Oracle Row Locking nader bekeken

Veel voorkomend applicatieprobleem

In de meeste Oracle Applicaties is Row Level locking een van de meest voorkomende synchronisatieproblemen. Andere synchronisatie problemen zijn bijvoorbeeld latch en enqueue contentie. Row Lock contentie is normaal gesproken een applicatieprobleem, maar er zijn echter een aantal situaties die weliswaar gelijkenis vertonen met Row Lock contentie, maar het eigenlijk niet zijn. DBA'ers en ontwikkelaars kunnen daardoor een hoop tijd verliezen door te zoeken naar applicatie row lock problemen die eigenlijk in en door de Oracle Kernel worden veroorzaakt.

De TX lock (of enqueue) wordt door de Oracle kernel gebruikt om row-level locking te implementeren. De TX lock is een transactie lock die aan het begin voor elke transactie geal-loceerd wordt. Mode 4 betekent Shared Mode en Mode 6 betekent Exclusive Mode. Hier volgt een overzicht van de meest voorkomende situaties.

Het traditionele row lock probleem

Voor dit voorbeeld creëren we een tabel:

```
Create table test (col1 number, col2 varchar(100));
```

En voeren dan een tweetal records toe aan de tabel:

```
Insert into test values (1, 'AAA');
Insert into test values (2. 'BBB');
Commit;
```

Dan starten we twee sessies die elk record 1 gaan updaten:

```
Sessie 1:                          Sessie 2:

Update test set col2 = 'aaa'
Where col1 = 1

                                   Update test set col2='aaa'
                                   Where col1 = 1
```

In deze situatie zal sessie 2 wachten totdat sessie 1 een commit van de transactie doet of een rollback van de transactie doet. In de v$ view v$session_wait kunnen we zien waar sessie 2 op zit te wachten:

```
Select sid, event, p1raw, p2, p3
From v$session_wait
Where sid = <sessie 2>
```

```
       SID EVENT
----------------------------------------------------------------------------------------
P1RAW             P2          P3
----------------------------------------
          9 enqueue

54580006     131079        1501
```

Hieruit blijkt dat sessie 2 wacht op het wait event enqueue. De waarden in p1raw, p2 en p3 laten zien dat het om een row lock gaat. Diezelfde informatie kan uit v$lock gehaald worden:

```
ADDR     KADDR        SID TY       ID1       ID2      LMODE     REQUEST
----------------------------------------------------------------------------------------
      CTIME       BLOCK
--------------------------
511D3CA8 511D3CBC      9 TM       21425         0         3         0
        954           0

510AAD08 510AAD18      9 TX      131079      1501         0         6
        954           0
```

Hier kunnen we zien dat sessie 2 een TX enqueue in mode 6 (exclusief) probeert te krijgen. We kunnen nog steeds niet zien welk record we proberen te updaten. Dat kun je zien in v$session.

Hier zien we dat sessie 2 object 21425, in file 1, block 44944 en row 0 probeert te locken. Dit alles is voor de meeste data-base administrators niet nieuw. De TX enqueue wordt dus door Oracle gebruikt om Row-level locking te implementeren.

```
SID ROW_WAIT_OBJ#    ROW_WAIT_FILE#    ROW_WAIT_BLOCK#    ROW_WAIT_ROW#

      1        -1             0                 0              0
      2        -1             0                 0              0
      3        -1             0                 0              0
      4        -1             0                 0              0
      5        -1             0                 0              0
      6        -1             0                 0              0
      7        -1             0                 0              0
      8        -1             0                 0              0
      9     21425             1             44994              0
     10        -1             0                 0              0
```

Niet genoeg transactie slots

Elk database block dat een index of een tabel bevat heeft een transactie tabel. De grote van die tabel wordt bepaald door twee opties voor het create table /index statement. Het gaat om INITRANS (initiële grote van de tabel) en MAXTRANS (de maximale grote van die tabel). Laten we nogmaals de test tabel creëren:

```
Create table test (col1 number, col2 varchar(100)) maxtrans 1;
```

