I

Buffer Lock과 Buffer Busy Waits

Oracle은 정교한 Row level lock 메커니즘을 제공하기 때문에 Block 단위로는 블로킹이 발생하지 않는것으로 알려져 있다. 하지만 실제로는 Block 단위의 경합에 의한 성능 문제가 많이 발생한다. 우리는 본 Article 에서 Block을 보호하기 위한 Buffer Lock 의 개념과 그로 인한 성능문제에 대해 논의할 것이다.

글 (주)엑셈 책임컨설턴트 조 동 욱(ukja@ex-em.com)

Oracle 은 매우 정교하고 효과적인 Row level locking 메커니즘을 제공하며, 사용자의 동시변경으로부터 데이터를 보호하기 위해 Block 단위로 Lock 을 거는 일은 없는 것으로 알려져 있다. 하지만 이것은 절반만의 사실이다. 공식적으로는 Block 단위의 Lock 이 존재하지 않지만, Oracle 의 논리적인 IO 가 Block 단위로 이루어지기 때문에 내부적으로 Block 단위의 Lock 은 반드시 필요하다. 가령 Row1, Row2 두 개의 Row 가 같은 Block 안에 있다고 가정하자. 두 명의 사용자 User1, User2 가 각각 Row1, Row2 를 Update 한다면 논리적으로는 두 개의 Update 행위 사이에는 서로 보호해야 할 데이터가 존재하지 않는다. Oracle 은 Row level 을 Lock 을 제공하기 때문에 서로 다른 Row 를 변경하는 것은 전혀 문제가 되지 않기 때문이다. 하지만 두 개의 Row 가 같은 Block 안에 있다는 물리적인 제한으로 인해 Block 을 변경하는 행위 자체는 동시에 이루어져서는 안 된다. 각 사용자는 Row 를 변경하기 위한 TX Lock 을 Exclusive 하게 획득했다하더라도 현재 단 한 명의 사용자만이 Block 를 변경하고 있다는 것을 보장받아야 한다. 이 경우에 획득해야 하는 Lock 을 Buffer Lock 이라고 부른다. 만일 Buffer Lock 을 획득하지 못하면 다른 Lock 들과 마찬가지로 Lock 을 획득할 때까지 대기해야한다.

Buffer Lock 을 획득하는 모드에는 Shared 모드와 Exclusive 모드가 있다. Buffer 를 읽는 과정에서는 Shared 모드의 Lock 을 획득해야 하고, 변경하는 과정에서는 Exclusive 모드의 Lock 을 획득해야 한다. Buffer Lock 을 획득하려는 세션들 간의 모드가 호환성이 없을 때 (가령 서로 Exclusive 하게 획득하려고 하거나 Shared 모드로 읽고 있는 중에 Exclusive 모드로 획득하는 경우) 경합이 발생하게 된다. Buffer Lock 은 cache buffers chains latch, TX Lock 과 함께 Buffer 의 변경을 동기화하는 역할을 한다. 추상적인 레벨에서 하나의 Row 를 변경하기 위해서 latch 나 lock 을 획득하는 과정은 다음과 같다.

1. 변경하고자 하는 Row 에 해당하는 Block 이 존재하는 위치에 찾아가기 위해 cache buffer chains latch 를 획득한다.
2. Block 을 찾은 해당 Buffer 에 대해 Buffer Lock 을 획득하고, cache buffers chains latch 를 해제한다.
3. 해당 Row 에 대해 TX Lock 을 획득하고 Row 를 변경한다.
4. Buffer Lock 을 해제한다.

불행히도 Buffer Lock 이라는 용어는 Oracle 의 공식용어가 아니며, 일반적으로 통용되는 용어도 아니다. Steve Adams 나 Jonathan Lewis 같은 Oracle 전문가들이 Buffer Lock 이라는 용어를 사용하고 있다. Buffer 가 변경 중이라는 사실을 보호하는 것이므로 Buffer Pin 이라는 용어가 의미적으로는더 적합하다고 볼 수도 있지만, 큰 차이가 없으므로 본 Article 에서는 Buffer Lock 이라는 용어를 따르기로 한다.

Buffer Lock 을 획득하기 위해 대기하는 것을 일반적으로 *buffer busy waits* 대기라고 부른다. *buffer busy waits* 는 가장 일반적으로 발생하는 대기현상 중 하나이며 그 원인 또한 다양하다. 또한 다른 System type lock 들과 마찬가지로 문제를 해결하는 것 또한 까다롭다.

<참조> buffer lock 을 획득하기 위한 이벤트는 총 3개(10g 에서는 네개)이다. *buffer busy waits*, *buffer busy due to global cache (RAC)*, *write complete waits*, *read by other session* (10g 부터) 이 그것이다. 본 Article 에서는 *buffer busy waits* 를 기준으로 설명을 하되 필요한 경우 각각의 대기이벤트에 대해 언급할 것이다.

buffer busy waits 대기이벤트의 정의는 Oracle 의 버전별로 조금씩 다르며 10g 이후부터는 대기이벤트 자체가 두 개로 분화되었다.

Metalink NoteID<34405.1>에서 자세한 정보를 얻을 수 있다.

Oracle 10g 이전 버전

아래내용은 메타링크에서 발췌한 것이다.

buffer busy waits 대기이벤트의대기 파라미터는 다음과 같다.

P1: 절대(absolute) File#

P2: Block#

P3: 원인(reason) 코드. 오라클 10g 이전까지는 대기의 원인을 나타낸다. 오라클 은 다양한 원인코드(reason code)를 이용하여 커널 코드안의 여러부분에서 발생되는다양한 대기 원인을 나타내주었다. 원인코드의 값은 오라클 버전에 따라 다르며, 오라클 8i 이전부터 9i 까지변경되어왔다. 하지만, 오라클 10g에서는 원인코드를 더 이상 사용하지않는다. 오라클 10g 에서 P3는 V$WAITCLASS 뷰의 클래스를 나타낸다. 6 장에서는 이러한 정보들을 해석하는 방법을 자세히 설명할 것이다.

아래의 표는 오라클 10g 이전에 사용된 원인코드와 그에 대한설명을 기술한다. 괄호 안의원인코드는 오라클 8.1.5 이하에서사용된 원인코드다.

원인 코드(Reason Code)	설 명
100(1003)	블로킹(blocking) 세션은 버퍼 캐시로 블록을 적재하는 중이며, 롤백을 위한 Undo블록일 가능성이 높다. 해당 정보를 이용하여 새로운 버전의 블록을 생성하기 위해, 배타적인(exclusive) 액세스를 하려는 세션은 대기해야 한다.
110(1014)	대기 세션은 블로킹 세션이 버퍼 캐시로 적재하고 있는 블록에 대한 현재(current) 이미지를 읽거나, 기록하려고 한다.
120(1014)	대기 세션은 블로킹 세션이 버퍼 캐시로 적재하고 있는 블록을 현재(current) 모드로 액세스하려고 한다. 버퍼 룩업(buffer lookup)시에 발생한다.
130(1013)	하나 이상의 세션이 버퍼 캐시에 존재하지 않는 블록을 액세스하려고 할 경우, 하나의 세션이 db file sequential read 또는 db file scattered read 이벤트를 발생시키면서 I/O 작업을 수행하는 동안, 다른 세션들은 해당 원인코드를 가지고 buffer busy waits 이벤트를 발생시킨다.
200(1007)	블로킹 세션이 버퍼 캐시 안의 블록을 변경하는 동안 새로운 버전의 블록을 생성하기 위해, 해당 블록에 배타적인(exclusive) 액세스를 해야 하는 세션은 대기해야 한다.
210(1016)	블로킹 세션이 블록을 변경 중일 때 배타적인(exclusive) 모드로 블록의 현재(current) 버전을 원하는 세션은 대기해야 한다. 두 개의 세션이 동일한 블록을 변경하려고 할 때 발생한다.
220(1016)	블로킹 세션이 블록을 변경 중일 때 현재(current) 모드로 블록을 액세스하려는 세션은 대기해야 한다.
230(1010)	블로킹 세션이 블록을 변경 중일 때 해당 블록을 공유(shared) 모드로 액세스하려는 세션은 대기해야 한다.
231(1012)	블로킹 세션이 블록을 변경 중일 때, 해당 블록의 현재(current) 버전을 읽고 있는 세션이, 해당 블록에 대한 공유(shared) 액세스를 하려고 할 경우에 대기 해야 한다.

