11g - Serial direct read 동작원리

㈜엑셈 컨설팅본부/DB컨설팅팀 임 경석

개요

오라클 11g 에서 처음 소개된 Serial direct read 는 대량의 데이터를 처리하는 엑사데이타에서 매우 중요한 기능이다. 스마트 스캔을 하기 위해서는 반드시 테이블 full scan 과 direct path read 가 선행되어야 하기 때문이다. 그러나, serial direct read 의 동작은 옵티마이져 환경에 영향을 받지 않으므로 힌트를 적용하여 제어할 수 없고, 뜻대로 동작하지 않거나 중간에 변경되기도 한다. 그렇다고 임의로 동작하는 것은 아니며 몇 가지 규칙이 존재한다. 필자는 serial direct read 가 어떠한 규칙에 의해 결정 되고, 변경 되는지 몇 가지 사례를 통해 자세히 소개하고자 한다.

Serial direct read 란?

DW 환경에서는 대량의 데이터를 조회하는 경우가 빈번하다. 따라서, 테이블이나 인덱스에 대한 풀 스캔을 피할 수 없다. 오라클에서 풀 스캔은 여러 개의 블록을 한번에 버퍼캐시로 읽어 들이는 멀티블록 I/O 를 통해 처리된다. 멀티블록 I/O 는 크기가 제한된 버퍼캐시를 통해 이루어지므로 내부적으로 복잡한 과정이 필요하므로 버퍼캐시에 상당한 부담이 될 수 있다. 블록을 읽기 위해 프리리스트를 검색하고 버퍼를 변경할 때 마다 래치를 획득하여 cache buffers chains latch, cache buffers lru chain latch 와 같은 이벤트가 발생할 수 있다. 이러한 블록들은 LRU 리스트 끝에 위치 하지만 다른 세션들이 필요로 하는 캐시 된 블록들을 age out 시키고, 이미 변경된 블록들에 대해서는 다량의 CR 복사본을 생성할 수 있다.

따라서, 이러한 이유들로 인해 일반적으로 테이블을 풀 스캔 할 경우 버퍼캐시를 거치지 않는 PQ(Parallel Query)를 많이 사용한다. PQ 슬레이브 세션들은 direct path read 를 통해 PGA 로 데이터를 직접 읽어 들이므로 버퍼캐시에 대한 부담을 줄여주고 성능에도 효과적이다. 하지만,

과도한 PQ 사용은 CPU, MEMORY 사용률 증가와 빈번한 체크포인트로 인해 오히려 시스템 부하를 가중시킬 수 있다.

오라클은 Parallel 옵션을 사용하지 않고 direct read 가 가능한 SDPR(serial direct path read)을 11g 에서 처음 소개하였다. SDPR 은 테이블 full scan 이나 인덱스 fast full scan 시 싱글 모드로 버퍼캐시를 거치지 않고 블록을 직접 읽어 들이는 것을 말한다. 사실 이러한 기능은 오라클 10G 에도 있었지만 본격적으로 적용된 것은 11g 부터 이다. 특히, 11gr2 이후에는 몇 가지 파라미터를 통해 적절한 제어도 가능해졌다.

serial direct read 는 히든파라미터 _serial_direct_read 로 정의한다. 10G 에는 기본값이 false 였지만, 11gr2 부터 auto 로 변경되면서 오라클이 적절하게 direct read 를 적용할 수 있게 되었다.

_serial_direct_read 옵션은 true, false, auto, always, never 로 크게 5 가지로 정의할 수 있다. 이중 true, always 는 풀 스캔 하는 모든 세그먼트를 항상 SDPR 로 처리하고, 반대로 never 는 어느 경우든 처리하지 않는다. 하지만, false 의 경우 never 와 같은 의미로 보이지만 오히려 auto 와 비슷하게 동작한다.

```
NAME                           VALUE   DESCRIPTION
-------------------------------------------------------------
_serial_direct_read        AUTO   enable direct read in serial
```

어떻게 동작 하는가?

데이터베이스에 존재하는 모든 테이블이 SDPR 의 대상이 되는 것은 아니다. 오라클은 SDPR 이 가능한 테이블의 선택 기준을 히든파라미터 _small_table_threshold 에 정의하였다. 기본값은 _db_block_buffers 의 2% 로써 이보다 작으면 오라클이 작은 테이블로 인식한다는 의미이다. 10g 에서는 테이블을 풀 스캔 할 때 작은 테이블은 LRU 리스트의 중간에 위치하고, 큰 테이블은 LRU 리스트의 맨 끝에 위치하므로 age out 되기가 쉬워 physical read 가 발생할 확률이 높음을 의미했다. 그러나, 11gr2 에서 해당 파라미터는 direct read 가 가능한 테이블 크기의 최소 임계치로 정의한다.

```
NAME                           DESCRIPTION
------------------------- ------------------------------------------------
_small_table_threshold     lower threshold level of table size for direct reads
```

테이블을 풀 스캔 하면 먼저 테이블의 세그먼트 헤더블록을 읽어 HWM 이하 차지하는 블록이 몇 개인지 조회한다. 블록의 개수가 _small_table_threshold 보다 크면 direct path read 로 처리되고, 이보다 작으면 버퍼캐시를 통한 멀티블록 I/O 로 인해 db file scattered read 로 처리된다.

