogy T_j-nek T_i után kell jönnie.
Ez azért van így, mert feltettük, hogy T_i is és T_j is bármit csinálhat A-val, amíg nála van a zár és ha pl. T_i írja, T_j meg olvassa A-t, akkor már csak a $T_i, \ldots, T_j$ sorrend lesz a jó.

Példa sorosítási gráfra

Az alábbi, csak zárkéréseket és zárelengedéseket tartalmazó ütemezés legális (HF: leellenőrizni):

$$l_5(A),\ l_1(B),\ u_5(A),\ l_4(C),\ u_1(B),\ l_2(A),\ l_2(B),\ u_2(A),$$
$$l_3(A),\ u_3(A),\ u_4(C),\ u_2(B),\ l_3(C),\ u_3(C)$$

Az ehhez tartozó sorosítási gráf:



Tétel a sorosítási gráfról

Tétel. *Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt sorosítási gráf DAG.*

Bizonyítás: ⇒: Ha nem DAG a gráf, akkor van benne irányított kör. Ebben a körben levő tranzakciók közül egyik sem előzheti meg a többit egy ekvivalens soros ütemezésben, amiből következik, hogy nincs ekvivalens soros ütemezés.

⇐: Teljes indukcióval: $n = 1$-re (1 tranzakció van csak) világos, egy ilyen ütemezés maga soros.
Legyen most az ütemezésben n tranzakció. Ha a gráf DAG, akkor létezik topologikus rendezése. Azt látjuk be, hogy a topologikus sorrend szerinti soros ütemezés ekvivalens lesz az eredeti ütemezéssel.
Ha T_i a topologikus rendezés szerinti első tranzakció, akkor nem megy bele él, vagyis ő csak olyan adategységeket használ, amiket az eredeti ütemezésben előtte senki. Így az ő összes utasítását előre mozgathatjuk, a hatás nem változik.
Ami ezután marad, az $n - 1$ tranzakció utasításaiból álló ütemezés, aminek a sorosítási gráfja szintén DAG, tehát ennek az indukció szerint létezik soros ekvivalense (a maradék tranzakciók topologikus sorrendjének megfelelően), ami T_i-vel kiegészítve soros ekvivalense lesz az eredetinek.

Következmény

Következmény: A bizonyításból látszik, hogy a soros ekvivalensek és a lehetséges topologikus sorrendek megfelelnek egymásnak, vagyis annyi soros ekvivalens lesz, ahány különböző topologikus sorrend van.

Például a korábban látott sorosítási gráf esetén 8 darab topologikus sorrend van, így nyolc soros ekvivalens van:

$T_5\ \ T_4\ \ T_1\ \ T_2\ \ T_3,$
$T_4\ \ T_5\ \ T_1\ \ T_2\ \ T_3,$
$T_4\ \ T_1\ \ T_5\ \ T_2\ \ T_3,$
$T_5\ \ T_1\ \ T_4\ \ T_2\ \ T_3,$
$T_1\ \ T_5\ \ T_4\ \ T_2\ \ T_3,$
$T_1\ \ T_4\ \ T_5\ \ T_2\ \ T_3,$
$T_5\ \ T_1\ \ T_2\ \ T_4\ \ T_3,$
$T_1\ \ T_5\ \ T_2\ \ T_4\ \ T_3,$

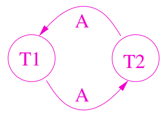
Példa, ami mutatja, hogy szigorúbbak vagyunk a kelleténél

Tekintsük az

$$l_1(A),\ \ r_1(A),\ \ u_1(A),\ \ l_2(A),\ \ r_2(A),\ \ u_2(A),\ \ l_1(A),\ \ w_1(A),\ \ u_1(A),\ \ l_2(B),\ \ r_2(B),\ \ u_2(B)$$

ütemezést.
Ha megnézzük az írás/olvasás műveleteket $(r_1(A),\ \ r_2(A),\ \ w_1(A),\ \ r_2(B))$, akkor látszik, hogy az ütemezés hatása azonos a T_2T_1 soros ütemezés hatásával, vagyis ez egy sorosítható ütemezés.

De ha felrajzoljuk a sorosítási gráfot (és ilyenkor persze nem nézzük, hogy milyen írások/olvasások vannak, hanem a legrosszabb esetre készülünk), akkor



lesz a gráf, és mivel ez nem DAG, ezért nem lesz sorosítható az az ütemezés, amiben már csak a zárak vannak benne.

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
   (Előny: nem óvatoskodik, nem korlátoz feleslegesen;
   Hátrány: drasztikus megoldás az ABORT)
2. Protokollt ír elő a tranzakciók számára, amit minden egyes tranzakciónak be kell tartania:
   2PL (two-phase locking, kétfázisú protokoll): a T_i tranzakció követi a kétfázisú protokollt, ha $UNLOCK_i$ után nincs $LOCK_i$, azaz ha nem kér már zárat miután elengedett már egyet.

   Tétel. *Ha az egyszerű tranzakciómodellbeli legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.*

Bonyolultabb zármodellek

Többfajta zár van, aszerint, hogy a tranzakciók mit akarnak csinálni az adattal. (Persze akkor, ha van több különböző művelet, nem csak írás és olvasás.)

