Page Locking Via Design

av
Mathias Magnusson Juli 2008
Översatt Mars 2009

Innehållsförteckning

Introduktion

Vi har alla sett lösningar som i sig själva är dåliga, men när de kombineras med att de motarbetar den teknologi som används så blir de en riktig katastrof. I det här dokumentet skall vi titta på en sådan situation som författaren har sett i lite variation under de senaste åren. Vi börjar med att beskriva den lösning som ofta används och de databas koncept som ligger bakom problematiken.

Designproblemet

Det är ganska vanligt i de system jag träffar på att komplicerade transaktioner har designats så att man börjar med att skapa en ny rad i en tabell och sedan går man tillbaka och uppdaterar den här raden vartefter data fylls i av användaren. Det här är givetvis en algoritm som i sig kan förbättras, men problemet kommer ifrån att man från början i stort sett skapar raden genom att bara ange primärnyckeln och resterande fält uppdateras senare. Ifall det här var slutet på problemet skulle det vara illa nog och en bättre lösning skulle troligen behövas för att systemet skall ha god prestanda. Tyvärr sker det sällan då det ser ut som en stor ändring utan istället försöker man jobba runt det genom att ändra andra saker.

Vad som gör den här designen ännu värre är ett koncept Oracle kallar ITL. ITL står för Interested Transaction List and är en del i hur Oracle hanterar radlås. Varje transaktion som uppdaterar en rad i ett block läggs in i blockets ITL. I senare versioner av Oracle växer den listan dynamisk så länge det finns plats i blockets header section. Skulle listan vara full och blocket inte har plats att utöka listan så väntar transaktionen på att en plats blir ledig.

Som resultat av detta kommer du få en låsning på sidnivå om ITL är full och den inte kan utökas. Dvs, även om den rad du vill uppdatera inte uppdateras av någon annan transaktion så kommer din transaktion att vänta på att en transaktion som uppdaterar en annan rad i samma block avslutas.

Som tur är så inträffar inte den här situationen så ofta eftersom rader ofta är så stora att det för det mesta finns utrymme kvar i blocket, en ny rad får inte plats men en ny ITL post får plats.

Det finns ytterligare ett databaskoncept som påverkar det här problemet. Row migration inträffar när en rad uppdateras så att den tar mer plats och det inte finns tillräckligt med plats där den är. Ju mer raden växer med, desto större är förstås risken att den behöver flyttas till ett annat block. Den designsituation vi har diskuterat här är förstås sådan att rader ofta kommer var några få bytes från början och sedan växer de genom en eller flera uppdateringar till att vara flera hundra bytes stora. Men varför är det här ett problem? En rad som flyttas kommer väl i framtiden skötas av en ITL på det block den flyttats till? Samma rad får väl samtidigt ett nytt rowid så endast referenser till var raden brukade finnas påverkar det block raden låg på från början?

Det visar sig att det inte alls fungerar på det viset, utan istället så ligger en pekare i det gamla blocket till radens nya position. Det är kanske inte så konstigt, men vad som är lite förvånande är att raden fortsätter hanteras av det block den var på rån början. Dess rowid ändras inte heller, utan fortsätter vara vad det var från början. Det här får effekten att om du skapar 600 rader små rader i ett block som sedan uppdateras och flyttar på sig så kommer ITL lås fortsätta tas på det ursprungliga blocket. Som resultat så har vi nu ett block som fungerar som ett index block och kommer ha mycket aktivitet och många lås. Tyvärr har alla de små ursprungliga raderna orsakat att ITL inte kan växa då det inte finns något utrymme kvar i blocket.

Nu tittar vi på ett konstruerat testfall som visar just den här effekten.

Testfallet

Det här testfallet testades på en Oracle 11gR1 databas, men bör fungera på alla versioner. Jag är inte medveten om något som är versionsspecifikt här (vet du eller märker du något så hör av dig så jag kan dokumentera det).

Först skapar vi en användare, du behöver vara inloggad som SYSTEM eller en annan DBA användare

```
create user itl identified by itl; grant create session, resource to
itl;
```

Nu skapar vi de två tabeller vi skall använda - inloggad som den användare vi just skapade (itl).

```
create table itl_test   (a number        not null   ,b varchar2(3000)
null   ,c varchar2(3000)    null)   pctfree 0; create table
chained_rows    (owner_name       varchar2(30),   table_name
varchar2(30),   cluster_name      varchar2(30),   partition_name
varchar2(30),   subpartition_name varchar2(30),   head_rowid
rowid,   analyze_timestamp date);
```

Tabellen itl_test är den tabell vi använder för att testa den effekt det här dokument handlar om. chained_rows är bara DDLen ifrån utlchn1.sql (använd utlchain.sql för versioner före 8.1) i databasens hembibliotek. Den tabellen använder vi för att kolla vilka rader som migrerade rader.

Tabellen är skapad med pct_free = 0, det är för att de till att testandet inte försvåras genom att vi hela tiden lämnar lite utrymme för framtida uppdateringar. Tabellen kommer också ha två ITL poster per block eftersom det är default. Effekten kan förstås fås med många fler ITL poster per block, det är bara en fråga om att ha fler samtidiga uppdateringar mot ett och samma block. Att ha fler ITL poster per block slösar bort plats i blocket i de fall då det inte behövs, men det kan vara en lösning för att minska ITL lås problem i produktionssystem.