11gR2 이전에는 테이블의 크기가 _small_table_threshold 설정 값의 5 배 또는 그 이상이 되어야 SDPR 이 가능하였다. 그러나, 11gR2 이후에는 _small_table_threshold 값과 테이블의 크기가 거의 비슷해지는 순간 발생한다. 또한, 테이블의 전체 블록 수와 비교하여 버퍼캐시에 캐시 되어 있는 테이블의 블록 또는 더티 블록이 차지하는 비율에 따라 달라지기도 한다. 특정 테이블이 차지하는 전체 블록 중 50%이상이 버퍼캐시에 캐시되어 있거나 25% 이상이 더티 상태로 캐시 될 경우, 테이블의 전체 블록수가 _small_table_threshold 보다 크더라도 SDPR 은 발생하지 않는다. 이와 같은 사실은 SDPR 이 옵티마이져에 의한 것이 아닌 실제 세그먼트 블록을 조회하는 런타임 시점에 결정되기 때문이다. 따라서, 테이블의 크기, 버퍼캐시의 크기, 버퍼캐시에 캐시 된 테이블의 블록 수를 고려하여 SDPR 여부는 언제든 바뀔 수 있어 좀더 안정적인 성능을 보장할 수 있다.

11gR2 이후에는 이와는 다른 한가지 방식이 추가 되었다. SDPR 의 가능여부를 실제 테이블의 블록 수를 조회하여 결정하던 것을 테이블의 통계정보에 저장된 블록 수 (dba_tables.blocks) 로 결정하는 것이다. 이것은 히든 파라미터 _direct_read_decision_statistics_driven 로 제어할 수 있으며 기본값은 true 이다. 통계정보의 blocks 를 참조할 경우 통계정보가 바뀌지 않는 한 테이블의 크기에 변화가 있더라도 SDPR 결정에 영향을 주지 않는다. 따라서, 통계정보를 활용하여 SDPR 를 좀더 유연하게 적용할 수 있다. 하지만, 버퍼캐시에 캐시 된 테이블의 블록 수와 상태에 따라서는 여전히 영향을 받는다.

serial direct read 등장 배경에는 엑사데이타와 밀접한 관련이 있다. 아마도 엑사 테이타의 주요 기능인 스마트 스캔이 테이블을 direct read 로 풀 스캔 할 때 가능하기 때문일 것이다. SSD 플래시디스크 활용이 증가하면서 플래시디스크를 통한 direct read 는 I/O 속도를 한 단계 더 진화시켰다. 따라서, serial direct read 를 적절하게 사용하면 성능에 큰 도움이 될 수 있다.

(1) 테이블 크기 변화에 따른 serial direct read.