Dit betekent dat de transactie tabel in elk block van de TEST tabel maar 1 transactie kan bevatten. Dus als we nu weer 2 sessies elk een aparte row laten updaten in het zelfde block, is er dus een probleem.

```
Sessie 1                        Sessie 2

Update test
Set col2 = 'aaa'
Where col1 = 1

                                Update test
                                Set Col2 = 'bbb'
                                Where col1 = 2;
```

Sessie 2 zal wachten totdat een transactie slot beschikbaar komt. Dat komt pas beschikbaar als sessie 1 een commit of een rollback doet. In v$session_wait blijkt sessie 2 weer te wachten op een enqueue.

```
Select sid, event, p1raw, p2, p3
From v$session_wait
Where sid = <sessie 2>
```

```
       SID EVENT
-----------------------------------------------------------------------------------------------
P1RAW             P2          P3
----------------  ----------  ----------
          9 enqueue
54580004     131102        1502
```

Ditmaal blijkt P1RAW een andere waarde te bevatten en als we kijken in v$lock blijkt dat de request mode 4 is:

```
Select * from v$lock where sid = <sessie 2>;
```

```
ADDR      KADDR      SID TY       ID1       ID2       LMODE      REQUEST
-------------------------------------------------------------------------------------------
      CTIME       BLOCK
----------  ----------
511D3CA8 511D3CBC    9 TM       21425         0         3          0
        954          0

510AAD08 510AAD18    9 TX      131102      1502        0          4
        954          0
```

In v$session staat nu geen informatie meer die bruikbaar is:

```
Select row_wait_obj#, row_wait_file#, row_wait_block#, row_wait_row#
from v$session where sid = <sessie 2>
```

De row_wait_file#, row_wait_block# en row_wait_row# bevatten nog informatie van de vorige keer dat er op een row gewacht moest worden. Alleen row_wait_obj# is op -1 gezet. Dit klopt, want er wordt niet echt op een bepaalde rij gewacht maar op een slot in de transactie tabel.

De oplossing in dit voorbeeld is eenvoudig: verhoog MAXTRANS.

```
Alter table test MAXTRANS 10;
```

Maar dat werkt niet, omdat nu alleen de nieuwe databaseblokken die worden gealloceerd een MAXTRANS met waarde 10 krijgen, niet de al gealloceerde blokken. De oplossing is om de tabel opnieuw te creëren.

De default waarde van MAXTRANS is 255. Dat betekent dat er 255 verschillende transacties kunnen plaats vinden in een datablok op hetzelfde moment. INITRANS heeft verschillende default waarden. Het hangt af van de versie van Oracle en het type object dat je creëert:

	Voor Oracle9i		**Oracle9i**	
	Index	**Tabel**	**Index**	**Tabel**
INITRANS	2	1	2	2
MAXTRANS	255	255	255	255

Niet genoeg transactie slots is alleen een probleem voor select .. for update, update en delete statements. Insert statements pakken een ander blok om de insert te laten doorgaan.

Unieke Index sleutels en Inserts

Wanneer we nu een unieke index creëren op de test tabel en dan inserts van uit verschillende sessies doen met de zelfde unieke sleutel dan zal 1 sessie slagen en de andere zal wachten:

Create unique index itest on test(col1);

```
Sessie 1                              Sessie 2
Insert into test values (4, 'ZZZ');   Insert into test values (4, 'ZZZ');
```

Sessie 2 wacht tot sessie 1 een commit of een rollback doet. In v$session_wait kun je zien dat sessie 2 weer op een enqueue wacht:

```
Select sid, event, p1raw, p2, p3
From v$session_wait
Where sid = <sessie 2>
```

```
       SID EVENT
----------------------------------------------------------------------------------------------------
P1RAW             P2          P3
-----------------------------------------
          9 enqueue
54580004     131112        1501
```

