Profiling the logwriter and database writer

Frits Hoogland
Trivadis Performance Days 2015

Updated fully to 12.1.0.2!
$(whoami)$

- 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:
Goals & prerequisites

• **Warning:** *This is a technical presentation!!*

• **Goal:** Learn about internal behaviour of both lgwr and dbwr, both visible (wait events) and inner-working.

• **Prerequisites:**
  – Understanding of (internal) execution of C programs.
  – Understanding of Oracle tracing mechanisms.
  – Understanding of interaction between processes inside the Oracle database.
Test system

• The tests and investigation is done in a VM:
  – Host: Mac OSX 10.10 / VMWare Fusion 7.1.2.
  – VM: Oracle Linux x86_64 6u7 (UEK3 3.8.13).
  – Oracle Grid 12.1.0.2 with ASM/External redundancy.
  – Oracle database 12.1.0.2.

  – Unless specified otherwise.
Logwriter, concepts guide

• From the concepts guide:
  – The lgwr manages the redolog buffer.

  – The lgwr writes all redo entries that have been copied in the buffer since the last time it wrote when:
    • User commits.
    • Logswitch.
    • Three seconds since last write*.
    • Buffer 1/3th full or 1MB filled.
    • dbwr must write modified (‘dirty’) buffers*.
Logwriter, idle

• The general behaviour of the log writer can easily be shown by putting a 10046/8 on lgwr:

SYS@v12102 AS SYSDBA> @who bg
...
93,1348,01 12275 DEDICATED oracle@bigmachine.local (LGWR)
...
SYS@v12102 AS SYSDBA> oradebug setospid 12275
Oracle pid: 13, Unix process pid: 12275, image: oracle@bigmachine.local (LGWR)
SYS@v12102 AS SYSDBA> oradebug unlimit
Statement processed.
SYS@v12102 AS SYSDBA> oradebug event 10046 trace name context forever, level 8;
Statement processed.
Logwriter, idle

• The 10046/8 trace shows:

*** 2013-12-18 14:12:32.479
WAIT #0: nam='rdbms ipc message' ela= 2999925 timeout=300 p2=0 p3=0 obj#=-1
tim=1387372352479352

*** 2013-12-18 14:12:35.479
WAIT #0: nam='rdbms ipc message' ela= 3000075 timeout=300 p2=0 p3=0 obj#=-1
tim=1387372355479531

*** 2013-12-18 14:12:38.479
WAIT #0: nam='rdbms ipc message' ela= 2999755 timeout=300 p2=0 p3=0 obj#=-1
tim=1387372358479381

*** 2013-12-18 14:12:41.479
WAIT #0: nam='rdbms ipc message' ela= 3000021 timeout=300 p2=0 p3=0 obj#=-1
tim=1387372361479499
Logwriter, idle

• “rdbms ipc message” indicates a sleep/idle event
  – strace (linux system call trace) on LGWR.
  – There isn’t an indication lgwr writes something:

  semtimedop(327683, {{15, -1, 0}}, 1, {3, 0}) = -1 EAGAIN (Resource temporarily unavailable)
  getrusage(RUSAGE_SELF, {ru_utime={0, 84000}, ru_stime={0, 700000}, ...}) = 0
  getrusage(RUSAGE_SELF, {ru_utime={0, 84000}, ru_stime={0, 700000}, ...}) = 0
  times({tms_utime=8, tms_stime=70, tms_cutime=0, tms_cstime=0}) = 431286151
  times({tms_utime=8, tms_stime=70, tms_cutime=0, tms_cstime=0}) = 431286151
  times({tms_utime=8, tms_stime=70, tms_cutime=0, tms_cstime=0}) = 431286151
  times({tms_utime=8, tms_stime=70, tms_cutime=0, tms_cstime=0}) = 431286151
  semtimedop(327683, {{15, -1, 0}}, 1, {3, 0}) = -1 EAGAIN (Resource temporarily unavailable)

  etc...
Logwriter, idle

- It does look in the /proc filesystem to the ‘stat’ file of a certain process:

  ```
  open("/proc/2218/stat", O_RDONLY)       = 21
  read(21, "2218 (oracle) S 1 2218 2218 0 -1"..., 999) = 209
  close(21)
  ```

- It does so every 20th time (20*3)= 60 sec.
- The PID is PMON.
Logwriter, idle

• Recap:
  – In an idle database.
  – The lgwr sleeps on a semaphore for 3 seconds.
    • Then wakes up, and sets up the semaphore/sleep again.
    • Processes sleeping on a semaphore do not spend CPU
  – Every minute, lgwr reads pmon's process stats.
  – lgwr doesn’t write if there’s no need.
• But what happens when we insert a row of data, and commit that?
Logwriter, commit

```
TS@//localhost/v11204 > insert into t values ( 1, 'aaaa', 'bbbb' );
1 row created.

TS@//localhost/v11204 > commit;
Commit complete.
```
Logwriter, commit - expected

commit;

foreground

logwriter

semctl(458755, 15, SETVAL, 0x7fff00000001)
semtimedop(458755, {{33, -1, 0}}, 1, {0, 100000000})

semctl(458755, 33, SETVAL, 0x1)
semtimedop(458755, {{15, -1, 0}}, 1, {3, 0})
io_submit(139981752844288, 1, {{0x7f5008e23480, 0, 1, 0, 256}})
io_getevents(139981752844288, 1, 128, {{0x7f5008e23480, 0x7f5008e23480, 3584, 0}, {600, 0}})
Logwriter, commit - actual

- `semctl(458755, 15, SETVAL, 0x7fff00000001)`
- `semtimedop(458755, {{15, -1, 0}}, 1, {3, 0})`
- `commit;`
- `io_submit(139981752844288, 1, {{0x7f5008e23480, 0, 1, 0, 256}})`
- `io_getevents(139981752844288, 1, 128, {{0x7f5008e23480, 0x7f5008e23480, 3584, 0}}, {600, 0})`

No 'log file sync' wait!