_small_table_threshold 를 기준으로 SDPR 이 결정되는 테이블의 최소크기는 어떻게 정해지는지 테스트를 통해 살펴보자. 한가지 주의할 것은 오라클 11gR2 의 경우, 정확한 테스트를 위해 테이블 통계정보를 생성하지 않거나 _direct_read_decision_statistics_driven=false 로 하여 통계정보 blocks 로 SDPR 이 결정되지 않게 해야 한다. 테스트로 사용되는 테이블은 한 블록에 한 개의 로우만 저장되도록 생성하고, 테이블 크기 변화에 따라 SDPR 발생여부를 측정할 수 있는 프로시져를 생성한다.

```
-- 오라클버전 확인

Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
PL/SQL Release 11.2.0.3.0 - Production
CORE    11.2.0.3.0      Production
TNS for Linux: Version 11.2.0.3.0 - Production
NLSRTL Version 11.2.0.3.0 - Production

-- 테스트 테이블생성.

SQL> create table sdpr_test (col varchar2(100)) pctused 1 pctfree 99 tablespace
users ;

-- 테이블의 크기를 측정하기 위한 프로시져 생성.

 create or replace procedure SDPR_PRO(
   p_start  in number default 0,
   p_step  in number,
   p_stop  in number default null
   ) is
   b_prd  number;
   b_prd1  number;
   b_prd2  number;
   b_blocks number:=0;
   b_start  number:=p_start;
   b_cnt number:=0 ;
   b_result number;

 begin
```

```
 dbms_output.enable(buffer_size=>10000);
 execute immediate 'truncate table sdpr_test';

 select value into b_prd1
 from v$sesstat st, v$statname sn
 where st.statistic#=sn.statistic#
 and sn.name = 'physical reads direct'  and st.sid = userenv('sid');

loop

  insert /*+ append */ into sdpr_test
  select rpad('*', 100, '*')
  from dual
  connect by level <= p_step + b_start;
  commit;

  b_blocks:=b_blocks + p_step + b_start;
  b_start:=0;

  execute immediate 'alter system flush buffer_cache';

  select /*+ full(sdpr_test) */ count(*) into b_cnt from sdpr_test;

    select value into b_prd2
    from v$sesstat st, v$statname sn
    where st.statistic#=sn.statistic#
    and sn.name = 'physical reads direct'
    and st.sid = userenv('sid');

    b_prd:=b_prd2 - b_prd1 ;

  exit when (b_prd > 0 or b_blocks > nvl(p_stop, b_blocks));

end loop;

  b_result:=b_blocks-p_step;

 dbms_output.put_line(chr(10)||'Block count for SDPR: '||b_result||' blocks');

end;
/
```

```
-- buffer.sql

accept obj prompt'enter the object name :'

SELECT o.owner ,
       o.object_name ,
       decode( state , 0 , 'free' , 1 , 'xcur' , 2 , 'scur' , 3 , 'cr' , 4 ,
'read' , 5 , 'mrec' , 6 , 'irec' , 7 , 'write' , 8 , 'pi' ) state ,
decode( bitand( flag , 1 ) , 0 , 'N' , 'Y' ) dirty ,
decode( bitand( flag , 16 ) , 0 , 'N' , 'Y' ) temp ,
decode( bitand( flag , 1536 ) , 0 , 'N' , 'Y' ) ping,
decode( bitand( flag , 16384 ) , 0 , 'N' , 'Y' ) stale,
decode( bitand( flag , 65536 ) , 0 , 'N' , 'Y' ) direct,
       COUNT( * ) blocks
FROM   sys.xm$bh b ,
       dba_objects o
WHERE  b.obj = o.data_object_id
AND    b.ts# >= 0
AND    o.object_name = UPPER( LTRIM( RTRIM( '&obj' ) ) )
AND    state = 1
GROUP  BY o.owner ,
       o.object_name ,
       state ,
decode( bitand( flag , 1 ) , 0 , 'N' , 'Y' )  ,
decode( bitand( flag , 16 ) , 0 , 'N' , 'Y' )  ,
decode( bitand( flag , 1536 ) , 0 , 'N' , 'Y' ) ,
decode( bitand( flag , 16384 ) , 0 , 'N' , 'Y' ) ,
decode( bitand( flag , 65536 ) , 0 , 'N' , 'Y' ) ,
       blsiz ;

-- _small_table_threshold 을 1000 으로 설정.

SQL> alter session set "_small_table_threshold"=1000 ;

-- sdpr_pro 프로시져 실행

SQL> exec sdpr_pro(p_start=>800,p_step=>1,p_stop=>1500);
Block count for SDPR: 976 blocks
```

```
-- 버퍼캐시 상태조회.

SQL> @buffer
enter the object name :sdpr_test
OWNER    OBJECT_NAM  STATE  DIRTY   TEMP   PING  STALE  DIREC   BLOCKS
-------- ---------- -----  -----  -----  -----  -----  ------  ------
SYS      SDPR_TEST   xcur      N      N      N      N      N         1
```

_small_table_threshold 를 1000 으로 하고 SDPR_TEST 테이블의 블록수가 800 블록부터 한 블록씩 증가하여 1500 블록까지 커지도록 프로시저를 실행한 결과, 테이블의 크기가 976 블록을 1 블록 초과하는 977 블록이 되는 순간 최초로 direct read 가 발생한다. 이것은 _small_table_threshold 가 1000 일 때 테이블 크기의 최소임계 치는 976 블록 임을 의미한다. 따라서, HWM 까지의 크기가 976 블록을 초과하는 테이블을 풀 스캔 한다면 direct path read 로 실행될 수 있다는 의미이다.

테스트 세션에 대해 10046 트레이스를 걸어 좀더 자세히 살펴보면, 제일먼저 db file sequential read 이벤트를 대기하면서 file#=4, block#=17098 블록을 읽는 것을 확인할 수 있다. 해당블록은 SDPR_TEST 테이블의 세그먼트 헤더블록으로 이를 통해 테이블이 차지하는 실제 HWM 이하까지의 블록 수를 조회한다. 테이블의 실제 블록 수를 _small_table_threshold 값과 비교하여 SDPR 을 수행하기로 결정되면, 뒤이어 더티 상태의 블록들을 디스크에 반영하는 체크포인트가 이루어지고 enq: KO - fast object checkpoint 이벤트를 대기한다. 체크포인트는 SDPR_TEST 테이블에 한하여 매우 짧은 시간 수행되고, 체크포인트가 끝나면 direct path read 로 테이블을 풀 스캔 하게 된다. 실제 버퍼캐시 상태를 x$bh 뷰를 통해 조회하면, SDPR_TEST 테이블의 전체 블록 중 세그먼트 헤더에 해당하는 블록 한 개만 버퍼캐시에 존재한다는 것을 알 수 있다.

```
-- 10046 세션 트레이스

WAIT #140630657766376: nam='db file sequential read' ela= 9 file#=4 block#=17098
blocks=1 obj#=80219 tim=1362977906994976
WAIT #140630657766376: nam='db file sequential read' ela= 6 file#=3 block#=192
blocks=1 obj#=0 tim=1362977906995033
```

```
WAIT #140630657766376: nam='reliable message' ela= 71 channel context=2736958984
channel handle=2736907704 broadcast message=2738048368 obj#=0
tim=1362977906995246
WAIT #140630657766376: nam='enq: KO - fast object checkpoint' ela= 483
name|mode=1263468550 2=65562 0=1 obj#=0 tim=1362977906995753
WAIT #140630657766376: nam='asynch descriptor resize' ela= 0 outstanding #aio=0
current aio limit=412 new aio limit=442 obj#=0 tim=1362977906995790
WAIT #140630657766376: nam='direct path read' ela= 24 file number=4 first
dba=17099 block cnt=5 obj#=80219 tim=1362977906995941
WAIT #140630657766376: nam='direct path read' ela= 39 file number=4 first
dba=16080 block cnt=8 obj#=80219 tim=1362977906996015
WAIT #140630657766376: nam='direct path read' ela= 96 file number=4 first
dba=16089 block cnt=15 obj#=80219 tim=1362977906996142
WAIT #140630657766376: nam='direct path read' ela= 43 file number=4 first
dba=16105 block cnt=15 obj#=80219 tim=1362977906996213
WAIT #140630657766376: nam='direct path read' ela= 9 file number=4 first
dba=16121 block cnt=7 obj#=80219 tim=1362977906996261
WAIT #140630657766376: nam='direct path read' ela= 11 file number=4 first
dba=17024 block cnt=8 obj#=80219 tim=1362977906996297
```