Oracle10g 이후:

Oracle10g부터는 buffer busy waits 대기이벤트의정의가 크게변경되었다.

첫째, 기존의buffer busy waits 가read by other session 과buffer busy waits 두 개의이벤트로 분화되었다. read by other session 은 Reason code 130에 해당하고buffer busy wait 은 Reason code 는 220에 해당한다.

둘째, Oracle10g 이후부터는 P3 의 용도가 변경되었다. Reason code는 더 이상 제공되지 않으며 P3의 값은Class# 을 나타낸다. Class#은 Block 의 종류를 나타내는 것으로 Class#은 다음 쿼리를 통해서 확인가능하다. 비록Reason code 는 없어졌지만 대기이벤트가 분화되었고, Class#이 제공되므로 분석에는 전혀지장이 없다. 오히려 명확한 Block 의 Class 를 알수 있기때문에 원인분석에 오히려 유리해졌다고 보는 것이 맞을것이다.

Oracle 이 사용하는 기본적인Block Class 는 다음과 같다.

```
SQL> select rownum as class#, class from v$waitstat;
        1 data block
        2 sort block
        3 save undo block
        4 segment header
        5 save undo header
        6 free list
        7 extent map
        8 1st level bmb
        9 2nd level bmb
       10 3rd level bmb
       11 bitmap block
       12 bitmap index block
       13 file header block
       14 unused
       15 system undo header
       16 system undo block
       17 undo header
       18 undo block
```

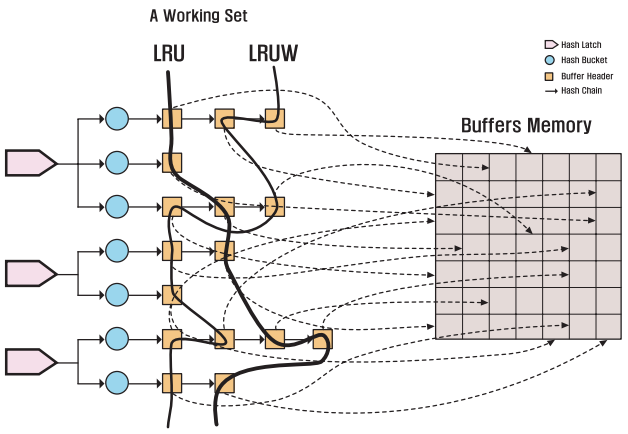
undo header, undo block 인 경우실제로 Buffer Header에 지정되는Class 값은 위의 값과는 다른데, 자세한 내용은 잠시 후에 설명하기로 한다.

*Buffer busy waits*에 대해서 본격적으로 논의하기 전에Oracle에서 Buffer Cache의 작동 메커니즘에 대해 기본적인사항을 짚고 넘어가기로 하자.

Buffer

Oracle 의Buffer Cache 는 다음과 같은구조로 이루어져있다.

Hash table -> Hash bucket -> buffer header chain -> buffer header -> buffer body -> block header -> block body



buffer header와 buffer body 는 1:1 의 구조로 이루어져 있다. 하나의 buffer 에 대해 buffer lock 을 획득한다는 것은 buffer header에 대해 lock 을 획득한다는 의미이다. buffer header 에는 buffer lock을 소유한 프로세스목록과 buffer lock 을 대기하고 있는(즉 *buffer busy waits* 대기이벤트를 겪고 있는) 프로세스목록 정보가 있으며 이를 이용해 lock의 획득과 해제가 순차적으로 이루어지게 된다.

buffer cache와 buffer header 의 구조를 알수 있는 가장좋은 방법은 buffer cache 를 파일로 dump하는것이다. buffer cache 를 dump하는 방법은 여러가지가 있지만 여기서는 oradebug 를 사용하기로 한다.

```
SQL> oradebug setmypid
SQL> oradebug dump buffers 10
(참조 1 = header only, 10 = header and body)
SQL> oradebug tracefile_name
/home/oracle/admin/ORA102/udump/ora102_ora_19249.trc
```

dump 파일의 내용중에 Buffer Header 에 해당하는 부분은 다음과 같다.

```
BH (c000000010ff03b0) file#: 4 rdba: 0x01000d5d (4/3421) class: 1 ba: c000000010e5a000
  set: 6 blksize: 8192 bsi: 0 set-flg: 0 pwbcnt: 0
  dbwrid: 0 obj: 53733 objn: 53733 tsn: 4 afn: 4
 hash: [c000000017acecc8,c000000017acecc8] lru: [c0000000107f9868,c0000000113eb998]
  lru-flags:
  ckptq: [NULL] fileq: [NULL] objq: [c0000000113eba00,c000000013fe3420]
  use: [c000000017b41b18,c000000017b41b18] wait: [NULL]
  st: CR md: SHR tch: 0
  cr: [scn: 0x1.9f98e79e],[xid: 0x0.0.0],[uba: 0x0.0.0],[cls: 0x1.9fb3ed19],[sfl: 0x0]
  flags: only_sequential_access
```

위의항목 중use, wait 항목이 Buffer lock 을 획득한(즉 현재buffer 에 대해 pin 을 수행하는) 프로세스와 대기하고 있는 프로세스를 나타낸다. (해당 값은 프로세스 주소로 추정됨) lock 을 획득한 세션이 여러 개인 것은 Buffer lock 을 획득하는 모드가 Shared 모드와 Exclusive 모드가 가능하기 때문이다.

md (Lock Mode) 값은 SHR 로, Shared 모드로 현재 Buffer Lock 을 획득된 상태이며, Buffer Lock 을 획득한 프로세스는 c000000017b41b18 하나이다. 또한 class = 1 은 data block 임을 가리킨다.

st (Status) 는 Buffer 의 상태를 가리킨다. 일반적으로 다음과 같은 값을 지닌다. (이 값에 대한 정의는 v$bh 와 x$bh 뷰를 참조한다)

```
0     FREE      no valid block image
1     XCUR      a current mode block, exclusive to this instance
2     SCUR      a current mode block, shared with other instances
3     CR        a consistent read (stale) block image
4     READ      buffer is reserved for a block being read from disk
5     MREC      a block in media recovery mode
6     IREC      a block in instance (crash) recovery mode
8     PI        past image (RAC 에서만 쓰임)
```

Single Instance 인 경우에는 대부분 XCUR (RENT) 또는 CR 상태이며, RAC와 같은 Multi instance 환경에서는 SCUR (RENT) 와 PI 상태를 지니기도 한다.

<참조> 위의 상태는 Buffer 의 상태를 가리킨다. Steve Adams 는 자신의 책에서 Buffer 의 상태와 Block 의 상태를 구분해서 정의하고 있다.
Steve Adams 의 분류법을 따르면 Block 의 상태는 Clean Current Block, Dirty Current Block, Stale Block 으로 나뉜다. Stale block 은 Consistent Read 를 위해 SGA에 보관되어 있는 즉, 과거 상태의 값을 가지고 있는 block 을 말한다. Current block 은 그 반대로 현재 값을 가지고 있는 block 이다.
Clean current block 은 current block 중에 이미 Disk에 쓰기가 이루어져 있는 상태를 말하며, Dirty current block 은 값은 바뀌었지만 아직 Disk에 쓰이지 않는 상태를 말한다.
CKPT 와 DBWR 프로세스에 의해 Dirty current block 은 Disk에 쓰기가 되고 그 상태가 Clean current block 으로 바뀌게 된다. 위의 Buffer 상태와 비교해 보면 Stale Block 은 CR Buffer 에 해당하며, Current Block 은 XCUR-RENT 또는 SCURRENT Buffer 에 해당한다.

