About multiblock reads

Frits Hoogland - Trivadis Performance Days - 2015
$(whoami)$

- Frits Hoogland
  - Working with Oracle products since 1996

- Blog: http://fritshoogland.wordpress.com
- Twitter: @fritshoogland
- Email: frits.hoogland@enkitec.com
- Oracle ACE Director
- OakTable Member
Books

Author, together with Martin Bach, Karl Arao and Andy Colvin.

Technical reviewer:
Agenda

• The Oracle full segment scan implementation
  • Options for full segments scans <=V10 vs. V11

• ‘traditional full scans’ & direct path full scans

• ‘autotune’ / adaptive direct path reads

• Implementation details
What is this presentation about?

• Multiblock reads can behave different after 10.2

• This could lead to different behaviour of applications using the database.

• I assume the audience to have basic understanding about:
  • Oracle execution plans.
  • Oracle SQL/10046 extended traces.
  • General execution behaviour of the RDBMS engine.
  • C language in general.
Row source operations

• Multiblock reads are an optimised method to read database blocks from disk for a database process.

• Mainly used for the rowsource operations:
  • ‘TABLE ACCESS FULL’
  • ‘FAST FULL INDEX SCAN’
  • ‘BITMAP FULL SCAN’
Row source operations

- For much of other segment access rowsource actions, like:
  - ‘INDEX UNIQUE SCAN’
  - ‘INDEX RANGE SCAN’
  - ‘INDEX FULL SCAN’
  - ‘TABLE ACCESS BY INDEX ROWID’

- Use single block reads are mostly used.

- The order in which individual blocks are read is important.
db file multiblock read count

• MBRC is the maximum amount of blocks read in one IO call.

• Buffered MBRC cannot cross extent borders.

• Concepts guide on full table scans: (11.2 version)
  • A scan of table data in which the database sequentially reads all rows from a table and filters out those that do not meet the selection criteria. All data blocks under the high water mark are scanned.
db file multiblock read count

• Multiblock reads are done up to DB_FILE_MULTIBLOCK_READ_COUNT blocks.
  • If MBRC is unset, default is ‘maximum IO size that can be efficiently performed’.

• Most operating systems allow a single IO operation up to 1 MB.
  • I prefer to set it manually.
My test environment

- Mac OSX Yosemite, VMware fusion 7.1.2.
  - VM: OL6u7 x86_64
    - Database 11.2.0.1/2/3/4 and 12.1.0.1/2
    - ASM GI 12.1.0.2

- Sample tables:
  - T1 - 21504 blocks - 176M - 1’000’000 rows
    - PK index - 2304 blocks / 19M
  - T2 - 21504 blocks - 176M - 1’000’000 rows
First test

• 10.2.0.1 instance:
  • sga_target = 600M
  • Effective buffercache size = 450M
  • Freshly started
First test

```
TS@v10201 > select /*+ index(t t1_pk_ix) */ count(id), sum(scattered) from t1 t;
```

<table>
<thead>
<tr>
<th>COUNT(ID)</th>
<th>SUM(SCATTERED)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>9999500000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>5</td>
<td>23234</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>T1</td>
<td>1000K</td>
<td>4884K</td>
<td>23234</td>
</tr>
<tr>
<td>3</td>
<td>INDEX FULL SCAN</td>
<td>T1_PK_IX</td>
<td>1000K</td>
<td></td>
<td>2253</td>
</tr>
</tbody>
</table>
First test

• How would you expect Oracle 10.2.0.1 to execute this?

• In other words:
• What would be the result of a SQL trace with waits? *

* If all blocks need to be read from disk (ie. not cached)
First test

• My guess would be:

• Index root block (1 block)
• None, one or more branch blocks (1 block)
• Index leaf block, fetch values (1 block)
• Table block via index rowid, fetch value(s) (1/1+ block)
• Index values, block value(s), etc.
First test