-- 버퍼캐시 상태 조회.

```
SQL> SELECT o.owner ,
  2         o.object_name ,
  3         decode( state , 0 , 'free' , 1 , 'xcur' , 2 , 'scur' , 3 , 'cr' , 4 ,
'read' , 5 , 'mrec' , 6 , 'irec' , 7 , 'write' , 8 , 'pi' ) state ,
  4         blsiz ,
  5         dbarfil ,
  6         dbablk
  7  FROM   sys.xm$bh b ,
  8         dba_objects o
  9  WHERE  b.obj = o.data_object_id
 10  AND    b.ts# >= 0
 11  AND    o.object_name = 'SDPR_TEST'
 12  AND    state = 1;

OWNER      OBJECT_NAM STATE      BLSIZ    DBARFIL     DBABLK
-----------------------------------------------------------
SYS          SDPR_TEST  xcur        8192          4        17098
```

```
-- SDPR_TEST 테이블의 세그먼트 헤더블록 조회.

SQL> select segment_name, header_file, header_block
  2  from dba_segments
3  where segment_name = 'SDPR_TEST';

SEGMENT_NAME       HEADER_FILE    HEADER_BLOCK
---------------   -------------   -------------
SDPR_TEST                    4          17098
```

(2) 테이블의 캐시된 블록 수 변화에 따른 serial direct read.

SDPR_TEST 테이블이 977 블록이상 커지더라도 항상 SDPR 로 수행되는 것은 아니다. SDPR_TEST 테이블이 현재 버퍼캐시에 얼마나 캐시 되어 있는지에 따라 SDPR 여부는 달라질 수 있다.

테이블의 전체 블록 중 50% 이상이 버퍼캐시에 이미 캐시 되어 있다면 더 이상 SDPR 은 발생하지 않는다. 테이블 전체블록을 디스크로부터 직접 읽는 것 보다 캐시 되어있는 블록을 읽는 것이 더 유리하다고 판단해서이다. 그러면, SDPR_TEST 테이블이 버퍼캐시에 어느 정도 캐시 되었을 때 SDPR 이 발생하지 않는지 테스트를 통해 살펴보자.

먼저, 버퍼캐시에 블록을 캐시 하기 위한 프로시저를 생성한다. SDPR_TEST 테이블 300 블록을 미리 캐시한 후, 캐시 된 블록이 한 개씩 증가하도록 프로시저를 실행하면 블록이 캐시된 블록이 몇 개일 때 SDPR 이 멈추는지 확인해 보자

```
-- 인덱스 생성.

SQL> create index idx_sdpr_test on sdpr_test (1) tablespace users;

-- 프로시져 생성.

create or replace procedure cached_sdpr_pro(
   p_start in number default 0,
   p_step in number default 1
   ) is
    b_v varchar2(100);
    b_trsh number:=0;
```

```
    b_prd  number;
    b_prd1  number;
    b_prd2  number;
    b_cnt number:=0;
    b_start number:=p_start;
    cursor b_cur is
    select /*+ index(sdpr_test idx_sdpr_test) */ * from sdpr_test ;
 begin
   dbms_output.enable(buffer_size=>10000);
   execute immediate 'alter system flush buffer_cache';

    select value into b_prd1
    from v$sesstat st, v$statname sn
    where st.statistic#=sn.statistic#
    and sn.name = 'physical reads direct'
    and st.sid = userenv('sid');

  open b_cur;

   loop
     for i in 1 .. p_step+b_start  loop
      fetch b_cur into b_v;
     end loop;
    b_trsh:=b_trsh+p_step+b_start;
    b_start:=0;
    select /*+ full(sdpr_test) */ count(*) into b_cnt from sdpr_test ;
    select value into b_prd2
    from v$sesstat st, v$statname sn
    where st.statistic#=sn.statistic#
    and sn.name = 'physical reads direct'
    and st.sid = userenv('sid');
    b_cnt:=b_prd2 - b_prd1 ;
    exit when b_cnt=b_prd or b_cur%notfound ;
    b_prd:=b_cnt;
   end loop;
  close b_cur;
  dbms_output.put_line(chr(10)||'Block count for impossible sdpr : '||b_trsh||'
blocks');
 end;
/
-- _small_table_threshold 을 1000 으로 설정.
```

```
SQL> alter session set "_small_table_threshold"=1000 ;

-- dynamic sampling 비활성화

SQL> alter session set optimizer_dynamic_sampling=0;

-- 프로시져 실행

SQL> exec cached_sdpr_pro(p_start=>300, p_step=>1);
Block count for impossible sdpr : 497 blocks
```