- No `semtimedop()`
- No `semctl()`

foreground

logwriter

commit;
Logwriter, commit

• Investigation shows:
  – Foreground scans log writer progress up to 3 times.
    • kcrf_commit_force() calls kcscur3()
  
  – If its data* in the redo log buffer is not written:
    • It notifies the lgwr that it is going to sleep on a semaphore.
    • semtimedop() for 100ms, until posted by lgwr.
  
  – If its data* has been written:
    • No need to wait on it.
    • No ‘log file sync’ wait.
Logwriter, commit

• Wait!!!
  – This (no log file sync) turned out to be an *edge case*.
    • I traced the kcrf_commit_force() and kcscurs3() calls using breaks in gdb.

  – In normal situations, the wait will appear.
    • Depending on log writer and FG progress.
    • The semtimedop() call in the FG can be absent.
      – As a result, lgwr will not semctl()
Logwriter, commit - post-wait

semctl(458755, 15, SETVAL, 0x7fff00000001)

c헛

semtimedop(458755, {{33, -1, 0}}, 1, {0, 100000000})

commit;

foreground

log file parallel write

log file sync

logwriter

semtimedop(458755, {{15, -1, 0}}, 1, {3, 0})

io_submit(139981752844288, 1,
{{0x7f5008e23480, 0, 1, 0, 256}})

log file parallel write

rdbms ipc message

kcscur3()

kcrf_commit_force()

io_getevents(139981752844288, 1, 128,
{{0x7f5008e23480, 0x7f5008e23480, 3584, 0},
{600, 0}})

>=12.1.0.1: kcrf_commit_force_int()

Grayed means: optional
• How did I come up with the function names??
  – The linux ‘perf’ utility
    • top: sample which functions are used.
    • record: record sampled functions, opt. with backtrace.
  – The GNU debugger: gdb
    • Can be used to break on c functions.
    • This can establish a timeline, or view function arguments.
  – Systemtap
    • “linux answer to dtrace”
My gdb script used for profiling:

break io_submit
commands
  silent
  printf "io_submit - nr:%d\n", nr
cend
break io_getevents_0_4
commands
  silent
  printf "io_getevents_0_4 - min_nr:%d, timeout:%d,%d\n", min_nr, timeout.tv_sec, timeout.tv_nsec
cend

Requires debuginfo of libaio
Extra page

break skgfr_return64
  commands
  silent
  printf "skgfr_return64\n"
cend

break kslwtbctx
  commands
  silent
  printf "kslwtbctx "
cend

break kskthewt
  commands
  silent
  printf "%d\n", $rsi
cend

break semtimedop
  commands
  silent
  printf "semtimedop\n"
cend

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

break pwrite64
  commands
  silent
  printf "pwrite64 fd, size: %d,%d\n",$rdi,$rdx
cend

Take advantage of the way Linux x86_64 passes arguments via CPU registers. All gdb macro’s on this page do NOT require debuginfo.
adaptive log file sync

• Feature of Oracle 11.2
  – Parameter '_use_adaptive_log_file_sync'
    • Set to FALSE up to 11.2.0.2
    • Set to TRUE starting from 11.2.0.3
    • Third value ‘POLLING_ONLY’
  – Makes Oracle adaptively switch between ‘post-wait’ and polling.
  – The log writer writes a notification in its logfile if it switches between modes (if param = ‘TRUE’).
Logwriter, commit - polling

foreground

logwriter

semct1(458755, 15, SETVAL, 0x7fff00000001)

kcrf_commit_force

kcscur3

nanosleep({0, 9409000}, 0x7fff64725480)

log file sync

log file parallel write

commit;

rdbms ipc message

SEMtimeout(458755, {{15, -1, 0}}, 1, {3, 0})

io_submit(139981752844288, 1, {{0x7f5008e23480, 0, 1, 0, 256}})

io_getevents(139981752844288, 1, 128, {{0x7f5008e23480, 0x7f5008e23480, 3584, 0}}, {600, 0})

>=12.1.0.1: kcrf_commit_force_int()
Logwriter, post-wait vs. polling

– No wait event ‘log file sync’ if:
  • Lgwr was able to flush the committed data before the foreground has issued kcscur3() 2/3 times in kcrf_commit_force() / kcrf_commit_force_int().

– If not, the foreground starts a ‘log file sync’ wait.
  • If in “post-wait” mode (default), it will record it’s waiting state in the post-wait queue, sleep in semtimedop() for 100ms at a time, waiting to be posted by lgwr.
  • If in “polling” mode, it will sleep in nanosleep() for computed time*, then check lgwr progress, if lgwr write has progressed beyond its committed data SCN: end wait, else start sleeping in nanosleep() again.
Logwriter

• The main task of lgwr is to flush data in the logbuffer to disk.
  – The lgwr is idle when waiting on ‘rdbms ipc message’.
  – There are two main* indicators of lgwr busyness:
    • CPU time.
    • Wait event ‘log file parallel write’.