In v$lock kunnen we zien dat sessie 2 weer op een TX enqueue wacht in mode 4. In v$session zien we het volgende:

```
select * from v$lock where sid = 9;
```

```
ADDR     KADDR       SID TY       ID1      ID2       LMODE       REQUEST
----------------------------------------------------------------------------------------------------
      CTIME       BLOCK
---------------------------
093072B0 09307380      9 TX      196618     1457        6           0
        147           0

511D3D1C 511D3D30      9 TM       21425      0          3           0
        147           0

510AAD08 510AAD18      9 TX      131112     1501         0           4
        147           0
```

De v$session view laat het volgende zien:

```
select sid,  ROW_WAIT_OBJ# ,  ROW_WAIT_FILE# , ROW_WAIT_BLOCK#  ,
ROW_WAIT_ROW#   from v$session;
```

Hieruit blijkt dat er niet een bepaalde row gelocked is.
Dat klopt ook, want we wachten op de transactie van sessie 1.
Als die klaar is kan sessie 2 pas kijken op de rij al bestaat.
Hetzelfde kan gebeuren bij een update statement.
Bijvoorbeeld een bestaande unieke sleutel wordt door twee sessies veranderd naar dezelfde unieke sleutel.

```
SID ROW_WAIT_OBJ#    ROW_WAIT_FILE#    ROW_WAIT_BLOCK#     ROW_WAIT_ROW#
----------------------------------------------------------------------------------------------------
   1        -1             0                  0                0
   2        -1             0                  0                0
   3        -1             0                  0                0
   4        -1             0                  0                0
   5        -1             0                  0                0
   6        -1             0                  0                0
   7        -1             0                  0                0
   8        -1             0                  0                0
   9        -1             1              44994                0
  10        -1             0                  0                0
  11        -1             0                   0               0
```

DML op Bitmap Indexes

Bitmap indexes slaan hun gegevens op in zogenaamde bitmap segmenten. Deze segmenten worden compleet ge-locked door een transactie. Dus ook als de transactie maar 1 rij veranderd zijn alle andere rijen in dat segment ge-locked. Een tweede transactie van een andere sessie op een rij in dat segment, moet wachten tot dat de eerste sessie een commit of rollback doet. De tweede sessie zal wachten op een TX lock in mode 4.

Niet genoeg Transaction Free List

Elke tabel of index wordt in een zogenaamd segment gecreëerd. Elk eerste block in een segment bevat informatie over het segment, zoals informatie over het High Water Mark, de extent map, freelists informatie et cetera. In een segment heb je process freelists (minimaal 1) en transaction freelists (minimaal 16).

Primary/Foreign key Constraints

Er zijn situaties denkbaar waarin je bijvoorbeeld een delete van een parent in één sessie doet en een insert in een child vanuit een andere sessie. De insert zal dan moeten wachten totdat de delete klaar is. Pas dan kan de insert beslissen of de parent nog bestaat. De insert zal dan wachten op een TX lock in mode 4.

Conclusie

Row locking problemen komen veel en vaak voor in applicaties. Ze zijn erg makkelijk te traceren. DBA en ontwikkelaar kunnen dan samen uitvechten hoe het probleem op te lossen. Het grote probleem is dat de Oracle kernel de Row Lock ook voor andere zaken gebruikt wordt (zoals niet genoeg transactie slots in een blok, etc.). Die zaken zijn vaak wat moeilijker te achterhalen en zullen waarschijnlijk tot een hoop frustratie tussen de DBA en ontwikkelaar leiden. Let dus goed op: een TX enqueue in v$lock met REQUEST 6, is een zuivere row-lock, een TX enqueue met REQUEST 4 wordt gebruikt voor andere interne Oracle kernel wacht situaties.

Anjo Kolk
is chief Oracle technologist bij Precise Software Solution.
Hij is per e-mail te bereiken op akolk@precise.com.