프로시저 실행결과 SDPR_TEST 테이블 전체 블록 중 50% (497/977*100 = 50.8)인 497 개 블록이 캐시 되는 시점에 더 이상 SDPR 은 발생하지 않는다. SDPR_TEST 테이블을 풀 스캔 할 때, 버퍼캐시에 이미 캐시 된 블록수가 497 개 이상 존재한다면, direct path read 가 아닌 버퍼캐시를 경유하는 db file scattered read 로 처리된다.

여기서 한가지 주의할 점은 테스트에 사용된 테이블은 통계정보가 존재하지 않으므로 프로시저를 처음 수행할 때 dynamic sampling 이 발생할 수 있다는 것이다. dynamic sampling 이 수행되면 테이블을 샘플링 하는 과정에 블록의 일부가 버퍼캐시에 미리 캐시된 채 프로시저가 실행될 수 있어 497 보다 작은 결과값이 나올 수 있다. 따라서, 정확한 결과값을 얻기 위해 프로시저를 한번 더 실행하거나 dynamic sampling 이 되지 않도록 optimizer_dynamic_sampling 파라미터 레벨을 '0'으로 변경하여야 한다.

(3) 캐시된 더티 블록수 변화에 따른 serial direct read 결정

테이블의 블록 중 25% 이상이 더티 블록 상태로 캐시 된 경우에도 SDPR 은 발생하지 않는다. 앞에서 설명했듯이 direct read 는 오브젝트 단위의 체크포인트를 동반하므로 테이블의 모든 더티 블록 들은 체크포인트의 대상이 된다. 따라서, 25% 이상 존재하는 더티 블록에 대해 체크포인트를 적용한 후 디스크로부터 다시 블록을 읽는 것 보다는 이미 캐시 된 상태의 블록을 읽는 것이 유리하다고 판단할 수 있다.

그럼 실제 몇 개의 더티 블록이 캐시될 SDPR 이 더 이상 발생하지 않는지 테스트를 통해 살펴보자. 더티 블록이 버퍼캐시에 캐시 되도록 프로시저를 생성한 후 어느 시점에 SDPR 이 멈추는지 프로시저를 수행한 결과값을 통해 확인해보자.

```
-- 더티 블록을 생성하기 위한 프로시저 생성.

create or replace procedure dirty_sdpr_pro (
    p_start in number default 0,
    p_step in number,
    p_stop in number default null
    ) is
    b_trsh number:=0;
    b_prd number:=0;
    b_prd1 number:=0;
    b_prd2 number:=0;
    b_cnt number:=0;
    b_start number:=p_start;
 begin

    dbms_output.enable(buffer_size=>10000);
    execute immediate 'alter system flush buffer_cache';

    select value into b_prd1
    from v$sesstat st, v$statname sn
    where st.statistic#=sn.statistic#
    and sn.name = 'physical reads direct'
    and st.sid = userenv('sid');

  loop
    b_trsh:=b_trsh+p_step+b_start;
    update sdpr_test set col=col where rownum <= b_trsh;
     commit;
     b_start:=0;
     select /*+ full(sdpr_test) */ count(*) into b_cnt from sdpr_test ;
    select value into b_prd2
    from v$sesstat st, v$statname sn
    where st.statistic#=sn.statistic#
    and sn.name = 'physical reads direct'
    and st.sid = userenv('sid');
```

```
    b_cnt:= b_prd2 - b_prd1 ;
    exit when b_cnt=b_prd or b_trsh > nvl(p_stop, b_trsh);
    b_prd:=b_cnt;

  end loop;
    dbms_output.put_line(chr(10)||'Block count for impossible sdpr :
'||b_trsh||' blocks');
 end;
/

-- _small_table_threshold 을 1000 으로 설정.

SQL> alter session set "_small_table_threshold"=1000 ;

SQL> alter session set optimizer_dynamic_sampling=0 ;

-- 프로시저 실행

SQL> exec dirty_sdpr_pro(p_start=>50, p_step=>1, p_stop=>1500) ;
Block count for impossible sdpr : 245 blocks

SQL> @buffer
enter the object name :sdpr_test


OWNER   OBJECT_NAM   STATE   DIRTY   TEMP   PING  STALE  DIREC   BLOCKS
------- -----------  ------ ------- ------ ------ ------ ------- ------
SYS      SDPR_TEST    xcur       Y     N     N     N       N        245
SYS      SDPR_TEST    xcur       N     N     N     N       N        733
```