Cél, hogy minél jobban tükrözzék a zárkérési lehetőségek a lehetséges műveleteket, mert így kevesebb lesz a várakozás (több olyan eset lesz, amikor lehet két tranzakciónak zárja ugyanott, ha a megfelelő műveletek mehetnek együtt) és megalapozottabb lesz a döntés a sorosíthatóságról (nem leszünk annyira feleslegesen szigorúak).

Példa: Legyen három művelet: olvasás, írás és növelés (increment).
Ez utóbbi azt jelenti, hogy az adategység aktuális értékét megnöveljük eggyel.

Ekkor bevezethetünk három zárat: RLOCK, WLOCK és INCR, a kézenfekvő használattal (a megfelelő művelet csak akkor mehet, ha a tranzakció megkapta a hozzá tartozó zárat).

Kompatibilitási mátrix

Egy mátrix segítségével adjuk meg, hogy különböző tranzakcióknak milyen zárai lehetnek egyszerre egy adategységen.
Ez a **kompatibilitási mátrix**: a sorok és az oszlopok is a lehetséges záraknak felelnek meg és a Z_i sor Z_j oszlopában pontosan akkor van `I`, ha egy tranzakció megkaphatja egy adategységre a Z_j zárat akkor, ha egy másik tranzakció Z_i zárat tart fenn ezen az adategységen. Ha nem kaphatja meg, akkor `N` áll a Z_i sor Z_j oszlopában.

Akkor lehet két különböző tranzakciónak Z_i és Z_j zárja ugyanazon az adategységen, ha mindegy, hogy a két zárnak megfelelő műveletek milyen sorrendben hajtódnak végre.

Ez alapján az RLOCK/WLOCK modell és az RLOCK/WLOCK/INCR modell mátrixai:

	RLOCK	WLOCK
RLOCK	I	N
WLOCK	N	N

	RLOCK	WLOCK	INCR
RLOCK	I	N	N
WLOCK	N	N	N
INCR	N	N	I

A kompatibilitási mátrix használata

1. Ez alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az `I` a mátrixban, annál kevesebb lesz a várakoztatás.
2. A mátrix alapján keletkező várakozásokhoz elkészített várakozási gráf segítségével az ütemező kezeli a holtpontot (ami tetszőleges zármodell esetén ugyanazt jelenti és a gráfot is ugyanúgy kell felépíteni).
3. A mátrix alapján készíti el az ütemező a sorosítási gráfot egy zárkérés-sorozathoz: a sorosítási gráf csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha van olyan A adategység, amelyre az ütemezés során Z_k zárat kért és kapott T_i, ezt elengedte, majd ezután A-ra legközelebb T_j kért és kapott Z_l zárat és a mátrixban a Z_k sor Z_l oszlopában `N` áll.
   Vagyis olyankor lesz él, ha a két zár nem kompatibilis egymással, nem mindegy a két művelet sorrendje.

Sorosíthatóság

A sorosíthatóságról most is pusztán a zárkérések alapján fogunk dönteni, a sorosítási gráf segítségével:
Tétel. *Egy csak zárkéréseket és zárelengedéseket tartalmazó legális ütemezés pontosan akkor sorosítható valamelyik zármodellben, ha a zármodellhez tartozó kompatibilitási mátrix alapján az előbbi módszerrel felrajzolt sorosítási gráf DAG.*

Bizonyítás: Pontosan ugyanúgy megy, ahogyan eddig.

Az ütemező egyik lehetősége a sorosíthatóság elérésére, hogy folyamatosan figyeli a sorosítási gráfot és ha irányított kör keletezne, akkor ABORT-ot rendel el.

Sorosíthatóság II.

Másik lehetőség a protokollal való megelőzés. Tetszőleges zármodellben értelmes a 2PL és igaz az alábbi tétel:

Tétel. *Ha valamilyen zármodellben egy legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.*

Bizonyítás: Pontosan úgy, ahogy eddig.

Megjegyzés: Minél gazdagabb a zármodell, minél több az `I` a kompatibilitási mátrixban, annál valószínűbb, hogy a sorosítási gráf DAG lesz minden külön protokoll nélkül. Ez azt jelenti, ilyenkor egyre jobb lesz az ABORT-os módszer (ritkán kellhet).

Mit látunk mi ebből?

Az adatbáziskezelő működése során az ütemező munkájába nem (nagyon) szólhatunk bele. Miért hasznos mégis tudni, hogy hogyan működik?

- Abba beleszólhatunk, hogy mennyire törekedjen sorosíthatóságra az adatbáziskezelő (akár azt is mondhatjuk, hogy semennyire). Ehhez nem árt, ha tudjuk, hogy mi is a sorosíthatóság, mit nyerünk vele és mibe kerül (bonyolult ütemező, lassabb futás).
- Ha ismerjük a különféle ütemezési technikákat: jobban fogjuk érteni az előforduló ABORT-okat, és majd az ABORT utáni visszaállítást is.