• The lgwr needs to be able to get onto the CPU in order to do process!
Logwriter - idle

semimedop(458755, {{15, -1, 0}}, 1, {3, 0})
Logwriter - idle

logwriter

- rdbms ipc message
- rdbms ipc message
Logwriter - idle

Idle mode latch gets:
- messages
- mostly latch-free SCN
- lgwr LWN SCN
- KTF sga latch
- redo allocation
- messages
Logwriter - writing

Write mode latch gets and frees:
‘messages’
‘mostly latch-free SCN’
‘lgwr LWN SCN’
‘KTF sga latch’
‘redo allocation’
‘messages’
‘redo writing’*
Logwriter - writing (11.2.0.3 - ASM)

With the Linux ‘strace’ utility, the non-blocking syscall is visible OR the blocking one syscall is visible.
Logwriter - writing (11.2.0.3 - ASM)

- The event ‘log file parallel write’ times:
  - Depends on version and ASM or filesystem.
  - Might not be IO latency time!!
Log writer - writing - ASM

- `semtimedop()`
- `io_submit()`
- `io_getevents()` timeout 0s
- `io_getevents()` timeout 600s

- kslwtectx()
  - rdbms ipc message
- kslwtectx()
  - log file parallel write
Log writer - writing - filesystem

- `semtimedop()`
- `io_submit()`
- `io_getevents()` timeout 600s

- `kslwtectx()` rdbms ipc message
- `kslwtectx()` log file parallel write
Logwriter - writing - 12c

• Actually, the version 12 schema could be false.

• A default Oracle 12 database uses a new feature:
  – Adaptive scalable lgwr workers

• Which means you got a master lgwr process,
  – and log writer slaves (lgnn processes)
Logwriter - writing - 12c

• The adaptive scalable logwr workers feature is controlled by the parameter:
  • _use_single_log_writer
    • ‘ADAPTIVE’ (default), ‘TRUE’, ‘FALSE’.

• I did set it to ‘TRUE’ to revert to 11g behaviour.
Logwriter - writing - 12c

• When adaptive, on my system*:
  • lgwr process.
  • 2 lgmn processes.

• ‘adaptive’ means either the lgwr process or the slave(s) write.
Logwriter - writing - 12c

• The log writer trace file tells what is happening:
  • Writing is moved from LGWR to slaves:

    *** 2014-11-15 13:16:04.003
    kcrfw_slave_adaptive_updatemode: single->scalable reodrate=182371 switch=37881

    *** 2014-11-15 13:16:04.004
    Adaptive scalable LGWR enabling workers

  • Writing is moved from slaves to LGWR:

    kcrfw_slave_adaptive_updatemode: scalable->single group0=2527 all=2552 rw=3957
    single=3100 scalable_nopipe=7914 scalable_pipe=4352 scalable=7878

    Adaptive scalable LGWR disabling workers
Logwriter - writing - 12c

• The waits are a bit different between single and scalable mode:
  • Single (LGWR) writes are discussed in this presentation.
    • The lgnn processes waits for
      – ‘LGWR worker group idle’ forever.

• This means the wait time is either startup or last time they wrote.
Logwriter - writing - 12c

• The waits are a bit different between single and scalable mode:
  • In scalable mode, LGWR receives write req.
    • LGWR semctl’s one or more slave’s to write.
    • Then sleeps in ‘rdbms ipc message’.
  • The lgnn processes wakes up, and writes.
    – io_submit&io_getevents in wait ‘log file parallel write’.
    – semctl’s FG once ready.
Logwriter - writing - 12c

• In scalable mode:
  • I suspended execution of the slaves.
  • After some time, this is noticed by LGWR:
    • Wait ‘target log write size’.
    • Wait ‘LGWR all worker groups’.
Logwriter wait events

• rdbms ipc message
  – timeout: 300 (centiseconds; 3 seconds).
  – process sleeping ~ 3 seconds on semaphore.

• log file parallel write
  – files: number of log file members.
  – blocks: total number of log blocks written.
  – requests: ?
    • I’ve seen this differ from the actual number of IO requests.
Logwriter wait events

• Let’s switch the database to synchronous IO.
  – Some platforms have difficulty with AIO (HPUX!)
  – Got to check if your config does use AIO.
    • Found out by accident that ASM+NFS has no AIO by default.
      – (need to set filesystemio_options to ‘setall’)
  – Good to understand what the absence of AIO means.