Block

Buffer Header 가 가리키고 있는 Buffer body 는 Block 정보를 담고 있다. Block 의 종류는 아래와 같이 매우 다양하다.

```
SQL> select rownum as class#, class from v$waitstat;
         1 data block
         2 sort block
         3 save undo block
         4 segment header
         5 save undo header
         6 free list
         7 extent map
         8 1st level bmb
         9 2nd level bmb
        10 3rd level bmb
        11 bitmap block
        12 bitmap index block
        13 file header block
        14 unused
        15 system undo header
        16 system undo block
        17 undo header
        18 undo block
```

위와 같이 보통 18 가지 정도 (Version 에 따라 다름) Block 이 존재하며 종류에 따라 다른 내용을 담고 있다.

Undo 에 해당하는 block 의 클래스 값 (15 이상) 을 해석할 때 주의할 점이 있다. 실제로 buffer busy waits 대기가 발생할 때 p3 (class) 를 추적해 보면 18 보다 큰 값이 나타나는 경우가 많다. 이러한 값들은 undo header block 또는 undo data block (또는 rollback segment) 를 가리키는 것이며 다음 공식에 의해 계산된다. (9i 이상)

```
Undo Header Block = 15 + 2*r (8i 의 경우는 11 + 2*r)
Undo Data Block = 16 + 2*r *(8i 의 경우는 12 + 2*r)
```

여기서 r 은 Rollback(Undo) segment 번호를 말하는데 dba_rollback_segs.segment_id 값으로 확인할 수 있다. 위의 공식을 간단하게해석하면만일p3 (class) 가15이상이면서 홀수이면Undo head block 에 해당하며, 짝수이면 Undo data block 에 해당한다.
15번에 해당하는system undo header 는segment_id 가0 이므로 15 번이된다.

Buffer Lock 경합이 어떤종류의Block 에 대해 발생하느냐에 따라그 문제의 원인 및 해결책이 모두 다르므로 buffer busy waits 대기가 발생하는 Block 의 종류 (Class)를 정확하게 판별하는 것이매우중요하다.

Block 의구조를가장 직관적으로 알수있는방법은block dump 를수행하는것이다. Block dump는 앞서 설명한buffer cache dump를 통해서도가능하며 "alter system dump datafile #, block# " 명령을 통해서도가능하다.

가장 일반적인block 인 data block 에 대한dump 결과는 다음과 같다.

Block Header

DBA (rdba), Segment/Object ID(seg/obj), SCN(csc), Type(typ, 1=Table, 2=Index), ITL(itl)등의 정보가Header 에보인다.

```
Block header dump:  0x0040197c
 Object id on Block? Y
 seg/obj: 0x2  csc: 0x00.e903  itc: 2  flg: -  typ: 1 - DATA
     fsl: 0  fnx: 0x0 ver: 0x01
 Itl         Xid                  Uba         Flag  Lck     Scn/Fsc
0x01   0x0007.02b.0000001d  0x00800c6c.0015.4a  C---    0  scn 0x0000.0000e8f4
0x02   0x0004.014.00000120  0x00800222.00c8.02  --U-    3  fsc 0x0000.00088f91
```

Block Body

Data area size (tsiz), Data header size (hsiz), Row count (nrow) 등의 정보와 Row dump (block row dump) 정보를 확인할수 있다.

```
data_block_dump,data header at 0xc00000001093005c
===============
tsiz: 0x1fa0
hsiz: 0x8e
pbl: 0xc00000001093005c
bdba: 0x0040197c
     76543210
flag=--------
ntab=12
nrow=40
frre=21
fsbo=0x8e
fseo=0x1a7d
avsp=0x19ef
tosp=0x19ef
0xe:pti[0]          nrow=10    offs=0
0x12:pti[1]         nrow=0     offs=10
...
block_row_dump:
tab 0, row 0, @0x1f89
tl: 23 fb: K-H-FL-- lb: 0x0  cc: 1
curc: 8 comc: 8 pk: 0x00401979.2 nk: 0x00401979.2
col  0: [ 3]  c2 25 4f
tab 0, row 1, @0x1f72
tl: 23 fb: K-H-FL-- lb: 0x0  cc: 1
curc: 1 comc: 1 pk: 0x0040197c.1 nk: 0x0040197c.1
col  0: [ 3]  c2 26 33
...
end_of_block_dump
```

직접dump 를수행한후class# 에 따라block header 와block body 가어떤구조를지나는지직접살펴보기 바란다.

필자는buffer busy waits 대기이벤트를분석함에있어서 기존의 자료와는 약간의 다른 접근방법을 따른다. 대부분의책이나 자료에서는Reason code나Class# 에서 시작하여buffer busy waits를 설명하고있으나 이 방법은 직관적인지 못한 면이 있다. 따라서 필자는 Select, Insert, Update, Delete 의 실제 SQL 문 수행 시 Buffer Lock 이 어떻게 획득되고buffer busy waits가어떻게 발생하는지를분석하고 이를통해서 buffer busy waits대기를 줄이는 방법을 논의할 것이다. 또한테스트 환경을 간결하게 하기위해 Table Segment 에 대해서만 테스트를

수행하되, 필요한 경우 Index Segment 에 대해서도언급할 것이다.

특별한 언급이 없다면 모든 테스트의 테스트환경은 10g R2 버전에서 수행한다.

A. Select / Select 와 Buffer Busy Waits

Select / Select 에 의한 Buffer Lock 경합은 동시에 같은 Block 을 메모리에 올리는 과정에서 발생하는 것이므로 *read by other session* 대기를 유발할 것으로 예상할 수 있다. 아래 테스트스크립트와 결과를 보자.

[테스트스크립트]

```
create tablespace bfw_tbs datafile size 50M autoextend on
extent management local uniform size 1M
segment space management auto;
create table bfw_test(id char(1000)) tablespace bfw_tbs;
insert into bfw_test select ' ' from all_objects where rownum <= 50000;
create or replace procedure bfw_do_select
is
begin
        for x in (select * from bfw_test) loop
                null;
        end loop;
end;
/

connect maxgauge/maxgauge@ora10gr2
select sid from v$mystat where rownum = 1;
var job_no number;
begin
        for idx in 1 .. 20 loop
                dbms_job.submit(:job_no, 'bfw_do_select;');
        end loop;
        commit;
end;
/

exec bfw_do_select;
@event;
```

[테스트결과]

```
EVENT                          TOTAL_WAITS   TIME_WAITED
------------------------------ -----------   -----------
db file sequential read                266           212
read by other session                  164           205
db file scattered read                 262           195
SQL*Net message from client             20             5
latch: cache buffers lru chain           1             4
log file sync                            4             0
SQL*Net message to client               21             0
```

20 여 개의 세션이 동시에 동일한 테이블을 읽어 들일 때 예상한 바대로 *read by other session* 대기이벤트가 광범위하게 나타난다. v$session_wait 뷰를 이용해 *read by other session* 대기이벤트를 캡처한 결과는 다음과 같다.

```
sid=149, event = read by other session, p1 = 10, p2 = 3820, p3 = 1
```

p1 (file#) = 1 에 해당하는 data file 은 해당 테이블이 존재하는 테이블스페이스의 데이터 파일이며

```
SQ> select name from v$datafile where file# = 10;
C:\ORACLE\PRODUCT\10.1.0\ORADATA\UKJADB\UKJADB\DATAFILE\O1_MF_BFW_TBS_1W622R9R
_.DBF
```

p3 (class#) = 1 이므로 data block (여기서는 table) 임을알 수 있다.