프로시저 수행결과, SDPR_TEST 테이블 전체블록 중 25% (245/977*100 = 25)인 245 개의 더티 블록이 캐시 된 시점에 더 이상 SDPR 은 발생하지 않는다. x$bh 뷰를 조회해 보면 SDPR_TEST 테이블의 더티 블록이 정확히 245 개 캐시 되는 시점에 direct path read 가 멈추고 전체 977 블록 중 나머지 733 개 블록이 멀티블록 i/o 로 인해 캐시 됐음을 확인할 수 있다.

(4) 통계정보 와 serial direct read

오라클 11gR2 이후에는 옵티마이져 통계정보에 저장된 tab$blkcnt, tabpart$.blkcnt, ind$.leafcnt 를 참조하여 serial direct read 여부가 결정된다.

통계정보를 적용하여 SDPR 을 결정하기 위해서는 _direct_read_decision_statistics_driven 히든 파라미터가 true 여야 한다. 해당 파라미터는 11gr2 에 새로 추가되었으며 기본값이 true 이다. 이것은 11gr2 이후에는 기본적으로 통계정보를 참조하여 SDPR 을 결정하겠다는 의미이다. 통계정보에 저장된 테이블의 blocks 가 _small_table_threshold 값보다 크면 SDPR 로 처리되고 작으면 버퍼캐시를 경유하는 멀티블록 I/O 로 처리된다.

```
NAME                                          VALUE       DESCRIPTION
--------------------------------------------- ----------- ------------------------
_direct_read_decision_statistics_driven       TRUE        enable direct read
                                                          decision based on
                                                          optimizer statistics
```

통계정보가 존재하는 테이블의 경우 테이블의 전체 블록 수에 큰 변화가 있더라도 SDPR 은 도중에 변경되지 않는다. SDPR 의 판단기준이 더 이상 테이블의 실제 블록 수를 통해 적용되지 않기 때문이다. 하지만, 테이블의 통계정보가 새로 갱신되거나 버퍼캐시에 캐시 된 테이블 블록 수와 더티 블록의 수에 변화가 생긴다면 SDPR 여부는 언제든 바뀔 수 있다. 이것은 앞에서 설명했듯이 SDPR 결정이 옵티마이져 환경이 아닌 런타임 시점에 결정되기 때문이다.

테이블 통계정보의 blocks 를 _small_table_threshold 값보다 크게 설정하여 테이블을 풀 스캔할 때 항상 SDPR 로 실행되게 하고, 테이블의 크기, 버퍼캐시 상태에 따라 SDPR 이 어떻게 적용되는지 테스트를 통해 살펴보자.

```
-- _small_table_threshold 를 1000 으로 설정.

SQL> alter session set "_small_table_threshold" = 1000 ;

-- SDPR_TEST 테이블 통계정보의 blocks 를 _small_table_threshold 보다 큰 1100 으로
설정

SQL>EXECDBMS_STATS.SET_TABLE_STATS(ownname=>'SYS',tabname=>'SDPR_TEST',numblks=>
1100,no_invalidate=>false);
```

```
-- _sdpr_pro 프로시져 실행.

SQL> exec sdpr_pro(p_start=>0,p_step=>1,p_stop=>1500);
Block count for SDPR: 0 blocks

SQL> @buffer
enter the object name :sdpr_test

OWNER   OBJECT_NAM   STATE  DIRTY  TEMP  PING  STALE  DIREC   BLOCKS
------ ------------ ------- ------ ----- ----- -----  ------ -------
SYS      SDPR_TEST   xcur      N     N     N     N     N            1

SQL> select statistic_name, value
  2  from v$segment_statistics
  3  where owner=user
  4  and object_name='SDPR_TEST'
  5  and statistic_name='physical reads direct';

STATISTIC_NAME                                                   VALUE
---------------------------------------------------- ----------
physical reads direct                                             1
```

테이블 통계정보의 blocks 를 small_table_threshold 보다 큰 1100 으로 하고, 테이블 크기가 0 부터 1 블록씩 증가하여 1500 블록이 될 때까지 sdpr_pro 프로시저를 수행하면 0 블록을 초과하는 시점인 실제 테이블 크기가 1 블록이상이 될 때 SDPR 이 최초로 발생한다. 통계정보의 blocks 가 _small_table_threshold 값보다 크므로 1 블록을 읽을 때도 SDPR 로 처리되는 것이다.