• If you can’t use AIO today, you are doing it WRONG!
log file parallel write (11204-SIO-ASM)

kslwtbctx 7
semtimedop
kslwtectx 7

pwrite64 fd, size: 256,1024
pwrite64 fd, size: 256,1024
kslwtbctx 135
kslwtectx 135

kslwtbctx 7
semtimedop
kslwtectx 7

pwrite64 fd, size: 256,1024
pwrite64 fd, size: 256,1024
kslwtbctx 135
kslwtectx 135
log file parallel write (12102-SIO-ASM)

kslwtbctx 8
semtimedop
kslwtectx 8

kslwtbctx 137
pwrite64 fd, size: 256,1024
pwrite64 fd, size: 256,1024
kslwtectx 137

kslwtbctx 8
semtimedop
kslwtectx 8

kslwtbctx 137
pwrite64 fd, size: 256,1024
pwrite64 fd, size: 256,1024
kslwtectx 137
Log writer - writing - SIO - ASM
Log writer - writing - SIO - filesystem

semimedop()  pwrite64()

11.2.0.1
11.2.0.2
11.2.0.3
11.2.0.4
12.1.0.1
12.1.0.2

kslwtectx()
rdbms ipc message

kslwtectx()
log file parallel write
log file parallel write

• Conclusion:
  – For Oracle versions up to 12.1.0.1.
  – Wait event ‘log file parallel write’.
  – ASM in use.
  – Synchronous IO (pwrite64() calls).
  – The wait event does not time the IO requests.

• How about the other log writer wait events?
## logwriter other IO & waits ASM

<table>
<thead>
<tr>
<th></th>
<th>Calls with AIO enabled</th>
<th>Calls with AIO disabled</th>
<th>Timing correct with AIO disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>log file parallel write</td>
<td>io_submit / io_getevents</td>
<td>pwrite64</td>
<td>NO</td>
</tr>
<tr>
<td>log file single write</td>
<td>pwrite64</td>
<td>pwrite64</td>
<td>YES</td>
</tr>
<tr>
<td>log file sequential read</td>
<td>pread64</td>
<td>pread64</td>
<td>YES</td>
</tr>
<tr>
<td>control file sequential read</td>
<td>pread64</td>
<td>pread64</td>
<td>YES</td>
</tr>
<tr>
<td>control file parallel write</td>
<td>io_submit / io_getevents</td>
<td>pwrite64</td>
<td>NO</td>
</tr>
</tbody>
</table>
Logwriter wait events logswitch

• Some of these waits typically show up during a logswitch.
  – This are all the waits which are normally seen:
    • os thread startup (semctl()-semtimedop())
    • control file sequential read (pread64())
    • control file parallel write (io_submit()-io_getevents())
    • log file sequential read (pread64())
    • log file single write (pwrite64())
    • KSV master wait (semctl() post to dbwr)

• This is with AIO enabled!
Logwriter, timeout message

• Warning:

Warning: log write elapsed time 523ms, size 2760KB

• Printed in logwriter tracefile (NOT alert.log)

• This is instrumented with the ‘log write parallel write’ event.

• Threshold set with parameter:
  – _side_channel_batch_batch_timeout_ms (500ms)
Logwriter, timeout message

- Warning (RAC!):

  Warning: log write broadcast wait time 2913ms (SCN 0xb86.cd638134)

- Printed in logwriter tracefile (NOT alert.log)
- This is instrumented with the ‘wait for scn ack’ event.
Logwriter: disable logging

• The “forbidden switch”: _disable_logging
  – Do not use this for anything else than tests!

• Everything is done the same — no magic
  – Except the write by the lgwr to the logfiles
  – No ‘log file parallel write’
  – Redo/control/data files are synced with shut normal

• A way to test if lgwr IO influences db processing
Logwriter: exadata

• How does this look like on Exadata?
Logwriter: exadata (11.2.0.4)

kslwtbctx 7
semtimedop
kslwtectx 7

oss_write
oss_write
kslwtbctx 135
oss_wait
oss_wait
kslwtectx 135

kslwtbctx 7
semtimedop
kslwtectx 7

oss_write
oss_write
kslwtbctx 135
oss_wait
oss_wait
kslwtectx 135
Logwriter: exadata (12.1.0.2)

kslwtbctx 8
semtimedop
kslwtectx 8

kslwtbctx 137
oss_write
oss_write
oss_write
oss_write
oss_write
oss_wait
oss_wait
kslwtectx 137

kslwtbctx 8
semtimedop
kslwtectx 8

kslwtbctx 137
oss_write
oss_write
oss_write
oss_write
oss_write
oss_wait
oss_wait
oss_wait
kslwtectx 137
Database writer

- From the Oracle 11.2 concepts guide:
  - The DBWn process writes dirty buffers to disk under the following conditions:
    - When a server process cannot find a clean reusable buffer after scanning a threshold of buffers, it signals DBWn to write. DBWn writes dirty buffers to disk asynchronously if possible while performing other processing.
    - DBWn periodically writes buffers to advance the checkpoint, which is the position in the redo thread from which instance recovery begins. The log position of the checkpoint is determined by the oldest dirty buffer in the buffer cache.
Database writer, idle

• The 10046/8 trace shows:

*** 2013-12-31 00:45:51.088
WAIT #0: nam='rdbms ipc message' ela= 3006219 timeout=300 p2=0 p3=0 obj#=-1
tim=1388447151086891

*** 2013-12-31 00:45:54.142
WAIT #0: nam='rdbms ipc message' ela= 3005237 timeout=300 p2=0 p3=0 obj#=-1
tim=1388447154140873

*** 2013-12-31 00:45:57.197
WAIT #0: nam='rdbms ipc message' ela= 3005258 timeout=300 p2=0 p3=0 obj#=-1
tim=1388447157195828

*** 2013-12-31 00:46:00.255
WAIT #0: nam='rdbms ipc message' ela= 3005716 timeout=300 p2=0 p3=0 obj#=-1
tim=1388447160253960
Database writer, idle