• That should look like something like this:

```
WAIT #8: nam='db file sequential read' ela= 326 file#=5 block#=43028 blocks=1
WAIT #8: nam='db file sequential read' ela= 197 file#=5 block#=43719 blocks=1
WAIT #8: nam='db file sequential read' ela= 227 file#=5 block#=43029 blocks=1
WAIT #8: nam='db file sequential read' ela= 125 file#=5 block#=20 blocks=1
WAIT #8: nam='db file sequential read' ela= 109 file#=5 block#=21 blocks=1
WAIT #8: nam='db file sequential read' ela= 242 file#=5 block#=22 blocks=1
WAIT #8: nam='db file sequential read' ela= 98 file#=5 block#=23 blocks=1
WAIT #8: nam='db file sequential read' ela= 76 file#=5 block#=24 blocks=1
WAIT #8: nam='db file sequential read' ela= 77 file#=5 block#=25 blocks=1
WAIT #8: nam='db file sequential read' ela= 77 file#=5 block#=26 blocks=1
WAIT #8: nam='db file sequential read' ela= 105 file#=5 block#=27 blocks=1
WAIT #8: nam='db file sequential read' ela= 82 file#=5 block#=28 blocks=1
WAIT #8: nam='db file sequential read' ela= 71 file#=5 block#=29 blocks=1
WAIT #8: nam='db file sequential read' ela= 93 file#=5 block#=43030 blocks=1
...```

15
First test

• Instead, I get:

WAIT #4: nam='db file scattered read' ela= 361 file#=5 block#=43025 blocks=8
WAIT #4: nam='db file scattered read' ela= 220 file#=5 block#=43713 blocks=8
WAIT #4: nam='db file scattered read' ela= 205 file#=5 block#=17 blocks=8
WAIT #4: nam='db file scattered read' ela= 219 file#=5 block#=25 blocks=8
WAIT #4: nam='db file scattered read' ela= 192 file#=5 block#=33 blocks=8
WAIT #4: nam='db file scattered read' ela= 141 file#=5 block#=41 blocks=8
WAIT #4: nam='db file scattered read' ela= 123 file#=5 block#=49 blocks=8
WAIT #4: nam='db file scattered read' ela= 190 file#=5 block#=57 blocks=8
WAIT #4: nam='db file scattered read' ela= 231 file#=5 block#=43033 blocks=8
WAIT #4: nam='db file scattered read' ela= 113 file#=5 block#=65 blocks=8
...

First test

• Sets of 8 blocks are read for rowsources which really need a single block.

• Reason:
  • This is an empty cache.
  • Oracle reads multiple blocks to get the cache filled.
  • ‘cache warming’
    • Statistic (‘physical reads cache prefetch’)

• Needed to tune the BC down to 50M and pre-warm it with another table to get single block reads again (!!)
• Or set “_db_cache_pre_warm” to false
Full scan - Oracle 10.2

• Let’s look at an Oracle 10.2.0.1 database

• SGA_TARGET 600M

• Table TS.T2 size 21504 blks / 176M
Full scan - Oracle 10.2

TS@v10201 > set autot on exp stat
TS@v10201 > select count(*) from t2;

    COUNT(*)
    -------
       1000000

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>3674 (1)</td>
<td>00:00:45</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>T2</td>
<td>1007K</td>
<td>3674 (1)</td>
<td>00:00:45</td>
</tr>
</tbody>
</table>
Statistics

212 recursive calls
0 db block gets
20976 consistent gets
20942 physical reads
0 redo size
515 bytes sent via SQL*Net to client
469 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
4 sorts (memory)
0 sorts (disk)
1 rows processed
SYS@v10201 AS SYSDBA>
select object_id, object_name, owner from dba_objects where object_name = 'T2';

<table>
<thead>
<tr>
<th>OBJECT_ID</th>
<th>OBJECT_NAME</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>10237</td>
<td>T2</td>
<td>TS</td>
</tr>
</tbody>
</table>

SYS@v10201 AS SYSDBA> select * from x$kcboqh where obj# = 10237;

<table>
<thead>
<tr>
<th>ADDR</th>
<th>INDX</th>
<th>INST_ID</th>
<th>TS#</th>
<th>OBJ#</th>
<th>NUM_BUF</th>
<th>HEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFD7FFD5C6FA8</td>
<td>335</td>
<td>1</td>
<td>5</td>
<td>10237</td>
<td>20942</td>
<td>000000038FBCF840</td>
</tr>
</tbody>
</table>
Full scan - Oracle 10.2

TS@v10201 > select count(*) from t2;

Statistics

----------------------------------------------------------
0  recursive calls
0  db block gets
20953  consistent gets
0  physical reads
0  redo size
515  bytes sent via SQL*Net to client
469  bytes received via SQL*Net from client
2  SQL*Net roundtrips to/from client
0  sorts (memory)
0  sorts (disk)
1  rows processed
Full scan - Oracle 11.2

- Now look at an Oracle 11.2.0.3 database

  - SGA_TARGET 600M

  - Table TS.T2 size 21504 blks / 176M
Full scan - Oracle 11.2

TS@v11203 > select count(*) from t2;

COUNT(*)
---------
1000000

Execution Plan

Plan hash value: 3724264953

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>3672 (1)</td>
<td>00:00:45</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>T2</td>
<td>1000K</td>
<td>3672 (1)</td>
<td>00:00:45</td>
</tr>
</tbody>
</table>
Full scan - Oracle 11.2

Statistics

----------------------------------------------------------
217  recursive calls
  0  db block gets
20970  consistent gets
20942  physical reads
  0  redo size
526  bytes sent via SQL*Net to client
523  bytes received via SQL*Net from client
  2 SQL*Net roundtrips to/from client
  4 sorts (memory)
  0 sorts (disk)
  1 rows processed
SYS@v11203 AS SYSDBA>
select object_id, object_name, owner from dba_objects where object_name = 'T2';

<table>
<thead>
<tr>
<th>OBJECT_ID</th>
<th>OBJECT_NAME</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>66614</td>
<td>T2</td>
<td>TS</td>
</tr>
</tbody>
</table>

SYS@v11203 AS SYSDBA> select * from x$kcboqh where obj# = 66614;

<table>
<thead>
<tr>
<th>ADDR</th>
<th>INDX</th>
<th>INST_ID</th>
<th>TS#</th>
<th>OBJ#</th>
<th>NUM_BUF</th>
<th>HEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFD7FFC541B18</td>
<td>43</td>
<td>1</td>
<td>5</td>
<td>66614</td>
<td>1</td>
<td>000000039043E470</td>
</tr>
</tbody>
</table>
Full scan - Oracle 11.2