이와 반대로 테이블 통계정보의 blocks 가 _small_table_threshold 값보다 작으면 테이블이 아무리 크더라도 SDPR 은 발생하지 않는다. 따라서, 데이터의 변화량이 큰 테이블 일수록 현재 상태에 맞게 통계정보를 생성하는 것이 중요하다. 이번에는 버퍼캐시에 캐시 된 테이블의 블록 수와 더터 블록의 수가 변경될 경우 SDPR 은 어떻게 처리 되는지 살펴보자

```
-- 테이블 2000 블록 생성.

SQL> truncate table sdpr_test;
```

```
SQL> insert /*+ append */ into sdpr_test
  2  select rpad('*', 100, '*')
  3  from dual
  4  connect by level <= 2000;

2000 rows created.
```

-- 통계정보 생성

```
SQL> exec dbms_stats.gather_table_stats(ownname=>'SYS', tabname=>'SDPR_TEST',
method_opt=>'FOR ALL COLUMNS SIZE 1', no_invalidate=>false);

SQL> select owner,table_name,blocks,num_rows
  2  from dba_tables
  3  where table_name = 'SDPR_TEST';
OWNER      TABLE_NAME                     BLOCKS    NUM_ROWS
--------  ---------------------------   ---------- ---------
  SYS       SDPR_TEST                       2040        2000
```

-- 테이블 캐시 프로시져 수행.

```
SQL> alter session set "_small_table_threshold" = 1000 ;

SQL> exec cached_sdpr_pro(p_start=>300, p_step=>1);
Block count for impossible sdpr : 1990 blocks

SQL> @buffer
enter the object name :sdpr_test
OWNER   OBJECT_NAM   STATE  DIRTY  TEMP  PING  STALE  DIREC   BLOCKS
------------------- -------- ----- ------ ----- ------ ------  -------
  SYS    SDPR_TEST    xcur     N     N     N     N       N       2001
```

-- 더티블록 생성 프로시져 수행.

```
SQL> exec dirty_sdpr_pro(p_start=>50, p_step=>1, p_stop=>1500) ;
Block count for impossible sdpr : 990 blocks

SQL> @buffer
enter the object name :sdpr_test
```

```
OWNER   OBJECT_NAM   STATE  DIRTY  TEMP  PING  STALE  DIREC   BLOCKS
------------------ ------- ----- ------ ----- ------ ------  -------
SYS       SDPR_TEST  xcur     Y      N     N     N      N        990
SYS       SDPR_TEST  xcur     N      N     N     N      N       1011
```

SDPR_TEST 테이블에 블록 2000 개를 생성하고 통계정보를 수집하면 통계정보가 없을 때 와 약간 다른 결과를 보여준다. 통계정보가 없을 때는 버퍼캐시에 SDPR_TEST 테이블 블록이 50% 이상 캐시 되거나 더티 블록이 25% 이상 캐시 될 경우 SDPR 이 멈추었다. 물론 버퍼캐시 및 테이블의 크기에 따라 약간의 차이가 있지만 그래도 거의 비슷한 수준에서 멈추었다. 그러나, 통계정보를 참조하는 경우 테이블의 캐시 된 블록수가 테이블의 크기와 거의 비슷한 100%일 때 와 더티 블록이 50% 일 때 SDPR 이 멈춘다는 것이다. 이것은 통계정보가 존재하지 않을 때의 캐시 비율과 비교할 때 각각 2 배씩 증가한 것으로 보인다. 이때, 통계정보의 테이블 blocks 를 임의로 1100 으로 작게 한다면 어떻게 될까?

```
-- 테이블 통계정보의 blocks 를 1100 으로 낮춤.

EXEC DBMS_STATS.SET_TABLE_STATS (ownname=>'SYS', tabname=> 'SDPR_TEST',
numblks=>1100 , no_invalidate=>false) ;

SQL> select owner,table_name,blocks,num_rows
  2  from dba_tables
  3  where table_name = 'SDPR_TEST';

OWNER      TABLE_NAME         BLOCKS   NUM_ROWS
-------- -----------------  --------- ----------
SYS         SDPR_TEST           1100       2000

SQL> exec cached_sdpr_pro(p_start=>300, p_step=>1);
Block count for impossible sdpr : 1073 blocks

SQL>  exec dirty_sdpr_pro(p_start=>50, p_step=>1, p_stop=>1500) ;
Block count for impossible sdpr : 497 blocks
```

테스트결과, 실제 테이블의 크기는 2000 블록 이상이지만 통계정보의 blocks 를 1100 으로 낮추었더니 1070 개의 블록이 캐시 될 때 또는 497 개의 더티 블록이 캐시 될 때 멈추었다. 이것은 테이블 캐시여부를 실제 테이블의 크기가 아닌 통계정보의 blocks 를 기준으로 결정한다는 것을 알 수 있다. 테이블 통계정보의 blocks 가 1100 이므로 캐시 된 테이블 블록수가 통계정보