• “rdbms ipc message” indicates a sleep/idle event
  – There isn’t an indication dbw0 writes something:

  semtimedop(983043, {{14, -1, 0}}, 1, {3, 0}) = -1 EAGAIN (Resource temporarily unavailable)
  getrusage(RUSAGE_SELF, {ru_utime={0, 31000}, ru_stime={0, 89000}, ...}) = 0
  getrusage(RUSAGE_SELF, {ru_utime={0, 31000}, ru_stime={0, 89000}, ...}) = 0
  times({tms_utime=3, tms_stime=8, tms_cutime=0, tms_cstime=0}) = 431915044
  times({tms_utime=3, tms_stime=8, tms_cutime=0, tms_cstime=0}) = 431915044
  times({tms_utime=3, tms_stime=8, tms_cutime=0, tms_cstime=0}) = 431915044
  semtimedop(983043, {{14, -1, 0}}, 1, {3, 0}) = -1 EAGAIN (Resource temporarily unavailable)
  ...etc...
Database writer, idle

• It does look in the /proc filesystem to the ‘stat’ file of a certain process:

```c
open("/proc/2218/stat", O_RDONLY) = 21
read(21, "2218 (oracle) S 1 2218 2218 0 -1"..., 999) = 209
close(21)
```

• It does so every 20th time (20*3)= 60 sec.
• The PID is PMON.
Database writer, idle

• Recap:
  – In an idle database.
  – The dbwr sleeps on a semaphore for 3 seconds.
    • Then wakes up, and sets up the semaphore/sleep again.
    • Processes sleeping on a semaphore do not spend CPU.
  – Every minute, dbwr reads pmon's process stats.
  – dbwr doesn’t write if there’s no need.
Database writer, force write

• We can force the dbwr to write:
  – Dirty some blocks (insert a row into a table).
  – Force a thread checkpoint (alter system checkpoint).

* There are multiple ways, this is one of them.
Database writer, force write

10046/8 trace:

- **db file async I/O submit?!**
  - It looks like the io_submit() call is instrumented for the dbwr!
  - But what does ‘requests=3’ mean for a single row update checkpoint?

WAIT #0: nam='rdbms ipc message' ela= 2261867 timeout=300 p2=0 p3=0 obj#=-1 tim=1388716669735046

WAIT #0: nam='db file async I/O submit' ela= 0 requests=3 interrupt=0 timeout=0 obj#=-1 tim=1388716669735493

WAIT #0: nam='db file parallel write' ela= 21 requests=1 interrupt=0 timeout=2147483647 obj#=-1 tim=1388716669735566

*** 2014-01-03 03:37:50.465

WAIT #0: nam='rdbms ipc message' ela= 729110 timeout=73 p2=0 p3=0 obj#=-1 tim=1388716670464967

And the write, via the event ‘db file parallel write’.
Let’s take a look at the Oracle wait events, together with the actual system calls.

That is:
- Setting a 10046/8 event for trace and waits.
- Execute `strace` with ‘-e write=all -e all’
dbwr, sql_trace + strace

io_submit(140195085938688, 3, {{0x7f81b622ab10, 0, 1, 0, 256}, {0x7f81b622a8a0, 0, 1, 0, 256}, {0x7f81b622a630, 0, 1, 0, 256}}) = 3

write(13, "WAIT #0: nam='db file async I/O ...", 108) = 108

| 00000 57 41 49 54 20 23 30 3a 20 6e 61 6d 3d 27 64 62 WAIT #0: nam='db | |
| 00010 20 66 69 6c 65 20 77 69 74 65 27 20 65 6c 61 3d 20 31 20 file async I/O | |
| 00020 73 75 62 6d 69 74 73 74 73 3d 33 20 69 6e 74 65 72 72 quests= 3 inter | |
| 00030 65 71 75 65 73 74 73 3d 33 20 69 6e 74 65 72 72 obj#=-1 tim=1388 | |
| 00040 75 70 74 3d 30 20 74 69 6d 65 6f 75 74 3d 32 31 4 upt=0 timeout=214 | |
| 00050 6f 62 6a 23 3d 30 34 35 37 39 74 7765180 4261 | |

io_getevents(140195085938688, 1, 128, {{0x7f81b622ab10, 0x7f81b622ab10, 8192, 0}, {0x7f81b622a8a0, 0x7f81b622a8a0, 8192, 0}, {0x7f81b622a630, 0x7f81b622a630, 8192, 0}, {600, 0}}) = 3