```sql
TS@v11203 > select count(*) from t2;
```

Statistics
----------------------------------------------------------
  0  recursive calls
  0  db block gets
 20945 consistent gets
 20941 physical reads
  0  redo size
  526  bytes sent via SQL*Net to client
  523  bytes received via SQL*Net from client
  2  SQL*Net roundtrips to/from client
  0  sorts (memory)
  0  sorts (disk)
  1  rows processed
Full scan 10.2 vs. 11.2

- Why does version 10 caches all the blocks read,
- And version 11 only 1 of them??

- Let’s do an extended SQL trace
  - AKA 10046 level 8 trace.
Full scan 10.2 vs. 11.2

- Relevant part of 10046/8 trace file of version 10.2.0.1:

  WAIT #1: nam='db file sequential read' ela= 32941 file#=5 block#=19 blocks=1
  WAIT #1: nam='db file scattered read' ela= 4003 file#=5 block#=20 blocks=5
  WAIT #1: nam='db file scattered read' ela= 6048 file#=5 block#=25 blocks=8
  WAIT #1: nam='db file scattered read' ela= 1155 file#=5 block#=34 blocks=7
  WAIT #1: nam='db file scattered read' ela= 860 file#=5 block#=41 blocks=8
  WAIT #1: nam='db file scattered read' ela= 837 file#=5 block#=50 blocks=7
  WAIT #1: nam='db file scattered read' ela= 1009 file#=5 block#=57 blocks=8
  WAIT #1: nam='db file scattered read' ela= 890 file#=5 block#=66 blocks=7
  WAIT #1: nam='db file scattered read' ela= 837 file#=5 block#=73 blocks=8
  WAIT #1: nam='db file scattered read' ela= 10461 file#=5 block#=82 blocks=7
  WAIT #1: nam='db file scattered read' ela= 623 file#=5 block#=89 blocks=8
  WAIT #1: nam='db file scattered read' ela= 1077 file#=5 block#=98 blocks=7
  WAIT #1: nam='db file scattered read' ela= 49146 file#=5 block#=105 blocks=8
  WAIT #1: nam='db file scattered read' ela= 719 file#=5 block#=114 blocks=7
  WAIT #1: nam='db file scattered read' ela= 1093 file#=5 block#=121 blocks=8
Full scan 10.2 vs. 11.2

- Relevant part of 10046/8 trace file of version 11.2.0.3:

```plaintext
WAIT #140120507194664:
  nam='db file sequential read' ela= 12607 file#=5 block#=43394 blocks=1
  obj#=14033 tim=1329685383169372

  nam='direct path read' ela= 50599 file number=5 first dba=43395 block cnt=13
  nam='direct path read' ela= 21483 file number=5 first dba=43425 block cnt=15
  nam='direct path read' ela= 10766 file number=5 first dba=43441 block cnt=15
  nam='direct path read' ela= 12915 file number=5 first dba=43457 block cnt=15
  nam='direct path read' ela= 12583 file number=5 first dba=43473 block cnt=15
  nam='direct path read' ela= 11899 file number=5 first dba=43489 block cnt=15
  nam='direct path read' ela= 10010 file number=5 first dba=43505 block cnt=15
  nam='direct path read' ela= 160237 file number=5 first dba=43522 block cnt=126
  nam='direct path read' ela= 25561 file number=5 first dba=43650 block cnt=126
  nam='direct path read' ela= 121507 file number=5 first dba=43778 block cnt=126
  nam='direct path read' ela= 25253 file number=5 first dba=43906 block cnt=126
```
First single block read

- The segment header is read separately
- Single block, read into SGA

- The header block is listed in `dba_segments`

```
select owner, segment_name, header_file, header_block
from dba_segments where segment_name like 'T2';
```

<table>
<thead>
<tr>
<th>OWNER</th>
<th>SEGMENT_NAME</th>
<th>HEADER_FILE</th>
<th>HEADER_BLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS</td>
<td>T2</td>
<td>5</td>
<td>130</td>
</tr>
</tbody>
</table>
Full scan 10.2 vs. 11.2

• A full scan uses direct path reads in the v11 case.

• Noticeable by ‘direct path read’ event

• Direct path reads go to PGA
• Which means the blocks read are not cached
Full scan 10.2 vs. 11.2 vs. 12.1

• Do all full scans in version 11 always use direct path?
  • Direct path reads are considered
    • #blk > 5*_stt (11.2.0.x ?)
    • #blk > _stt (12.1.0.2, pbly already with 11203)

• PS: MOS note 787373.1
  • “How does Oracle load data into the buffer cache for table scans?”
  • Mentions _small_table_threshold being the limit
    • Note INCORRECT!
Direct path read

- Small table threshold of my Oracle 11 instance:

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>_small_table_threshold</td>
<td>245</td>
</tr>
</tbody>
</table>
Direct path read

• This means objects up to $245 \times 5 = 1225$ blocks will be read into buffercache / SGA.

• Let’s create a table with a size just below 1225 blocks:

  ```sql
  TS@v11203 > create table t1_small as select * from t1 where id <= 47000;
  TS@v11203 > exec dbms_stats.gather_table_stats(null,'T1_SMALL');
  ```
Direct path read

SYS@v11203 AS SYSDBA>

select segment_name, blocks, bytes
from dba_segments where segment_name = 'T1_SMALL';

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>BLOCKS</th>
<th>BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1_SMALL</td>
<td>1024</td>
<td>8388608</td>
</tr>
</tbody>
</table>

SQL@v11203 AS SYSDBA> alter system flush buffer_cache;
Direct path read

TS@v11203 > set autot trace exp stat
TS@v11203 > select count(*) from t1_small;

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>176 (1)</td>
<td>00:00:03</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>T1_SMALL</td>
<td>47000</td>
<td>176 (1)</td>
<td>00:00:03</td>
</tr>
</tbody>
</table>
Direct path read

Statistics

- 0 recursive calls
- 0 db block gets
- 983 consistent gets
- 979 physical reads
- 0 redo size
- 527 bytes sent via SQL*Net to client
- 523 bytes received via SQL*Net from client
- 2 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 1 rows processed
**Direct path read**

```sql
SYS@v11203 AS SYSDBA>
select object_id, object_name, owner
from dba_objects where object_name = 'T1_SMALL';

<table>
<thead>
<tr>
<th>OBJECT_ID</th>
<th>OBJECT_NAME</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>66729</td>
<td>T1_SMALL</td>
<td>TS</td>
</tr>
</tbody>
</table>

SYS@v11203 AS SYSDBA> select * from x$kcboqh where obj# = 66729;

<table>
<thead>
<tr>
<th>ADDR</th>
<th>INDX</th>
<th>INST_ID</th>
<th>TS#</th>
<th>OBJ#</th>
<th>NUM_BUF</th>
<th>HEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFD7FFC6E1EF0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>66729</td>
<td>979</td>
<td>0000000390437840</td>
</tr>
</tbody>
</table>
```
Direct path read

• Ah, now the full scan is buffered!
• Another scan will reuse the cached blocks now:

```
TS@v11203 > select count(*) from t1_small;
...