blocks 의 100%에 가까운 1070 블록, 더티 블록이 blocks 의 50%이 497 블록에서 SDPR 이 멈추었다. 통계정보가 없는 테이블을 풀 스캔 할 때는 커서를 실행할 때 마다 테이블의 실제 블록 수를 세그먼트 헤더블록을 통해 조회하였다. 그러나, 통계정보를 적용할 때는 tab$blkcnt, tabpart$.blkcnt, ind$.leafcnt 값을 커서의 어딘가에 저장하고 실행할 때 마다 저장된 값을 참조하여 SDPR 여부를 결정한다. 따라서, 커서가 invalid 되거나, 통계정보를 새로 생성하거나, 캐시 및 더티 블록 비율이 변할 경우 SDPR 여부도 런타임 시점에 언제든 바뀔 수 있다.

```
SQL> truncate table sdpr_test;

SQL> analyze table sdpr_test delete statistics ;

SQL> alter system flush shared_pool ;

-- 테이블 500 블록 생성.

SQL> insert /*+ append */ into sdpr_test
  2  select rpad('*', 100, '*')
  3  from dual
  4  connect by level <= 500;

500 rows created.

-- sdpr_test 테이블 통계정보 제거.

SQL> alter table sdpr_test delete statistics;

-- test-1 번 커서 실행.

SQL> alter session set "_small_table_threshold" = 1000 ;

SQL> select /*+ full(sdpr_test) test-1 */ count(*) from sdpr_test;

  COUNT(*)
----------
       500


-- v$segment_statistics 조회.
```

```
SQL> select statistic_name, value
  2  from v$segment_statistics
  3  where owner=user
  4  and object_name='SDPR_TEST'
  5  and statistic_name='physical reads direct';

STATISTIC_NAME                    VALUE
------------------------------ ----------
physical reads direct                   0

-- 테이블의 통계정보 blocks 를 1100 으로 설정 ( no_invalidate 옵션제거 )

SQL> EXEC DBMS_STATS.SET_TABLE_STATS(ownname=>'SYS',
tabname=>'SDPR_TEST',numblks=>1100);

-- 커서 test-1 재실행.

SQL> select /*+ full(sdpr_test) test-1 */ count(*) from sdpr_test;

-- v$segment_statistics 조회.

STATISTIC_NAME                      VALUE
-------------------------------- ---------
physical reads direct                    0

-- 커서 test-2 실행.

SQL> select /*+ full(sdpr_test) test-2 */ count(*) from sdpr_test;

  COUNT(*)
----------
      1000

-- v$segment_statistics 조회.

STATISTIC_NAME                 VALUE
--------------------------   --------
physical reads direct            2000
```

SDPR_TEST 테이블에 500 블록을 생성한 후 통계정보를 제거한 상태에서 테이블을 풀 스캔 하면 실제 블록수가 _small_table_threshold 보다 작으므로 SDPR 는 발생하지 않는다. 통계정

보의 blocks 를 1100 으로 설정하고 다시 동일한 SQL 을 실행할 경우 마찬가지로 SDPR 로 실행되지 않는다. 통계정보 blocks 가 1100 이므로 당연히 SDPR 로 처리 되야 하지만 그렇지 않은 이유는 통계정보를 no_invalidate 로 생성했기 때문이다. no_invalidate 옵션을 빼면 기본값이 auto 모드로 통계정보가 생성되어 일정시간이 지난 후에야 커서가 invalid 된다. 따라서, 먼저 수행된 test-1 번 커서는 재실행 했을 때도 커서가 아직 invalid 되지 않았으므로 SDPR 로 실행되지 않지만, 통계정보 생성 후 처음 실행되는 test-2 번 커서는 통계정보 blocks 를 적용하여 SDPR 로 실행된다. 따라서, test-2 커서는 invalid 되지 않거나, 버퍼캐시에 테이블 블록의 캐시 및 더티 블록 상태가 바뀌지 않는 이상 계속 SDPR 로 실행된다. [표-1]은 앞에서 설명한 serial direct read 여부를 결정하는 판별기준을 표로 정리한 것이다.

구분	SDPR 결정 기준	SDPR 멈춤 (블록 캐시)	SDPR 멈춤 (더티블록 캐시)
통계정보 있음	통계정보 blocks (tab$blkcnt, tabpart$.blkcnt, ind$.leafcnt)	통계정보 blocks 대비 100% 캐시	통계정보 blocks 대비 50% 캐시
통계정보 없음	세그먼트 실제 블록 수(세그먼트 헤더블록 조회)	세그먼트 블록 수 대비 50% 캐시	세그먼트 블록 수 대비 25% 캐시

[표 1] serial direct read 발생기준

결론

지금까지 SDPR(serial direct read)이 어떻게 동작하는지 몇 가지 사례를 통해 살펴보았다. 하지만, 경우에 따라 다양하고 복잡하게 느껴질 수도 있을 것이다. 관련된 파라미터를 변경하지 않고 통계정보만 적절히 생성하여 오라클이 알아서 수행하도록 놔두면 될 일이지만, 뭐든지 부족하거나 과하면 좋지 않듯이 불필요한 direct read 가 과하게 발생하거나, 반대로 그렇지 못할 경우 SDPR 의 동작방식을 알고 있다면 위에서 언급한 파라미터와 통계정보 등을 활용하여 적절한 튜닝도 가능할 것으로 생각한다.

참고 문헌

http://afatkulin.blogspot.co.uk , http://blog.tanelpoder.com , http://docs.oracle.com