write(13, "WAIT #0: nam='db file parallel w"...", 116) = 116

| 00000 57 41 49 54 20 23 30 3a 20 6e 61 6d 3d 27 64 62 WAIT #0: nam='db | |
| 00010 20 66 69 6c 65 20 77 69 74 65 27 20 65 6c 61 3d 20 35 38 rite' ela= 58 re | |
| 00020 72 69 74 65 27 20 65 6c 61 3d 20 31 20 file parallel w | |
| 00030 71 75 65 73 74 73 3d 31 20 69 6e 74 65 72 72 quests=1 interru | |
| 00040 70 74 3d 30 20 74 69 6d 65 6f 75 74 3d 32 31 34 pt=0 tim eout=214 | |
| 00050 37 34 33 36 34 37 39 74 7483647 obj#=-1 tim=1388 97765180 | |
| 00060 74 69 6d 3d 31 33 38 38 39 37 37 36 35 31 38 30 tim=1388 97765180 | |
| 00070 34 35 37 39 4579 | |
dbwr, sql_trace + strace

```c
io_submit(140195085938688, 3, {{0x7f81b622ab10, 0, 1, 0, 256}, {0x7f81b622a8a0, 0, 1, 0, 256}, {0x7f81b622a630, 0, 1, 0, 256}}) = 3
write(13, "WAIT #0: nam='db file async I/O "..., 108) = 108
| 00000  57 41 49 54 20 23 30 3a 20 6e 61 6d 3d 27 64 62 | WAIT #0: nam='db|
| 00010  20 66 69 6c 65 20 61 73 79 6e 63 20 49 2f 4f 20 | file async I/O|
| 00020  73 75 62 6d 69 74 27 20 65 6c 61 3d 20 31 20 72 | submit el a= 1 r|
| 00030  66 69 6c 65 61 73 79 6e 63 20 49 2f 4f 20 | equests= 3 inter r|
| 00040  75 70 74 3d 30 20 74 69 6d 65 6f 75 74 3d 30 20 | upt=0 ti meout=0|
| 00050  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 00060  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 00070  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 00080  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 00090  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 000a0  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 000b0  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 000c0  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 000d0  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 000e0  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 000f0  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 00100  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 00110  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 00120  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 00130  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 00140  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 00150  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|
| 00160  39 37 77 6e 74 65 72 72 75 70 74 3d 30 20 74 69 97765180 4261 |
| 00170  6f 62 6a 23 3d 3d 33 20 69 6e 74 65 72 72 75 | obj#=-1 tim=1388|

io_getevents(140195085938688, 1, 128, {{0x7f81b622ab10, 0x7f81b622ab10, 8192, 0}, {0x7f81b622a8a0, 0x7f81b622a8a0, 8192, 0}, {0x7f81b622a630, 0x7f81b622a630, 8192, 0}, {600, 0}}) = 3
write(13, "WAIT #0: nam='db file parallel w"..., 116) = 116
| 00000  57 41 49 54 20 23 30 3a 20 6e 61 6d 3d 27 64 62 | WAIT #0: nam='db|
| 00010  20 66 69 6c 65 20 70 61 72 61 6c 6c 65 6c 20 77 | file pa rallel w|
| 00020  72 69 74 65 27 20 65 6c 61 3d 20 58 20 72 65 | rite' el a= 58 re|
| 00030  71 75 65 65 73 74 73 3d 31 20 69 6e 74 65 72 72 75 | quests=1 inter ru|
| 00040  70 74 3d 30 20 74 69 6d 65 6f 75 74 3d 32 31 34 | pt=0 tim eout=214|
| 00050  37 34 38 33 36 34 37 20 6f 62 6a 23 3d 2d 3d 2d 31 34 | 7483647 obj#=-1|
| 00060  74 69 6d 3d 31 33 33 33 32 36 34 37 20 39 37 37 36 35 31 38 30 | tim=1388 97765180|
| 00070  34 35 37 39 |
```

enkitec
dbwr, sql_trace + strace

io_submit(140195085938688, 3, {{0x7f81b622ab10, 0, 1, 0, 256}, {0x7f81b622a8a0, 0, 1, 0, 256}, {0x7f81b622a630, 0, 1, 0, 256}}) = 3

write(13, "WAIT #0: nam='db file async I/O "..., 108) = 108

io_getevents(140195085938688, 1, 128, {{0x7f81b622ab10, 0x7f81b622ab10, 8192, 0}, {0x7f81b622a8a0, 0x7f81b622a8a0, 8192, 0}, {0x7f81b622a630, 0x7f81b622a630, 8192, 0}}, {600, 0}) = 3

write(13, "WAIT #0: nam='db file parallel w"..., 116) = 116

This is the MINIMAL number of requests to reap before successful.
(min_nr - see man io_getevents)

The timeout for io_getevents() is set to 600 seconds.
(struct timespec { sec, nsec })

Despite only needing 1 request, this call returned all 3.
This information is NOT EXTERNALISED (!!!)

This is the MINIMAL number of requests to reap before successful.
(min_nr - see man io_getevents)

The timeout for io_getevents() is set to 600 seconds.
(struct timespec { sec, nsec })

Despite only needing 1 request, this call returned all 3.
This information is NOT EXTERNALISED (!!!)
dbwr, db file async I/O submit

• Let’s take a look at what the documentation says about “db file async I/O submit”:

  db file asynch I/O submit
  When asynchronous I/O is available, this wait event captures any time spent in submitting I/Os to the underlying storage.
  See Also:
  "db file parallel write"

• Indicates io_submit() being timed.
• This seems to be added recently!
• Previously there was no documentation.
dbwr, db file async I/O submit ASM

- Database version 12.1.0.2

  kslwtbctx 8
  semtimedop
  kslwtectx 8

  io_submit - nr:13
  kslwtbctx 157
  kslwtectx 157
  kslwtbctx 157
  kslwtbctx 156
  io_getevents_0_4 - min_nr:1, timeout:600,0
  skgfr_return64
  ...
  skgfr_return64
  kslwtectx 156

  kslwtbctx 8
  semtimedop
  kslwtectx 8
Database writer - writing - ASM

semimedop()  
io_submit()  
io_getevents() timeout 600s  
io_getevents() timeout 0s  
kslwtectx() / kslwtbctx()  
db file parallel write