Statistics
----------------------------------------------------------
  0  recursive calls
  0  db block gets
 983  consistent gets
  0  physical reads
```
Direct path read

• What type of wait event will be used for a full scan:

• Oracle version 11.2
• If size is smaller than 5* _small_table_threshold?
  • Or _small_table_threshold with version 12
Direct path read

Well, try it:

```
TS@v11203 > alter session set events '10046 trace name context forever, level 8';
TS@v11203 > select count(*) from t1_small;
...
TS@v11203 > alter session set events '10046 trace name context off';
```

It shows:

```
WAIT #140358956326184:
   nam='db file sequential read' elas=38476 file#=5 block#=88706 blocks=1
   obj#=14047 tim=133036998567263

   nam='db file scattered read' elas=116037 file#=5 block#=88707 blocks=5
   nam='db file scattered read' elas=56675 file#=5 block#=88712 blocks=8
   nam='db file scattered read' elas=11195 file#=5 block#=88721 blocks=7
   nam='db file scattered read' elas=132928 file#=5 block#=88728 blocks=8
   nam='db file scattered read' elas=18692 file#=5 block#=88737 blocks=7
   nam='db file scattered read' elas=87817 file#=5 block#=88744 blocks=8
```
Oracle 11 multiblock IO

• In version 11 of the Oracle database

• Multiblocks reads use both wait events:
  • db file scattered read
  • direct path read

• Which are two different codepath’s
Implementation

• Buffered multiblock reads

  • Buffered multiblock reads == ‘db file scattered read’
  • Up to version 10 the ONLY\* option for non-PQ multiblock reads
  • Starting from version 11, a possible multiblock read option
**Buffered multiblock reads**

```sql
SYS@v10201 AS SYSDBA>
select segment_name, extent_id, block_id, blocks, bytes
from dba_extents
where segment_name = 'T2' and owner = 'TS' order by extent_id;
```

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>EXTENT_ID</th>
<th>BLOCKS</th>
<th>BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>0</td>
<td>8</td>
<td>65536</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>15</td>
<td>8</td>
<td>65536</td>
</tr>
<tr>
<td>T2</td>
<td>16</td>
<td>128</td>
<td>1048576</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>78</td>
<td>128</td>
<td>1048576</td>
</tr>
<tr>
<td>T2</td>
<td>79</td>
<td>1024</td>
<td>8388608</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>91</td>
<td>1024</td>
<td>8388608</td>
</tr>
</tbody>
</table>
Buffered multiblock reads

Version 10 multiblock reads:

WAIT #2: nam='db file sequential read' ela= 12292 file#=5 block#=19 blocks=1
WAIT #2: nam='db file scattered read' ela= 179162 file#=5 block#=20 blocks=5
WAIT #2: nam='db file scattered read' ela= 47597 file#=5 block#=25 blocks=8
WAIT #2: nam='db file scattered read' ela= 5206 file#=5 block#=34 blocks=7
WAIT #2: nam='db file scattered read' ela= 94101 file#=5 block#=41 blocks=8
WAIT #2: nam='db file scattered read' ela= 512 file#=5 block#=50 blocks=7
WAIT #2: nam='db file scattered read' ela= 87657 file#=5 block#=57 blocks=8
WAIT #2: nam='db file scattered read' ela= 27488 file#=5 block#=66 blocks=7
WAIT #2: nam='db file scattered read' ela= 24316 file#=5 block#=73 blocks=8
WAIT #2: nam='db file scattered read' ela= 55251 file#=5 block#=82 blocks=7
WAIT #2: nam='db file scattered read' ela= 641 file#=5 block#=89 blocks=8
WAIT #2: nam='db file scattered read' ela= 455 file#=5 block#=98 blocks=7
WAIT #2: nam='db file scattered read' ela= 43826 file#=5 block#=105 blocks=8
WAIT #2: nam='db file scattered read' ela= 32685 file#=5 block#=114 blocks=7
WAIT #2: nam='db file scattered read' ela= 60212 file#=5 block#=121 blocks=8
WAIT #2: nam='db file scattered read' ela= 37735 file#=5 block#=130 blocks=7
WAIT #2: nam='db file scattered read' ela= 59565 file#=5 block#=137 blocks=8

(ps: edited for clarity)
name='db file scattered read' ela= 87657 file#=5 block#=147 blocks=126
Non buffered multiblock reads

WAIT #140120507194664:
nam='db file sequential read' ela= 12607 file#=5 block#=43394 blocks=1 obj#=14033
tim=1329685383169372

nam='direct path read' ela= 50599 file number=5 first dba=43395 block cnt=13
nam='direct path read' ela= 21483 file number=5 first dba=43425 block cnt=15
nam='direct path read' ela= 10766 file number=5 first dba=43441 block cnt=15
nam='direct path read' ela= 12915 file number=5 first dba=43457 block cnt=15
nam='direct path read' ela= 12583 file number=5 first dba=43473 block cnt=15
nam='direct path read' ela= 11899 file number=5 first dba=43489 block cnt=15
nam='direct path read' ela= 10010 file number=5 first dba=43505 block cnt=15
nam='direct path read' ela= 160237 file number=5 first dba=43522 block cnt=126
nam='direct path read' ela= 25561 file number=5 first dba=43650 block cnt=126
nam='direct path read' ela= 121507 file number=5 first dba=43778 block cnt=126
nam='direct path read' ela= 25253 file number=5 first dba=43906 block cnt=126
nam='direct path read' ela= 21483 file number=5 first dba=43425 block cnt=15
not in tracefile. more on this later.
ASSM

• Automatic segment space management
  • Tablespace property
  • Default since Oracle 10.2

• Uses L 1/2/3 bitmap blocks for space management
  • 8 blocks: 1 BMB as first block of every other extent
  • 128 blocks: 2 BMB as first blocks in all extents
  • 1024 blocks: 4 BMB as first blocks in all extents
Multiblock implementation

• Conclusion:
  • Buffered reads scan up to:
    • Non data (space admin. bitmap) block
    • Extent border
    • Block already in cache (from TOP)

• Direct path/non buffered reads scan up to:
  • Non data (space admin. bitmap) block
  • Block already in cache (from TOP)
Waits and implementation

• ‘Wait’ or wait event
  • Part of the formula:
    • Elapsed time = CPU time + Wait time

• Inside the Oracle database it is meant to record the time spent in a specific part of the oracle database code where Oracle suspects stalling/waiting.

• Let’s look at the implementation of some of the wait events for multiblock reads!
Waits and implementation

• Quick primer on read IO System calls on Linux:
  
  • Synchronous
    • pread64()
  
  • Asynchronous
    • io_submit()
    • io_getevents()
strace

• Linux tool for tracing (viewing) system calls
  • Solaris/AIX: truss, HPUX: tusc.

• Very, very, useful to understand what is happening
• Much people are using it for years
• STRACE LIES! (at least on linux)
strace lies

• strace doesn’t show io_getevents() if:
  • timeout struct set to {0,0} (‘zero’)
  • does not succeed in reaping any IO’s

• This strace omission is not documented!
strace lies

• This is best seen with system’s IO capability severely throttled (1 IOPS)

• See http://fritshoogland.wordpress.com/2012/12/15/throttling-io-with-linux/

• Cgroups
  • Control groups
  • Linux feature
  • Fully available with OL6
strace lies

• Strace output
  • Version 11.2.0.3
  • IO throttled to 1 IOPS
  • Full table scan doing ‘count(*) on t2’
  • With 10046 at level 8
    • To show where waits are occurring
  
• Start of FTS, up to first reap of IO
strace lies