하지만, 위의 테스트 결과를 해석할 때는 상당한 주의를 요한다. 동일한 방식으로 다시 한번 테스트를 수행하면 *read by other session* 대기및 *db file sequential read, db file scattered read* 대기이벤트가 대부분 사라진다.

db file sequential read, db file scattered read 대기가 사라진 것은 모든 Block 들이 SGA 에 로드 되었기 때문에 물리적 IO 가 사라진 것과 관련이 있다. *read by other session* 대기이벤트가 사라진 것 또한 같은 이유다. 읽고자 하는 Block 들이 이미 SGA 에 로드되어 있는 경우에는 Buffer Lock 경합이 발생하지 않는다. 즉, *read by other session* 대기이벤트는 *db file sequential read, db file scattered read* 대기이벤트와 밀접한 관련이 있다.

<참조> 읽기 작업인 경우에도 Buffer Lock 을 Exclusive 하게 획득한다는 것에 유의해야 한다. 이것은 하드파싱이 발생하는 경우에 해당 SQL Cursor 에 대해 library cache pin 을 Exclusive 하게 잡는 것과 유사한 개념이다. 이 문제에 대해서는 library cache lock 과 library cache pin 에 대한 Article 에서 자세히 논의한다.

앞서 Buffer Lock 은 Shared 모드와 Exclusive 모드만이 존재한다고 했다. 이미 SGA 에 적재되어있는 Block 을 읽을 때는 Shared 모드로 Buffer Lock 을 획득하기 때문에 Buffer Lock 에 의한 경합이 발생하지 않는다. 하지만 물리적 IO 가 발생하여 Block 을 새롭게 SGA 에 올리는 것은 Buffer 를 새로 생성/변경하는 작업을 요구하므로 최초로 Buffer 를 생성하는 세션은 Buffer Lock 을 Exclusive 하게 획득하게 된다. 따라서 해당 Block 을 읽기 위해 Buffer Lock 을 Shared 모드로 획득하려는 다른 세션들은 Exclusive Buffer Lock 이 해제될 때까지 기다려야 한다. 이로 인해서 *read by other session* 대기가 발생하게 되는 것이다.

위의 테스트결과는 가능한 물리적 IO 를 줄이고 논리적 IO 를 늘리는 것이 얼마나 중요한지에 대한 또 다른 예가 된다. 더불어 논리적 IO 자체를 최적화해야 하는 당위성을 제공한다.

위의 테스트에서는SGA의 크기가 읽을 대상이 되는Block들을 모두메모리에 상주시킬 정도로 크기 때문에 두 번째테스트에서는 IO 및Buffer 관련 대기가 거의없어졌지만 만일SGA가 너무작거나 너무나 많은Block을 읽어버리는 바람에 다른세션들에의해 해당Block들이SGA에서 내려가버리면 다시물리적 IO가 발생하게되고 *read by other session*대기가 계속해서나타나게 된다.

따라서 Select / Select 에 의한 *read by other session* 대기를 줄이는 방법은 다음과같이 정리할수있다.

- SQL 최적화를통해서 최소한의 논리적 IO 만으로 원하는 결과를 가져올수 있도록 해야한다.
- SGA 사이즈(또는db cache size) 가 시스템 전반적인 IO에 비해작다면 SQL 튜닝만으로는 문제를 해결할수 없으며 SGA 의 물리적 크기를 늘려주어야 한다.

B. Select / Update 에 의한 Buffer Busy Waits

Select / Update 에 의한 Buffer Lock 경합은 Select / Select 나 Update / Update 에 의한Buffer Lock 경합과는 그 메커니즘이 상당히 다르다.
Oracle의 Select는 기본적으로 Consistent Read 에 기반하므로 실제 읽어야 할 데이터block이 변경되었다면 해당Block의 과거이미지를 가지고 있는 CR Block 또는 Undo Block을 읽어야 한다. 만일 여러 세션이 동시에 Undo Block에 대해 읽기를 시도할경우 Undo block을 메모리에 올리는 과정에서 Buffer Lock경합이 발생하게 된다. 따라서 Select / Update 에 의한 Buffer Lock 경합은 다음과 같은 상황에서발생할 것으로 예측할수있다.

- 특정 세션이 특정 테이블을변경(Update) 한다.
- 이때 다른세션(들)이 변경 중인Block 에 대해 읽기를 시도한다.

아래테스트를 통해 이를확인해보자.

[테스트스크립트]

-- Select를수행하는 프로시저

```
create or replace procedure bfw_do_select
is
begin
    for x in (select t1.id as id1, t2.id as id2 from bfw_test t1, bfw_test t2 where rownum
<= 500000)
loop
            null;
    end loop;
end;
/
```

-- 동일 테이블에 대해Update 를수행하는프로시저

```
create or replace procedure bfw_do_update
is
begin
          update bfw_test set id = ' ';
end;
/
connect system/oracle
select sid from v$mystat where rownum = 1;
var job_no number;
```

-- 업데이트를 수행하는동안여러 세션이 동시에Select를수행한다.

```
begin
        dbms_job.submit(:job_no, 'bfw_do_update;');
        commit;
        for idx in 1 .. 10 loop
          dbms_job.submit(:job_no, 'bfw_do_select;');
        end loop;
        commit;
end;
/
exec dbms_lock.sleep(1);
exec bfw_do_select;
@event;
```

[테스트결과]

위의 테스트스크립트가수행되는 도중에 v$session_wait 뷰를 캡처하면 다음과 같다. p3(class#)에 유의하자.

```
sid=139, event = read by other session, p1 = 10, p2 = 1980, p3 = 1
sid=140, event = read by other session, p1 = 10, p2 = 1980, p3 = 1
sid=139, event = buffer busy waits, p1 = 7, p2 = 169, p3 = 73
sid=154, event = buffer busy waits, p1 = 7, p2 = 57, p3 = 59
sid=123, event = read by other session, p1 = 7, p2 = 36549, p3 = 60
sid=123, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=130, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=137, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=138, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=139, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=140, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=143, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=149, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=152, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
sid=154, event = read by other session, p1 = 7, p2 = 58139, p3 = 60
...
```

p3가 73,59 등의 값을 지니는 *buffer busy waits* 대기가 일부 보이며p3(class)가 60의 값을 지니는 *read by other session* 대기가 광범위하게나타난다.

우선, 그 정도는 미약하지만 특수성으로 인해 충분한 분석가치가 있는 *buffer busy waits* 의 경우를 먼저 살펴보자.

*buffer busy waits*의 경우p1 (file#) = 7 이고p3 = 59, 73 으로나타난다.

일단file# = 7 에 해당하는 파일명과 테이블스페이스 명은 다음과 같이 Undo 영역임을 알수있다.

```
SQL> select f.name, t.name from v$datafile f, v$tablespace t where f.file# = 7 and t.ts# = f.ts#
C:\ORACLE\PRODUCT\10.1.0\ORADATA\UKJADB\UKJADB\DATAFILE\O1_MF_UNDOTBS2_1Q818KMK_.DBF
UNDOTBS2

SQL>show parameter undo_tablespace
UNDOTBS2
```

문제는 class 73, 59 에 해당하는 Block 의 종류를 얻는 것이다. 앞서 설명한 Block 종류에 대한 내용을 기억한다면 73, 59 는 15 이상의 홀수이므로 Undo head block 임을 알수있다. 실제로 block 의 종류를 얻기 위해buffer cache 에 대한 dump 파일을 이용하면다음과 같은 값을 확인할수 있다.

```
class = 73 : KTU SMU HEADER BLOCK
class = 59 : KTU SMU HEADER BLOCK
```

KTU는 언두영역의 내부관리(Internal management of undo and rollback segments) 와 관련된 오라클 커널(Kernel) 영역을 가리키는 말이며 SMU 는 System Management Undo 의 약자로 AUM (Automatic Undo Management) 을 사용하는 언두영역임을 말한다. 따라서 73, 59 클래스의 블록들은AUM 을 사용하는 언두 세그먼트 헤더 블록(Undo Segment Header Block)이다. 이 경우언두 헤더 블록에서의 buffer lock 경합은 Update 세션의 언두 헤더 블록변경 작업과 Select 세션에 의한 언두 헤더블록 읽기 작업간의buffer lock 경합에 의해 발생한다. 헤더 블록을 변경하는 프로세스는 buffer lock 을 Exclusive 하게 획득해야하고, 헤더 블록을 읽는 프로세스는 buffer lock 을 Shared 모드로 획득해야하기때문이다.