11.2.0.1  
11.2.0.2  
11.2.0.3  
11.2.0.4  
12.1.0.1  
12.1.0.2

kslwtectx()  
rbms ipc message  
kslwtbctx() / kslwtectx()  
db file async I/O submit  
kslwtbctx() / kslwtectx()  
db file parallel write
Database writer - writing - filesystem

- `semtimedop()`
- `io_submit()`
- `io_getevents()` timeout 600s
- `io_getevents()` timeout 0s

- kslwtectx()
- rdbms ipc message
- kslwtctx() / kslwtectx()
- db file async I/O submit
- kslwtctx() / kslwtectx()
- db file parallel write

- 11.2.0.1
- 11.2.0.2
- 11.2.0.3
- 12.1.0.1
- 12.1.0.2
- 12.1.0.4
dbwr, db file async I/O submit

• Event “db file async I/O submit”:
  • Does not time io_submit with ASM.
    • Clearly a bug.
  • What does the wait indicate?
    • ASM: blocking on io_submit.
      – Device queue full.
  • filesystem: lack of CIO.
    – Concurrent IO; ext4 lacks CIO.
dbwr, db file parallel write

• Let’s look at the “db file parallel write” event.
dbwr, db file parallel write

• Description from the Reference Guide:

**db file parallel write**

This event occurs in the DBWR. It indicates that the DBWR is performing a parallel write to files and blocks. When the last I/O has gone to disk, the wait ends.

**Wait Time:** Wait until all of the I/Os are completed.

**Parameter Description**

requests: This indicates the total number of I/O requests, which will be the same as blocks
interrupt: timeout: This indicates the timeout value in hundredths of a second to wait for the I/O completion.
dbwr, db file parallel write

• *New description from the Reference Guide:*

`db file parallel write`

This event occurs in the DBWR. It indicates the time that DBWR spends waiting for I/O completion.

If asynchronous I/O is available, then the db file asynch I/O submit wait event captures any time spent in submitting I/Os to the underlying storage.

When asynchronous I/O is not available, `db file parallel write` captures the time spent during submit and reap.

**Wait Time:** While there are outstanding I/Os, DBWR waits for some of the writes to complete. DBWR does not wait for all of the outstanding I/Os to complete.

requests: This indicates the total number of I/O requests, which will be the same as blocks.

interrupt:

timeout: This indicates the timeout value in hundredths of a second to wait for the I/O completion.
dbwr, db file parallel write

• Recap of previous traced calls:

  kslwtbctx 8
  semtimedop
  kslwtecxt 8

  io_submit - nr:3
  kslwtbctx 157
  kslwtecxt 157
  kslwtbctx 156

  io_getevents_0_4 - min_nr:1, timeout:600,0
  skgfr_return64
  skgfr_return64
  skgfr_return64
  skgfr_return64
  kslwtecxt 156

  kslwtbctx 8
  semtimedop
  kslwtecxt 8

So….how about severely limiting OS IO capacity and see what happens?
dbwr, db file parallel write

- **Database writer — severely limited IO (1 IOPS)**

  19 IO requests are submitted to the kernel.

  But only 2 IOs are needed to satisfy `io_getevents()`
  Which it does in this case… leaving outstanding IOs.

  The dbwr starts issuing non-blocking calls to reap IOs!
  It seems to be always 2 if outstanding IOs remain.
  Minnr = # outstanding IOs, max 128.
Database writer - writing - ASM

11.2.0.1

11.2.0.2

11.2.0.3

11.2.0.4

12.1.0.1

12.1.0.2

semimedop()  io_submit()  io_getevents() (timeout 600s)  io_getevents() (timeout 0s)  kslwtbctx() / kslwtectx() (db file parallel write)

kslwtectx() (rdbms ipc message)  kslwtbctx() / kslwtectx() (db file async I/O submit)  kslwtbctx() / kslwtectx() (db file parallel write)
Database writer - writing - filesystem

- `semtimedop()`
- `io_submit()`
- `io_getevents()` timeout 600s
- `io_getevents()` timeout 0s

Commands:
- `semtimedop()`
- `io_submit()`
- `io_getevents()`

Operations:
- `kslwtectx()`
- `rdbms ipc message`
- `kslwtectx()`
- `db file async I/O submit`
- `kslwtectx()`
- `db file parallel write`
dbwr, db file parallel write

• This got me thinking...
• The dbwr submits the IOs it needs to write.

• But it waits for a variable amount of IOs to finish.
  – Wait event ‘db file parallel write’.
  – Amount seems 33-25% of submitted IOs*
  – After that, 2 tries to reap the remaining IOs*
  – Then either submit again, DFPW until IOs reaped or back to sleeping on semaphore.
dbwr, db file parallel write

• This means ‘db file parallel write’ is not:
  – Physical IO indicator.
  – IO latency timing

• I’ve come to the conclusion that the blocking io_getevents call for a number of IOs of the dbwr is an IO limiter/throttle.
• ...and ‘db file parallel write’ is the timing of it.
dbwr, synchronous IO

• Let’s turn AIO off again.
  – To simulate this, I’ve set disk_asynch_io to FALSE.

• And set a 10046/8 trace and strace on the dbwr.
• And issue the SQLs as before:
  – insert into followed by commit
  – alter system checkpoint
dbwr, SIO, 11.2.0.4, ASM

3 pwrite() calls. This is synchronous IO!