```c
io_submit(139801394388992, 1, {{0x7f260a8b3450, 0, 0, 0, 257}}) = 1

io_submit(139801394388992, 1, {{0x7f260a8b31f8, 0, 0, 0, 257}}) = 1

io_getevents(139801394388992, 1, 128, {{0x7f260a8b3450, 0x7f260a8b3450, 106496, 0}}, {600, 0}) = 1

write(8, "WAIT #139801362351208: nam='dire"..., 133) = 133

* edited for clarity
```
strace lies

• Profile the same using ‘gdb’
• Set breakpoints at functions:
  • io_submit, io_getevents_0_4
  • kslwtbctx, kslwtectx
• Let gdb continue after breakpoint
• The symbol table is preserved in the oracle binary
• Making it able to set breakpoints at functions
strace lies

When not successful (no IOs reaped), these calls do not show up with strace.

#0  io_submit (ctx=0x7f46fe708000, nr=1, iocbs=0x7fff24547ce0) at io_submit.c:23
#0  io_submit (ctx=0x7f46fe708000, nr=1, iocbs=0x7fff24547ce0) at io_submit.c:23

Breakpoint 3, io_getevents_0_4 (ctx=0x7f46fe708000, min_nr=2, nr=128, events=0x7fff24550348, timeout=0x7fff24551350) at io_getevents.c:46
Breakpoint 3, io_getevents_0_4 (ctx=0x7f46fe708000, min_nr=2, nr=128, events=0x7fff24553428, timeout=0x7fff24554430) at io_getevents.c:46
Breakpoint 3, io_getevents_0_4 (ctx=0x7f46fe708000, min_nr=2, nr=128, events=0x7fff24550148, timeout=0x7fff24551150) at io_getevents.c:46
Breakpoint 3, io_getevents_0_4 (ctx=0x7f46fe708000, min_nr=2, nr=128, events=0x7fff24553228, timeout=0x7fff24554230) at io_getevents.c:46

#0  0x00000000008f9a652 in kslwtbctx ()
Breakpoint 3, io_getevents_0_4 (ctx=0x7f46fe708000, min_nr=1, nr=128, events=0x7fff24550138, timeout=0x7fff24551140) at io_getevents.c:46
#0  0x00000000008fa1334 in kslwtectx ()

* edited for clarity
strace does not lie after all..

- The Linux AIO library source code can be viewed:
  - https://git.fedorahosted.org/cgit/libaio.git/

- This is the source code for the `io_getevents_0_4` function, inside `io_getevents.c`:
strace does not lie after all..