Undo Segment Header에서의 buffer busy waits 는 일반적으로 Manual Undo Segment (즉 Rollback segment) 를 쓰는환경에서 Rollback Segment 의 개수가 너무 작거나 Extent 크기가 작아서 Header의 정보가 자주변경되는 경우에 발생한다고 알려져 있다. 하지만, 위의 테스트 결과에서 알수있듯이 SMU (AUM)을 쓰는환경에서도 Segment Header Block 에 대한경합은 분명히 발생한다.

가장성능에 많은 문제를 일으키는 class = 60 에 해당하는 read by other session 대기이벤트를 분석해보자. class = 60 에 해당하는 Block 은 15 이상의 짝수이므로 Undo data block 에 해당하며 buffer cache dump 파일에서다음과 같이확인할수 있다.

```
class = 60 : KTU UNDO BLOCK
```

Undo block 에 대한 경합은 Oracle 의 가장 기본적인 메커니즘 중 하나인 Consistent Read 에 그 원인이 있다. Select 세션들이 Data block 을 읽을 때, Update 에 의해 변경된 상태인 경우에는 기본적으로 Undo Block 을 이용해서 과거의 상태를 읽어와야 한다. 이 때, Undo Block 을 동시에 많은 세션들이 읽기를 시도하면서 Undo block 에 대해서 1 번 테스트에서와 동일한 Select / Select 에 의한 Buffer Lock 경합이 발생하고 이로인해 *read by other session* 대기이벤트가발생하게 된다. 일반적으로 Select 와 Update 는 서로간에 경합을 일으키지 않는다고알려져있지만 위의 테스트결과와 같이 좀더 하위 레벨에서는 buffer lock 경합이 발생하는 것을확인할수 있다.

Select / Update 에서 언두 블록을 읽는과정에서 발생하는 read by other session 대기의 해결책은 데이터 블록을 동시에 읽는 과정에서 발생하는 read by other session 대기의 경우와 동일하다.

SQL 문을적절히 튜닝해서 불필요하게 많은 언두 블록을 읽지 않도록 한다. SGA 크기가 지나치게 작으면 버퍼 캐시에 CR 블록이 상주하지 못해 물리적 읽기 작업이 발생하고 이로 인해 read by other session 대기가 증가할수 있다. 따라서 SGA 의 크기를 적절이 유지해주어야 한다.

C. Insert / Insert 에 의한 Buffer Busy Waits

여러개의 세션이 동시에 같은 테이블에 동시에 Insert 를 수행하는 경우, 매우 복잡한 성능문제가 발생하며 이로 인해 다양한 대기이벤트가 관찰된다. 대량의 Insert 가 발생하는 경우 Segment 영역이 급속히 확장되기 때문이다. 일반적으로 관찰되는 대기이벤트는 다음과 같다.

- *enq: HW - contention*
- *enq: US - contention*
- *enq: ST - contention*
- *enq: TX - row lock contention, enq:TX - allocate ITL entry, enq:TX - index contention*
- ***buffer busy waits***

Insert / Insert 에 의한 Buffer Lock 의 경합은 대부분 잘못된 Freelist 값의 설정에 기인한다고 알려져 있다. FLM (Free List Management) 를 사용하면서 Freelist 값을 1 로 주는 경우 Buffer busy waits 대기가 어떻게 나타나는지 테스트해보자.

[테스트스크립트]

-- Manual Segment Space Management 를 사용하는 경우

```
create tablespace bfw_tbs datafile '/home/oracle/oradata/10gr2/ORA102/ukja_test_01.dbf' size 50M autoextend on
extent management local uniform size 1M
segment space management manual;
```

--Freelist 에 의한 효과를 설명하기 위해freelists = 1 로 매우작은 값을 줌(기본 설정값이 1)

```
create table bfw_test(id char(1000))
storage(freelists 1)
tablespace bfw_tbs;
create or replace procedure bfw_do_insert
is
begin
          for idx in 1 .. 10000 loop
                    insert into bfw_test values(' ');
          end loop;
          commit;
end;
/
connect maxgauge/maxgauge@ora10gr2

select sid from v$mystat where rownum = 1;
var job_no number;
```

--동시에 insert 를 수행

```
begin
       for idx in 1 .. 10 loop
          dbms_job.submit(:job_no, 'bfw_do_insert;');
          commit;
       end loop;
end;
/
exec bfw_do_insert;
@event;
```

[테스트결과]

아래와같이 HW enqueue 와buffer busy waits 에의한 대기가광범위하게 나타난다.

EVENT	TOTAL_WAITS	TIME_WAITED
buffer busy waits	2386	1585
enq: HW - contention	243	1103
events in waitclass Other	23	171
log buffer space	11	130
latch: cache buffers chains	61	10
log file sync	3	9
SQL*Net message from client	22	7
latch: library cache pin	16	3
db file sequential read	70	3
latch: library cache	12	1
SQL*Net message to client	23	0

이 경우 buffer busy waits 대기의 발생은 freelists = 1 에 의한 Buffer Lock 의 경합으로 해석할 수 있다.

ASSM 을 사용하지않고 FLM(Manual 모드로 Segment Space 를 관리) 을 사용하는 경우 freelists 의 기본설정 값이 1 로 설정된다.

freelists = 1 이라는 것은 곧 여러개의 세션이 하나의 freelist 로부터 새로운 Block 을 할당 받는다는 의미이다. 이 경우 여러 개의 세션이 동시에 Insert 를 수행하면 모두 하나의 freelist 에서 Block 을 할당 받아 Buffer 를 생성하므로 같은 Buffer 에 대해 Exclusive 하게 Buffer Lock 을 획득하기 위해 경쟁하게 된다(HW Lock Article 에서 Freelist 에 대해 비교적 자세히 설명하고 있으므로 참조하기 바란다).

v$session_wait 뷰를 캡쳐 하면 다음과 같이 p3(class) 가 대부분이 1 이며 간혹 4 인 것을 확인할 수 있다.

```
sid=9, event = buffer busy waits, p1 = 11, p2 = 1666, p3 = 1
sid=15, event = buffer busy waits, p1 = 11, p2 = 1662, p3 = 1
sid=17, event = buffer busy waits, p1 = 11, p2 = 1665, p3 = 1
...
sid=77, event = buffer busy waits, p1 = 11, p2 = 9, p3 = 4
sid=83, event = buffer busy waits, p1 = 11, p2 = 9, p3 = 4
...
sid=118, event = buffer busy waits, p1 = 11, p2 = 1665, p3 = 1
sid=131, event = buffer busy waits, p1 = 11, p2 = 1665, p3 = 1
sid=151, event = buffer busy waits, p1 = 11, p2 = 9, p3 = 4
```

p3(class) 가 1 인 경우는 data block 에 해당하므로 대부분의 buffer busy waits 대기가 테이블의 같은 Block 을 업데이트하려는 과정에서 발생함을 알수 있다.

p3(class) 가 4 인 경우는 segment header block 에 해당한다. Insert 를 수행하면서 segment header block 을 변경하는 것은 freelist 정보를 변경하거나 High water mark 를 변경하기 위한것이다. HW Enqueue 대기는 High water mark 를 변경하는 과정에서 발생한다.

위의 테스트결과를 보면 freelists = 1 의 기본설정 값을 사용하는것이 얼마나 위험한것인지 알수 있다. HW Enqueue 대기와 buffer busy waits 대기가 모두 잘못된 freelist 값 설정에서 비롯한다.

동일한 테스트를 ASSM 을 사용하는 환경에서 수행해보자.