The db file parallel write wait event shows 3 requests!

But why a second db file parallel write wait event?
dbwr, synchronous IO

• There’s no ‘db file async I/O submit’ wait anymore.
  – Which is good, because SIO has no submit phase.
• The ‘db file parallel write’ waits seem suspicious.
  – It seems like the wait for DFPW is issued twice.
  – Without the need for a second wait.
• Let’s look a level deeper and see if there’s more to see.
dbwr, SIO (11.2.0.4 - ASM)

kslwtbctx 7
semtimedop
kslwtectx 7

pwrite64 fd, size: 256,8192
pwrite64 fd, size: 256,8192
pwrite64 fd, size: 256,8192
kslwtbctx 150
kslwtectx 150
kslwtbctx 150
kslwtectx 150
kslwtbctx 7
semtimedop
kslwtectx 7

3 IO’s via pwrite() calls.

Two db file parallel write waits (which aren’t parallel) for which both the begin of the waits are started AFTER the IOs (!!)
The 3 synchronous IO’s are now inside the first wait.

However, there still is a second ‘db file parallel write’ wait. Which doesn’t time any IO.
dbwr, synchronous IO and ASM

• So, my conclusion on the wait events for the dbwr with synchronous IO and ASM:
  — The events are not properly timed
  — It seems like the wait for DFPW is issued twice.
  — My guess this is a bug in the synchronous IO implementation in ASM.
dbwr - writing - SIO - ASM

- semtimedop()
- pwrite64()
- kslwtectx()
  rdbms ipc message
- kslwtectx()
  db file parallel write
dbwr - writing - SIO - filesystem

semtimedop()

11.2.0.1
11.2.0.2
11.2.0.3
11.2.0.4
12.1.0.1
12.1.0.2

pwrite64()

kslwtectx()

rdbms ipc message

db file parallel write
dbwr: exadata

• How does this look like on Exadata?
dbwr: exadata 11.2.0.4

oss_write() issues a write request to a cell. They seem to perform the same function as io_submit().

This is the ‘db file async I/O submit’ event. Alike non-exadata it does not actually time submitting the I/Os.

This is the ‘db file parallel write’ event. It seems to include up to 6 oss_wait() calls.

If there are more than 6 writes submitted, they are reaped outside of the wait()!!
dbwr: exadata 12.1.0.2

oss_write(): submitting I/Os

This is the ‘db file async I/O submit’ event.

This is the ‘db file parallel write’ event. With 12.1.0.2, I see 2 or 4 oss_wait() calls timed in the wait. Also, the 1:1 relationship between oss_write() and oss_wait() calls seems to be a 2:1 relationship now.

I/Os reaped outside of the ‘db file parallel write’ event!
Conclusion

- Logwriter:
  - When idle, is sleeping on a semaphore/rdbms ipc message
  - Gets posted with semctl() to do work.
  - Only writes when it needs to do so.
  - Version 11.2.0.3: two methods for posting FGs:
    - Polling and post/wait.
    - Post/wait is default, might switch to polling.
    - Notification of switch is in log writer trace file.
    - Polling/nanosleep() time is variable.
Conclusion

• Logwriter:
  – Log file parallel write
    – AIO: two io_getevents() calls.
    – AIO: time waiting for all lgwr submitted IOs to finish.
      – Not IO latency time!
  – SIO+ASM: does not do parallel writes, but serial.
  – SIO+ASM: does not time IO.
Conclusion

• Logwriter:
  – Wait event IO timing with ASM:
    – All the ‘* parallel read’ and ‘* parallel write’ events do not seem to time IO correctly with synchronous IO*.
    – All the events which cover single block IOs do use synchronous IO calls, even with asynchronous IO set.
  – Logwriter writes a warning when IO time and SCN broadcast ack time exceeds 500ms in the log writer trace file.
  – _disable_logging *only* disables write to logs.
Conclusion

• Database writer:
  – When idle, is sleeping on a semaphore/rdbms ipc message
  – Gets posted with semctl() to do work.
  – Only writes when it needs to do so.
  – Since version 11.2.0.3, event ‘db file async I/O submit’:
    – Is not shown with synchronous I/O.
    – Shows the actual amount of IOs submitted.
    – Does not time io_submit() with ASM.
Conclusion

• Database writer:
  – Event ‘db file parallel write’:
    – Shows the minimal number io_getevents() waits for.
    – The number of requests it waits for varies, but mostly seems to be ~ 25-33% of submitted IOs.
    – After the timed, blocking, io_getevents() call, it issues one/two non-blocking io_getevents() calls for the remaining non-reaped IOs, if any.
    – My current idea is the blocking io_getevents() call is an IO throttle mechanism.
Conclusion

• Database writer:
  – Event ‘db file parallel write’, with synchronous IO and ASM:
    – pwrite64() calls are issued serially.
    – These are not timed up to version 12.1.0.1.
    – The event is triggered twice starting from version 11.2.0.3.

  – On exadata, up to a certain number of the total number of
    oss_wait() calls are timed with the event ‘db file parallel
    write’, depending on version.
Q & A
Thanks & Links

- Enkitec
- Tanel Poder, Martin Bach, Klaas-Jan Jongsma, Jeremy Schneider, Karl Arao, Michael Fontana, Luca Canali.