```c
int io_getevents_0_4(io_context_t ctx, long min_nr, long nr, struct io_event * events, struct timespec * timeout)
{
    struct aio_ring *ring;
    ring = (struct aio_ring*)ctx;
    if (ring==NULL || ring->magic != AIO_RING_MAGIC)
        goto do_syscall;
    if (timeout!=NULL && timeout->tv_sec == 0 && timeout->tv_nsec == 0) {
        if (ring->head == ring->tail)
            return 0;
    }
    do_syscall:
    return __io_getevents_0_4(ctx, min_nr, nr, events, timeout);
}
```

If the timeout struct is set to \{0,0\}

And head == tail, in other words: no IOs are ready yet, return. **Before the io_getevents system call is issued.**
strace does not lie after all..

• The io_getevents_0_4 function takes advantage from the fact that the IO results are visible in userspace.
• In other words: the kernel places IO results in the ring buffer once they return.
• This is a “trick” to avoid the overhead of doing a system call.
• It makes it less easy to understand.
**db file scattered read**

**Basic principle**

- File # and # blocks are determined
- Elapsed time of 'db file scattered read'
- Read ready, blocks available
- Read call of # bytes

**Time**
db file scattered read

synchronous IO
ASM & filesystem
11.2.0.1-12.1.0.2

time

pread64(fd, buf, #bytes, offset)

darker quare means ‘optional’
db file scattered read

asynchronous IO
filesystem
11.2.0.1-12.1.0.2

pread64(fd, buf, #bytes, offset)
db file scattered read

asynchronous IO
ASM
11.2.0.1

pread64(fd, buf, #bytes, offset)

io_submit(aio_ctx, #cb, {iocb})

io_getevents(aio_ctx, min_nr, nr, io_event, {600,0})
db file scattered read

asynchronous IO
ASM
11.2.0.2-12.1.0.1

time

pread64(fd, buf, #bytes, offset)

io_submit(aio_ctx, #cb, {iocb})

io_getevents(aio_ctx, min_nr, nr, io_event, {600,0})

1 I/O

>1 I/O
db file scattered read

asynchronous IO
ASM
12.1.0.2

1 I/O

time

pread64(fd, buf, #bytes, offset)

>1 I/O

time

io_submit(aio_ctx, #cb, {iocb})
io_getevents(aio_ctx, min_nr, nr, io_event, {600,0})
direct path read - 11g

• Time spent on waiting for reading blocks to put them into the PGA.

• Reports wait time of the request that gets reaped with a blocking io_getevents() call only.

• Multiple IO requests can be submitted with AIO.

• At start, Oracle tries to keep 2 IO’s in flight.

• Wait time is only reported if ‘waiting’ occurs:
  • Waiting means: an IO can not be reaped immediately after submitting.
direct path read

io_submit(aio_ctx, #cb, {iocb})

io_getevents(aio_ctx, min_nr, nr, io_event, {0,0})

io_getevents(aio_ctx, min_nr, nr, io_event, {600,0})
direct path read

synchronous IO
ASM
10.2.0.1-12.1.0.1

file # and # blocks are determined

ela time of ‘direct path read’
read ready, blocks available

time

pread64(fd, buf, #bytes, offset)

IMPORTANT: synchronous filesystem based ‘direct path read’ waits are timed correctly!
**direct path read**

Implementation
synchronous IO; ASM & filesystem
12.1.0.2

file # and # blocks are
determined

ela time of ‘direct path read’

read ready, blocks
available

pread64(fd, buf, #bytes, offset)
IO Slots

Discussion with Kerry Osborne about IO’s on Exadata
Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options

SQL> alter session set cell_offload_processing=false;
Session altered.

(reverse-i-search)`cou`: select count(*) from cg_var;
Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options
[oracle@dm01db01 ~]$ sqlplus cg/cg

SQL*Plus: Release 11.2.0.2.0 Production on Tue Nov 22 23:45:33 2011
Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options

SQL> alter session set cell_offload_processing=false;
Session altered.
(reverse-i-search)``
IO Slots

• Jonathan Lewis pointed me to ‘total number of slots’
  • v$sysstat
  • v$sesstat

• Global or per session number of slots

• ‘Slots are a unit of I/O and this factor controls the number of outstanding I/Os’
  • Comment with event 10353
IO Slots

• ‘total number of slots’

• Is NOT cumulative!

• So you won’t capture this statistic when taking delta’s from v$sysstat/v$sesstat!
IO Slots

• Let’s look at the throughput statistics again

• But together with number of slots
1. oracle@dm01db01.jdf.prod:/home/oracle ORACLE_SID=t11201 ORACLE_HOME=/u01/app/oracle/product/11.2.0/dbh...

oracle@sol10.local:/expor... oracle@sol10.local:/expor... oracle@dm01db01.jdf.pro... java oracle@dm01db01.jdf.pro...

[oracle@dm01db01 [t11201] ~]$ rsqlplus cg/cg

SQL*Plus: Release 11.2.0.1.0 Production on Wed Nov 23 22:52:45 2011

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP, Data Mining and Real Application Testing options
(reverse-i-search)```: alter session set cell_offload_processing=false;
Copyright (c) 1982, 2010, Oracle. All rights reserved.

ERROR:
ORA-28002: the password will expire within 7 days

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options

(reverse-i-search)`

```
IO Slots

• IO Slots are a mechanism to take advantage of storage bandwidth using AIO.

• With version 11+ direct path reads can be used by both PQ slaves as well as non PQ foregrounds.

• IO Slots are not used with buffered reads (!).

• Each outstanding asynchronous IO request is tracked using what is called a ‘slot’.

• Default and minimal number of slots: 2.
‘autotune’

- Mechanism in the direct path code with version 11 and 12.

- The database foreground measures direct path I/O effectiveness.
  - It measures time, wait time and throughput.
- The oracle process has the ability to add more asynchronous I/O slots.
- Only does so starting from 11.2.0.2.
  - Although the mechanism is there in 11.2.0.1.
‘autotune’

• Introducing event 10365:
  • “turn on debug information for adaptive direct reads”

• Set to 1 to get debug information:
  • `alter session set events ‘10365 trace name context forever, level 1’`
`autotune`

kcbldrssini: Timestamp 61180 ms
kcbldrssini: Current idx 16
kcbldrssini: Initializing kcbldrps
kcbldrssini: Slave idx 17
kcbldrssini: Number slots 2
kcbldrssini: Number of slots per session 2

kcblsinc: Timing time 1693472, wait time 1291416, ratio 76 st 248752270 cur 250445744
kcblsinc: Timing curidx 17 session idx 17
kcblsinc: Timestamp 64180 ms
kcblsinc: Current idx 17
kcblsinc: Slave idx 17
kcblsinc: Number slots 2
kcblsinc: Number of slots per session 2
kcblsinc: Previous throughput 8378 state 2
kcblsinc: adaptive direct read mode 1, adaptive direct write mode 0
‘autotune’