-- Auto Segment Space Management 를 사용하는경우

```
create tablespace bfw_tbs datafile '/home/oracle/oradata/10gr2/ORA102/ukja_test_01.dbf' size 50M autoextend on
extent management local uniform size 1M
segment space management auto;
create table bfw_test(id char(1000)) tablespace bfw_tbs ;
```

--동일한 Insert 수행

[테스트결과]

```
EVENT                          TOTAL_WAITS TIME_WAITED
------------------------------ ----------- -----------
events in waitclass Other              193        2304
free buffer waits                      817        1596
buffer busy waits                     1964        1237
log buffer space                        44         830
enq: HW - contention                   137         390
log file switch (private stran           3         130
d flush incomplete)
log file switch completion               2         108
log file sync                            4          47
latch: cache buffers chains             65          15
...
```

buffer busy waits 대기의 경우 만족스럽지는 않지만 분명히 줄어든 것을 확인할 수 있다. HW Enqueue 대기는 크게 줄어들었다.

v$session_wait 뷰를 캡쳐하면 p3(class)가 다음과 같이 8번이 많은 비중을 차지하면 9번도 눈에 띈다.

```
sid=10, event = buffer busy waits, p1 = 11, p2 = 11, p3 = 9
sid=11, event = buffer busy waits, p1 = 11, p2 = 1802, p3 = 8
sid=55, event = buffer busy waits, p1 = 11, p2 = 1802, p3 = 8
sid=66, event = buffer busy waits, p1 = 11, p2 = 1802, p3 = 8
sid=77, event = buffer busy waits, p1 = 11, p2 = 1913, p3 = 1
sid=118, event = buffer busy waits, p1 = 11, p2 = 11, p3 = 9
sid=127, event = buffer busy waits, p1 = 11, p2 = 1913, p3 = 1
sid=130, event = buffer busy waits, p1 = 11, p2 = 1802, p3 = 8
sid=132, event = buffer busy waits, p1 = 11, p2 = 1802, p3 = 8
```

8, 9 번에 해당하는 Block Class는 다음과 같다.

```
p3(class#) = 8 : 1st level bmb
p3(class#) = 9 : 2nd level bmb
```

ASSM을 사용하는 경우 Segment space 관리에 사용되는 Bitmap Block 에 대한 Buffer Lock 경합이 발생함을 알수 있다. 특히 1st level bmb Block이 Leaf Block으로 많은 변경이 발생하게 되므로 해당 block 에서 p3 = 8 에 해당하는 buffer busy waits 대기가 많이 발생하게 된다.
(ASSM에서의 Bitmap block 에 대한 자세한 설명은 참조 문서 중 Poder 의 자료를 참조한다)

즉, ASSM을 사용하지 않는 경우에는 잘못된 freelists 지정으로 인해 Data block 또는 Segment Head block 에 대한 buffer busy waits 대기가 많이 발생하는 반면, ASSM을 사용하는 경우에는 Bitmap block 에 대한 buffer busy waits 대기가 많이 발생함을 알수 있다.

다행히 ASSM을 사용하면 buffer busy waits 가 전반적으로 줄어드는 효과가 있으며 특히 HW Enqueue 대기가 크게 줄어드는 효과가 있다.

Insert / Insert 에 의한 buffer busy waits 대기를 줄이는 방법은 다음과 같이 정리할 수 있다.

- ASSM을 사용할 수 없는 환경이라면 freelists, freelist groups의 값을 시스템의 부하를 고려해서 적절히 부여한다. freelist 와 관련된 속성을 기본값으로 사용하는 것은 매우 위험하다.

- 9i 이상부터는 가능하면 ASSM을 사용한다. ASSM을 사용하는 경우에는 어떤 환경에서도 극단적인 성능저하를 피할 수 있다.

D. Update / Update 에 의한 Buffer Busy Waits

동시에 여러개의 세션이 같은 Row를 업데이트하는 것은 기본적으로 TX Enqueue 에 의해 동기화가 이루어진다. 하지만 동시에 여러 개의 세션이 서로 다른 Row를 업데이트하는 경우에도 만일 해당 Row 들이 같은 Block 안에 있다고 하면 Buffer Lock 에 의한 동기화가 필요하다. 이 경우에는 발생하는 Buffer Lock 에 의한 경합은 TX Lock 에 의한 경합과는 전혀 다른 성격을 지니고 있으므로 그 현상을 해석할 때 주의하여야 한다.

가령 TX Lock 이 발생하는 경우에는 Lock 을 획득한 Lock Holder 의 Transaction 을 종료(commit, rollback, kill) 하는 것만이 유일한 해결책이다. 하지만 동일 Block 을 변경하는 과정에서 발생하는 Buffer Lock 경합은 그 해결책이 전혀 다르다.

먼저 Update / Update 에 의한 Buffer busy waits 대기현상을 테스트해보자.

[테스트스크립트]

-- 여러 개의 row 가 같은 block 에 있으면서 동시에 여러 세션이 같은 Row를 업데이트하지 않으면서 같은 Block 을 업데이트하게끔 하는 것이 테스트의 요점이다. 테스트 스크립트가 복잡할 수도 있으므로 유의해서 보기 바란다.

```
create tablespace bfw_tbs datafile size 50M autoextend on
extent management local uniform size 1M
segment space management auto;
```

-- Block Size 가 8K 인 점을 고려해서 적절한 크기의 Row 가 되게끔 설정

```
create table bfw_test(id number, name char(700))
tablespace bfw_tbs;
```

위와같이 table을 생성하면 정확하게 10개의 Row가 하나의 Block에 들어가게된다.

--총 100건(=10 Block)의 데이터를 생성한다.

```
begin
for idx in 1 .. 100 loop
  insert into bfw_test values(idx, ' ');
end loop;
end;
/
```

--Block별로 Row수를 확인해보면 아래와같이 정확하게 10개씩 들어가있음을 확인할수 있다.

```
SQL> select dbms_rowid.rowid_block_number(rowid) as block_no, count(*)
          from bfw_test
          group by dbms_rowid.rowid_block_number(rowid);

 BLOCK_NO  COUNT(*)
--------------------
       12         10
       17         10
       16         10
       18         10
       13         10
       14         10
       19         10
       10         10
       11         10
       15         10
```

--위와같이 Block을 생성한 후 10개의 독립적인 세션이 TX Lock을 유발하지 않으면서 즉 동일 Row를 업데이트하지 않으면서 동일 Block을 업데이트하게끔 한다. 가령 1번 세션은 id(1,11,21,...,91)을 업데이트하고, 2번 세션은 id(2,12,22,...,92)를 업데이트하는 식이다.

```
create or replace procedure bfw_do_update(p_idx in number)
is
begin
     for n in 1 .. 1000 loop
        for idx in 1 .. 10 loop
          update bfw_test set name = ' ' where id = 10*(idx-1) + p_idx;
                    commit;
        end loop;
     end loop;
end;
/

connect maxgauge/maxgauge@ora10gr2
select sid from v$mystat where rownum = 1;
var job_no number;

begin
    for idx in 1 .. 9 loop
         dbms_job.submit(:job_no, 'bfw_do_update('||idx||');');
              commit;
    end loop;
end;
/

exec bfw_do_update(10);
@event;
```

[테스트결과]

아래와같이 buffer busy waits 대기이벤트가 매우광범위하게 나타난다. 이 세션의 전체수행시간은 67초였는데 buffer busy waits 대기이벤트에 45초를 소비하고있으므로 대단히 심각한수준이라고 할수있다.

```
EVENT                          TOTAL_WAITS TIME_WAITED
------------------------------ ----------- -----------
buffer busy waits                     3616        4951
enq: HW - contention                   103        1058
events in waitclass Other              882         901
latch: cache buffers chains           1714         680
latch: In memory undo latch            905         573
log file switch (checkpoint in complete)  4         304
Data file init write                    15          75
latch: row cache objects               125          62
control file parallel write             15          38
...
```

v$session_wait 뷰를 캡처 해보면 아래와 같다.

```
sid=101, event = buffer busy waits, p1 = 11, p2 = 907, p3 = 1
sid=110, event = buffer busy waits, p1 = 11, p2 = 907, p3 = 1
sid=112, event = buffer busy waits, p1 = 11, p2 = 907, p3 = 1
sid=120, event = buffer busy waits, p1 = 11, p2 = 907, p3 = 1
sid=123, event = buffer busy waits, p1 = 6, p2 = 2617, p3 = 101
sid=131, event = buffer busy waits, p1 = 11, p2 = 907, p3 = 1
sid=133, event = buffer busy waits, p1 = 11, p2 = 907, p3 = 1
sid=140, event = buffer busy waits, p1 = 11, p2 = 907, p3 = 1
```

우선, 동일한 11번 Block에 대해 Buffer Lock을 Exclusive하게 획득하기 위해 대기하는 것을 확인할수 있다.