Nästa steg är att är att fylla ett block med data. På mitt system innebär det att 660 rader skapas i det första blocket och sedan en rad i det andra. Den sista raden behövs egentligen inte, men är till så att vi vet att det först blockets fyllts med data.

```
declare
  rid         rowid;
  start_block number;
  new_block   number;
begin
     insert
       into itl_test
            (a,b,c)
     values (1,null,null)
  returning rowid
       into rid;

  start_block := dbms_rowid.
                 rowid_block_number(rid);

  for i in 2..2000 loop
       insert
         into itl_test
              (a,b,c)
       values(i,null,null)
    returning rowid
         into rid;

    new_block := dbms_rowid.
                 rowid_block_number(rid);

    if not start_block = new_block
    then
      exit;
    end if;
  end loop;
end;
/
commit;
```

Nu när vi har rader med id 1-660 (i ett 8 KB block i min databas) i det första blocket så är det dags att uppdatera så att de första tre raderna flyttas till tre anndra block via row migration.

```
update itl_test    set b = rpad('x', 3000, 'x')       ,c = rpad('x',
3000, 'x')   where a <= 3;
```

Varje rad uppdateras så att den är över 6000 bytes lång. Eftersom jag använder block som är 8 KB så får det bara plats ett sådant per block. Eftersom det första blocket redan är fullt så kommer de här raderna att flyttas till nya block. Vi kan verifiera att det verkligen hände med följande kod.

```
analyze table itl_test list chained rows;
select * from chained_rows;
select * from itl_test
 where rowid in (select head_rowid
                   from chained_rows);
```

De tre flyttade raderna kommer visas. Vi vet att uppdateringen placerade raderna på olika block, så vi har nu ett block med många små rader och tre block med en stor migrerad rad var.

Nästa steg är att uppdatera det tre migrerade raderna i tre olika sessioner.

```
session 1: update itl_test    set b='b'       ,c='c'  where a = 1;
session 2: update itl_test    set b='b'       ,c='c'  where a = 2;
session 3: update itl_test    set b='b'       ,c='c'  where a = 3;
```

Den tredje sessionen kommer nu hänga och vänta på att session 1 eller 2 avslutar sin transaktion. Anledningen till denna wait kan ses i v$session (i tidigare versioner kan man se det i v$session_wait).

```
select sid
      ,event
      ,blocking_session
      ,seconds_in_wait
  from v$session
 where event# = 201;
```

Den selecten kommer returnera den session som nu väntar på en ITL post att frigöras. Vi väntar alltså på en ITL post trots att de tre raderna är i olika block. Vi skulle förstås ha fått samma problem om vi uppdaterade raderna utan att ha migrerat dem till andra block, men då vore det mer självklart eftersom de då var i samma block.

Det är värt att notera att det inte finns någon korrelation mellan antalet rader vi uppdaterar och antalet rader som migrerades. Samma sak skulle hända om fler eller färre rader migrerades eller om vi uppdaterade en mix av migrerade och icke migrerade rader. Det enda som spelar någon roll för effekten är att det ursprungliga blocket inte har plats för ytterligare ITL poster och att vi uppdaterar fler än två rader som ursprungligen låg i det blocket.

En rimlig fråga efter att ha sett hur det här fungerade med de tre migrerade raderna är om samma sak hade hänt om vi hade gjort skapat dem stora nog att hamna på olika block från början. Rent logiskt sett borde det förstås inte hända, men det skadar aldrig att verifiera. Dessutom ger det oss en chans att se hur man fixar problem med migrerad data. Att verkligen fixa kräver förstås att man ändrar applikationen så att det inte inträffar i någon större grad, men här tittar vi bara på hur man kan få raderna att inte vara migrerade längre.

När vi körde analyze kommandot så lades en rad per migrerad rad in i tabllen chained_rows. En av kolumnerna i tabellen är head_rowid för som förstås leder oss tillbaka till en migrerad rad. Vi kan nu använda den kolumnen för att kopiera de migrerade raderna till en annan tabell, sedan ta bort dem i ursprungstabellen och därefter kopiera tillbaka dem. På så vis kommer raderna inte längre vara migrerade utan ser nu ut som om de lades till så store som de är nu.

```
create table itl_temp as   select *     from itl_test    where rowid
in (select head_rowid                      from chained_rows);  delete
from itl_test    where rowid in (select head_rowid
from chained_rows);  insert into itl_test   select * from itl_temp;
drop table itl_temp  delete from chained_rows; analyze table itl_test
list chained rows; select * from chained_rows;
```

Den sista selecten kommer nu inte returnera några rader, vilket förstås visar att vi inte har några migrerade rader kvar i tabellen. Om vi nu kör samma tre uppdateringar i tre olika sessioner kommer de nu fungera eftersom raderna nu hör hemma i tre olika block och inget ITL problem uppstår för att raderna en gång låg på samma block. Migreringen är alltså anledningen till att låsen markerades i ITL listan på det ursprungliga blocket.

Slutsats

Vad lärde vi oss av det här? En dålig design kan ibland få riktigt otrevliga effekter om teknologin som används gör sitt bästa för att plåga dig.

En sorts sidlås kan uppstå när en ITL lista inte kan utvidgas. En rads ursprungliga sida använder en ITL post när raden uppdateras även om raden har placerats på en annan sida sedan dess.

Vi lärde oss också att trista koncept som ITL faktiskt kan vara riktigt häftiga, åtminstone om du är en bitpulande databasfanatiker som uppskattar att lära dig mer om hur Oracle har fått databasen att fungera med alla de avancerade koncept som vi tar för givet.