*** 2011-11-28 22:58:54.988
kcb1sinc:Timing time 2962717, wait time 2923226, ratio 98 st 253662983 cur 256625702
kcb1sinc: Timing curidx 19 session idx 19
kcb1sinc: Timestamp 70270 ms
kcb1sinc: Current idx 19
kcb1sinc: Slave idx 19
kcb1sinc: Number slots 2
kcb1sinc: Number of slots per session 2
kcb1sinc: Previous throughput 11210 state 1
kcb1sinc: adaptive direct read mode 1, adaptive direct write mode 0
kcb1sinc: Adding extra slos 1

*** 2011-11-28 22:58:58.999
kcb1sinc:Timing time 4011239, wait time 3528563, ratio 87 st 256625785 cur 260637026
kcb1sinc: Timing curidx 20 session idx 20
kcb1sinc: Timestamp 74170 ms
kcb1sinc: Current idx 20
kcb1sinc: Slave idx 20
kcb1sinc: Number slots 3
kcb1sinc: Number of slots per session 3
kcb1sinc: Previous throughput 12299 state 2
kcb1sinc: adaptive direct read mode 1, adaptive direct write mode 0
‘autotune’

• Looking at the 10365 trace, the reason 11.2.0.1 does not ‘autotune’ could be guessed....
'autotune'

kcblsinc: Timing time 3092929, wait time 0, ratio 0 st 4271872759 cur 4274965690
kcblsinc: Timing curidx 65 session idx 65
kcblsinc: Timestamp 192430 ms
kcblsinc: Current idx 65
kcblsinc: Slave idx 65
kcblsinc: Number slots 2
kcblsinc: Number of slots per session 2
kcblsinc: Previous throughput 20655 state 2
kcblsinc: adaptive direct read mode 1, adaptive direct write mode 0

kcblsinc: Timing time 2944852, wait time 0, ratio 0 st 4274965762 cur 4277910616
kcblsinc: Timing curidx 66 session idx 66
kcblsinc: Timestamp 195430 ms
kcblsinc: Current idx 66
kcblsinc: Slave idx 66
kcblsinc: Number slots 2
kcblsinc: Number of slots per session 2
kcblsinc: Previous throughput 20746 state 1
kcblsinc: adaptive direct read mode 1, adaptive direct write mode 0
IO slots

11.2.0.3
FAST IO

# slots starts with 2

io_submit(aio_ctx, 1, {iocb})

kcbsinc()

io_submit(aio_ctx, 1, {iocb})

kcbgtcr()}

io_submit(aio_ctx, 1, {iocb})

io_getevents(aio_ctx, 2, 128, io_event, {0, 0}) OK!

kcbgtcr()}

io_getevents(aio_ctx, 2, 128, io_event, {0, 0}) OK!

kcbgtcr()}

io_getevents(aio_ctx, 2, 128, io_event, {0, 0}) OK!

io_getevents(aio_ctx, 2, 128, io_event, {0, 0}) OK!
IO slots

11.2.0.3
FAST IO

```
kcblsinc ()
+1 'slos'
```

```
io_submit(aio_ctx,1 , {iocb})
```

```
io_getevents(aio_ctx, 3, 128, io_event, {0, 0})  OK!
```
Time and waits

• Waits implementation:
  • Most are system call instrumentation;
  • db file sequential read
  • ‘direct path read’ is different.
  • Only shows up if no IOs can be reaped immediately.
  • The wait only occurs if process is truly waiting.
  • With AIO, a process has the ability to keep on processing without waiting on IO.
• Wait time *IS NOT* physical IO latency
Super huge IOs

• Christian Antognini showed me an AWR report
  • With an experiment of setting MBRC to 4000
  • Which lead to IOs / stats of IOs > 1MB

• This conflicts with the official description about MBRC in the Oracle concepts manual
Super huge IOs

• Traditional multiblock reads
• “db file scattered read” event
• Indeed limited to 1 MB / block size
Super huge IOs

- “New” multiblock read: “direct path read” event.
- Tested with 12.1.0.2 (available with 11.2 too)
- Testing shows multiblock reads up to 4096 blocks.
- Limited by BMB and MBRC.

- ASM:
  - IO size up to allocation unit size.
  - Exceeding AU means multiple requests.

- Filesystem:
  - IO size up to 4096 blocks: 32MB.
Super huge IOs

WAIT!

Isn’t the maximum physical (OS) limit 1MB on linux?

(and most other platforms)

Did Oracle magically overcome this limitation?
Super huge IOs

- Investigate!
- On the Oracle level:
  - *Throttle* IOPS to get waits
  - Set MBRC=10000 & sql_trace level8

```plaintext
WAIT #140567216695864: nam='direct path read' ela= 33964979 file number=4 first
dba=214920 block cnt=2168 obj#=20535 tim=16108554608
WAIT #140567216695864: nam='direct path read' ela= 73087338 file number=4 first
dba=217088 block cnt=4096 obj#=20535 tim=16183492978
WAIT #140567216695864: nam='direct path read' ela= 41224996 file number=4 first
dba=221184 block cnt=1920 obj#=20535 tim=16225494328
WAIT #140567216695864: nam='direct path read' ela= 37619632 file number=4 first
dba=223112 block cnt=2168 obj#=20535 tim=16263490285
WAIT #140567216695864: nam='direct path read' ela= 59552230 file number=4 first
dba=225280 block cnt=4096 obj#=20535 tim=16323503896
WAIT #140567216695864: nam='direct path read' ela= 36906264 file number=4 first
dba=229376 block cnt=1920 obj#=20535 tim=16361490691
```
Super huge IOs