즉, 비록서로 다른 Row를 업데이트하지만 많은 세션이 같은 Block을 변경하는것만으로 매우 심각한 성능문제를 야기하게 됨을 확인할 수 있다.

이것은 Index의 경우에도 마찬가지다. Index Segment의 동일 Leaf Block에 대해 Block변경이 동시다발적으로 발생하는 경우에도 buffer busy waits 대기에 의한 성능저하가 생기게 된다.

둘째로 Undo Segment Header block에 대한 경합도 많이 보인다. 이것은 Update문에 의해 Undo를 생성하면서 Header block을 변경하는과정에서 발생한다.

Update / Update 에 의해 Buffer Lock 경합이 발생하는 경우에는 다양한 해결책들이 제시되고 있다. (해결책이 많다는 것은 거꾸로 Update /Update 에 의한 경합이 매우 보편적이라는 것을 반증한다)

Buffer Lock 경합이 발생하는 원인이 서로 다른 Row 가 같은 Block 에 있다는 데서 기인하므로 서로 다른 Row 를 서로 다른 Block 에 흩어지게끔 분산시키는 방법이 가장 보편적으로 사용된다. Row 를 분산시키는 방법은 여러 가지가 있다.

- PCTFREE 를 높게 준다. 이 방법은 Row 를 분산시키는 가장 확실한 해결책이지만 공간의 낭비를 초래하고 그로 인해 Table Full Scan 이나 Index Full Scan/Range Scan 의 성능에 영향을 주게 된다. 뿐만 아니라 동일한 데이터를 처리하기 위해 생성해야 할 Block 의 수가 늘어나므로 SGA 내의 Buffer Cache 에 대한 낭비를 초래하고 이로 인해 cache buffers chains latch 경합과 관련된 성능 문제를 야기할 수 있다.

- Partition 기법을 사용하여 물리적으로 다른 Block 으로 흩어지게끔 한다. 이 경우 PCTFREE 를 높게 주는 방법에 비해 공간을 낭비하는 문제는 없지만 업무 로직 변경에 의해 Update 하는 방식이 바뀌면 같은 문제가 재현될 가능성이 있다.

- 작은 Block 사이즈를 사용한다. 이것은 Block 사이즈가 작으면 자연스럽게 한 Block 안에 들어가있는 Row 수가 줄어든다는 것을 이용하는 것이다. Oracle9i 부터는 서로 Block 사이즈가 다른 Tablespace 를 생성할 수 있다. 따라서 작은 Block 사이즈를 사용하는 Tablespace 를 생성한 다음 문제가 발생한 Table 이나 Index 를 Move 함으로써 문제를 해결할 수 있다. 하지만 이 경우에도 Table Full Scan 이나 Index Full Scan/Range Scan 의 성능에는 부정적인 영향을 준다. 또한 Block 의 수가 늘어나므로 cache buffer chain 과 관련된 성능 문제를 야기할 수 있다.

하지만 위의 해결책을 적용할 때는 대단히 조심해야 한다.

가령, 대부분의 책이나 자료에는 PCTFREE 를 높게 줌으로써 쉽게 문제를 해결할 수 있다고 되어있으나, 높은 PCTFREE 에 의해 야기되는 다른 성능상의 문제에 대해서는 별다른 언급을 하지 않고 있다.

PCTFREE 를 높게 주는 경우 Buffer busy waits 대기에 주는 영향이 어느 정도인지 테스트해보자..

우선, 아래와 같이 PCTFREE 를 매우 높게 주었다.

```
create table bfw_test(id number, name char(700))
pctfree 90 pctused 10
tablespace bfw_tbs;
```

위와 같이 테이블을 생성한 후 100 개의 Row 를 insert 하면 정확하게 한 Block 당 하나의 row 가 생성된다.

이 테이블에 대해 동일한 테스트를 수행하면 그 결과는 다음과 같다.

[테스트결과]

EVENT	TOTAL_WAITS	TIME_WAITED
latch: cache buffers chains	318	2637
enq: HW - contention	90	1770
buffer busy waits	1120	1696
events in waitclass Other	947	890
latch: In memory undo latch	389	382
Data file init write	5	320
control file parallel write	51	129
log file switch completion	13	119
latch: library cache	39	85
latch: row cache objects	44	46
latch: library cache pin	28	27

...

buffer busy waits 는 줄어들었지만 기대한 만큼 크게 줄어들지는 않았고, 늘어난 Buffer (Block) 의 수만큼 cache buffers chains latch 경합이 크게 증가해서 전체 성능면에서는 큰 개선효과가 없어진다. 한가지 재미있는 것은 보통 cache buffers chains latch 경합을 해소하는 방법 중의 하나로 PCTFREE 를 높이는 것을 권고한다는 것이다. PCTFREE 를 높임으로써 데이터의 분산이 이루어지고, latch 의 경합도 줄어드는 경우가 있기 때문이다. 하지만 이번 테스트에서는 Block 의 수가 늘어남으로써 오히려 latch 경합이 증가하는 것을 목격할 수 있다. 9i 이후로는 cache buffers chains latch 는 읽기전용인 경우에는 공유가 가능하므로 경합이 줄어드는 효과가 있다. 하지만 본 테스트와 같이 변경을 위해 접근하는 경우에는 Exclusive 모드로 latch 를 획득해야 하기 때문에 latch 경합이 크게 증가한 것으로 해석할 수 있다.

PCTFREE 를 크게 해서 Row 를 분산시켰음에도 불구하고 buffer buys waits 가 여전히 높은 이유는 무엇일까? 이것은 앞서 언급한 것처럼 Update 에 의한 Undo Segment Header Block 에 대한 경합은 없어지지 않으며, Block 수가 늘어남에 따라 오히려 생성해야 할 Undo Block 수가 늘어났기 때문이다.

실제로 v$session_wait 뷰를 캡쳐 하면 다음과 같은 결과가 나타난다.

```
sid=113, event = buffer busy waits, p1 = 8, p2 = 2, p3 = 13
sid=123, event = buffer busy waits, p1 = 8, p2 = 2, p3 = 13
sid=130, event = buffer busy waits, p1 = 8, p2 = 2, p3 = 13
...
sid=110, event = buffer busy waits, p1 = 6, p2 = 2617, p3 = 101
sid=120, event = buffer busy waits, p1 = 6, p2 = 1865, p3 = 87
sid=123, event = buffer busy waits, p1 = 6, p2 = 1017, p3 = 73
```

위의 결과는 Undo Header 에 대한 경합이 발생하고 있다는 것을 말한다. p1 (file#) 6, 8 번은 모두 Undo data file 번호이다.

높은 PCTFREE 에 의해 Block 수가 늘어나면서 Update 에 의한 Undo Block 수가 늘어나면서 Undo Segment Header Block 에 대한 경합이 늘어난 것이 Buffer busy

waits 대기가 크게 줄어들지 않은가장 큰 이유이다.

위의 테스트결과를정리해보면 Update / Update 에 의한buffer busy waits를 줄이기 위해높은PCTFREE 값을 사용하는것은cache buffers chains latch 경합을 증가시키고Undo header block 에 대한 경합을 증가시킬 우려가있다는 것이다.

따라서, buffer busy waits 문제를 해결하기 위해 Storage 속성을 바꿀 때는충분한 테스트를 거쳐 다른 성능상의부정적인 효과가 없는지를 충분히 검토해서 적용해야한다.

그렇다면 Partitioning 기법을 이용해Block 수를 늘이지 않으면서Row를 분산시키는 효과를 가져올 수 있는 방법을 사용하면 어떨까?

아래테스트결과를 보자.

[테스트스크립트]

-- Hash partitioned table 을 생성

```
create table bfw_test(id number, name char(700) default  ' ')
partition by hash(id)
partitions 5
tablespace bfw_tbs;
```

-- 100건을 생성

```
begin
for idx in 1 .. 100 loop
  insert into bfw_test values(idx, ' ');
end loop;
end;
/
```

-- Block 별로 Row수를 확인해보면 적절히 분포가 이루어졌음을알수 있다.

```
SQL>  select  dbms_rowid.rowid_block_number(rowid)  as  block_no,
count(*)
         from bfw_test
         group by dbms_rowid.rowid_block_number(rowid);

    BLOCK_NO   COUNT(*)
  ---------- ----------
         524          5
         651         10
         652          7
         778          9
         395         10
         650         10
         266         10
         394         10
         396          8
         267          1
         522         10
         523         10
```

-- 위의 테이블에 대해동일하게 Update / Update 를 수행한다.

[테스트결과]

```
EVENT                          TOTAL_WAITS   TIME_WAITED
---------------------------------------------------------
buffer busy waits                     1675          3177
enq: HW - contention                    98          1387
events in waitclass Other              664           639
latch: cache buffers chains            549           364
latch: In memory undo latch            410           268
Data file init write                    21           122
log file switch completion              10            85
control file parallel write             21            57
latch: row cache objects                67            17
latch: library cache                    24            11
latch: library cache pin                28             9
...
```

위의 테스트결과를 보면Partitioning 을 한 경우Partitioning 을 하지 않은경우에 비해 Buffer busy waits 가 다소 줄었지만 만족할만한 수치는 아니다. Partitioning 을 이용해 Buffer Lock 대기를 줄이려면 Update 하는 방식과 Partition을 나누는 방식을 고려해서 최적의 분산효과가 나게끔 설계를 해주어야한다. 위의 테스트스크립트에서 확인할수 있듯이 현재 업데이트 방식이 1,11,21,..,91 을 하나의 세션에서업데이트하고, 2,12,22, ..,92를 다른세션에서 업데이트하는 식이므로 하나의Block에 1,11,21,...,91이 들어가고또 다른Block에 2,12,22,..92 가들어가게끔 배치하는 것이가장 이상적인Partitioning 이라고 할 수 있다.

아래의 테스트스크립트와 테스트결과를 보자.

[테스트스크립트]

-- List Partition 을 이용해서최적화된 방식으로 데이터가 들어가게끔 한다.

```
create table bfw_test(id number, name char(700) default  ' ')
partition by list (id)
(
partition id_1 values(1,11,21,31,41,51,61,71,81,91),
partition id_2 values(2,12,22,32,42,52,62,72,82,92),
partition id_3 values(3,13,23,33,43,53,63,73,83,93),
partition id_4 values(4,14,24,34,44,54,64,74,84,94),
partition id_5 values(5,15,25,35,45,55,65,75,85,95),
partition id_6 values(6,16,26,36,46,56,66,76,86,96),
partition id_7 values(7,17,27,37,47,57,67,77,87,97),
```

```
partition id_8 values(8,18,28,38,48,58,68,78,88,98),
partition id_9 values(9,19,29,39,49,59,69,79,89,99),
partition id_10 values(10,20,30,40,50,60,70,80,90,100),
partition id_rest values(default)
)
tablespace bfw_tbs;
```

-- 100 건을 생성

```
begin
for idx in 1 .. 100 loop
  insert into bfw_test values(idx, ' ');
end loop;
end;
/
```

-- 원하는대로 데이터가 들어가있는지 확인해본다. 10 block에 골고루분산되었음을 알 수 있다.

```
SQL>  select  mod(id,10),  dbms_rowid.rowid_block_number(rowid)  as
block_no, count(*)
          from bfw_test
          group by mod(id,10), dbms_rowid.rowid_block_number(rowid)
          order by 1

MOD(ID,10)   BLOCK_NO COUNT(*)
---------- ---------- ----------
         0       2954         10
         1       1802         10
         2       1930         10
         3       2058         10
         4       2186         10
         5       2314         10
         6       2442         10
         7       2570         10
         8       2698         10
         9       2826         10
```

-- 이 상태에서 위의 테스트와 동일하게 업데이트를 수행한다.

[테스트결과]

결과는 매우극적이다. Buffer busy waits 대기가 완전히 사라진 것을 확인할 수 있다.

```
EVENT                                  TOTAL_WAITS    TIME_WAITED
---------------------------------------- ----------  -----------
log file sync                             9834           9315
events in waitclass Other                  259            126
latch: In memory undo latch                196            125
latch: cache buffers chains                109            102
latch: library cache pin                    32             28
SQL*Net message from client                 22              8
latch: library cache                        25              4
db file sequential read                      1              0
SQL*Net message to client                   23              0
```

Update 방식을 고려하여 최적의 Partition 을 구성한 결과 Buffer Lock 경합이 완전히 사라진 것을 확인할 수 있다.

위에서 살펴본 바와 같이 Update /Update 에 의한 Buffer busy waits 대기를 줄이기 위해 제시되고 있는 다양한 해결책들이 제시되고 있지만, 기본적인 테스트나 개념이해 없이 무작정 적용하는 것은 바람직하지 못하다. 이러한 해결책들을 기본으로 하되, 다양한 테스트를 통해 최적의 해결책을 찾는것이 중요하다.

지금까지의 테스트 결과와 분석 결과를 토대로 Buffer Lock 경합을 줄이는 방법을 정리하면 다음과 같다.

- select / select 에 의해 발생하는 read by other session 대기를 줄이는 최선의 방법은 SQL 최적화를 통해 가장 적은 I/O 로 원하는 결과를 얻는것이다. 이 작업이 선행되어도 해결이 되지 않는다면 SGA(Buffer cache) 의 크기가적절한지 점검해보아야 한다.

- Select / Update 에 의해 발생하는 read by other session 대기는 Select / Select 에 의한 read by other session 대기와 해결책이 동일하다.

- insert / insert 에 의해 발생하는 buffer busy waits 대기는 적절한 Space 관리 기법을 사용함으로써 해결이 가능하다. 9i 이상의 버전이라면 ASSM 을 사용할 것을 권장한다. 8i 라면 freelists 속성을 적절히 지정해야 한다. Transaction 의 양에 비해 freelists 값이 작은 경우에 buffer lock 에 의한 경합이 광범위하게 나타난다. freelists 값만으로 해결이 되지 않을 때는 _BUMP_HIGHWATER_MARK_COUNT 히든 파라미터 값을 크게 해주는것 또한 도움이 된다.

- update /update 에 의해 발생하는 buffer busy waits 대기는 동일 Block 에 대해 동시에 업데이트가 이루어지지 않게끔 개선함으로써 해결이 가능하다. Update 형태를 고려한 최적의 Partitioning 을 구성하는 것이 좋은 해결책이 된다. PCTFREE 를 높게 주거나 작은 크기의 Block 사이즈(9i 부터 가능)를 사용함으로써 Block 을 분산시킬 수 있으며 이로 인해 Buffer Lock 경합을 줄어들 수 있다. 하지만 위의 테스트결과에서 보듯이 테스트를 통해 사이드 이펙트가없는지 충분한 검토가필요하다.

<참고 문헌>

OWI 를 활용한진단&튜닝; 엠섹 역; MacGraw-Hill Korea; 2005
Oracle 8i Internal Services for Waits, Latches, Locks, and Memory; Steve Adams; O'reilly, 1999
Oracle Database Concept; Oracle
Oracle Database Reference; Oracle
http://metalink.oracle.com
http://integrid.info