• Investigate!
  • On the linux level:
    • Install libaio debuginfo package
    • Use gdb macro:
Super huge IOs

break io_submit
commands
  silent
  printf "io_submit - %d,%x - nr,ctx\n",nr,ctx
  set $c = nr -1
  while ( $c >= 0 )
    printf " fd: %d, nbytes: %d\n", iocbs[$c].aio_fildes, iocbs[$c].u.c.nbytes
    set $c = $c - 1
  end
end

break io_getevents_0_4
commands
  silent
  printf "io_getevents - min_nr: %d, ctx: %x, timeout { %d,%d }\n",min_nr,ctx,timeout.tv_sec,timeout.tv_nsec
end

break pread64
commands
  silent
  printf "pread64 - fd, size - %d,%d\n",$rdi,$rdx
end

break pwrite64
commands
  silent
  printf "pwrite64 - fd, size - %d,%d\n",$rdi,$rdx
end
Super huge IOs - gdb macro, ASM

io_submit - 32, 5e271000 - nr, ctx
fd: 256, nbytes: 983040
fd: 256, nbytes: 1048576
fd: 256, nbytes: 1048576
fd: 256, nbytes: 1048576

...more IOs, removed for clarity...
fd: 256, nbytes: 1048576
fd: 256, nbytes: 1048576

io_getevents - min_nr: 91, ctx: 5e271000, timeout { 0, 0 }
io_getevents - min_nr: 91, ctx: 5e271000, timeout { 0, 0 }
io_getevents - min_nr: 91, ctx: 5e271000, timeout { 0, 0 }
io_getevents - min_nr: 91, ctx: 5e271000, timeout { 0, 0 }
io_getevents - min_nr: 31, ctx: 5e271000, timeout { 600, 0 }
io_getevents - min_nr: 1, ctx: 5e271000, timeout { 600, 0 }
Super huge IOs - gdb macro, filesystem

io_submit - 1,c62fd000 - nr,ctx
  fd: 257, nbytes: 33554432

io_getevents - min_nr: 3, ctx: c62fd000, timeout { 0,0 }
io_getevents - min_nr: 3, ctx: c62fd000, timeout { 0,0 }
io_getevents - min_nr: 1, ctx: c62fd000, timeout { 600,0 }
Super huge IOs

• Conclusion:
  • When direct path multiblock IO is done.
  • The database can read up to 4096 blocks.
  • Tested with 8K blocksize.
  • If AU, BMB, MBRC allow.
Conclusion

• In Oracle version 10.2 and earlier non-PX reads use:
  • db file sequential read / db file scattered read events
  • Read blocks go to buffercache.

• Starting from Oracle version 11 reads could do both
  • buffered reads
  • unbuffered or direct path reads

• Segment size determination
  • Segment header or statistics (11.2.0.2+)
Conclusion

• Direct path read is decision in IO codepath of full scan.
• NOT an optimiser decision(!)

• In Oracle version 11, a read is done buffered, unless the database decides to do a direct path read
Conclusion

- Direct path read decision for tables is influenced by
  - Type of read (FTS)
  - Size of segment (> _small_table_threshold)
  - < VLOT: Number of blocks cached (< ~ 99%)
  - < VLOT: Number of blocks dirty (< ~ 50%) -- + deterministics

- Direct path read decision for indexes is influenced by
  - Type of read (FFIS)
  - Size of segment (> 5 * _very_large_object_threshold)
  - Number of blocks cached (< ~ 99%)
  - Number of blocks dirty (< ~ 50%) -- + deterministics
Conclusion

• By default, (AIO) direct path read uses two slots.
  • ‘autotune’ scales up in steps.

• Direct path code has an ‘autotune’ function, which can add IO slots.
  • I’ve witnessed it scale up to 32 slots.
  • In order to be able to use more bandwidth
  • Direct path ‘autotune’ works for PX reads too!

• ‘autotune’ does not kick in with Oracle version 11.2.0.1
  • Probably because some measurements return 0
Conclusion

• Buffered reads / db file scattered read:
  • Only reads a single range of blocks.
  • Bound by IO latency.
  • Limited scaling.

• PGA / unbuffered reads / direct path read:
  • Simultaneously reads two ranges of blocks.
  • Can scale up.
  • Can take advantage of modern high bandwidth IO.
Thank you for attending!

Questions?
Thanks, Links, etc.

- Tanel Poder, Kerry Osborne, Jason Arneil, Klaas-Jan Jongsma, Doug Burns, Cary Millsap, Freek d’Hooge, Jonathan Lewis, Christian Antognini.


- [http://www.oracle.com/pls/db112/homepage](http://www.oracle.com/pls/db112/homepage)


kfk: async disk IO

• Only seen with ‘direct path read’ waits and ASM
• Always seen in version 11.2.0.1
• Not normally seen in version 11.2.0.2+

• KFK = Kernel File K.. (?)
• Some ASM code is in this layer
kfk: async disk io

file # and # blocks are determined for a number of IO’s

Implementation (ASM) asynchronous IO 11.2.0.1

io_subit(aio_ctx, cb, {iocb})

io_getevents(aio_ctx, min_nr, nr, io_event, timeout)

ela time of ‘direct path read’*

read ready, blocks available

time
Super huge IOs - Extra slide

• A distinct next layer is the Linux device layer.
• Layer underneath SCI and scheduler.

• A device propagates its capabilities during initialisation.
• Block device capabilities are visible in: /sys/block/<device>/queue
• Max IO size: max_hw_sectors_kb (ro)
• Cur. IO size: max_sectors_kb (rw)
• Device queue size: nr_requests (rw)