How To Do Impossible "Log File Sync" Performance Analysis With ASH Data

https://www.orapub.com/ppts

Craig Shallahamer
craig@orapub.com

ORACLE ACE Director

ORAPUB

COLLABORATE19
TECHNOLOGY AND APPLICATIONS FORUM FOR THE ORACLE COMMUNITY

IOUG

APRIL 7-11, 2019
SAN ANTONIO

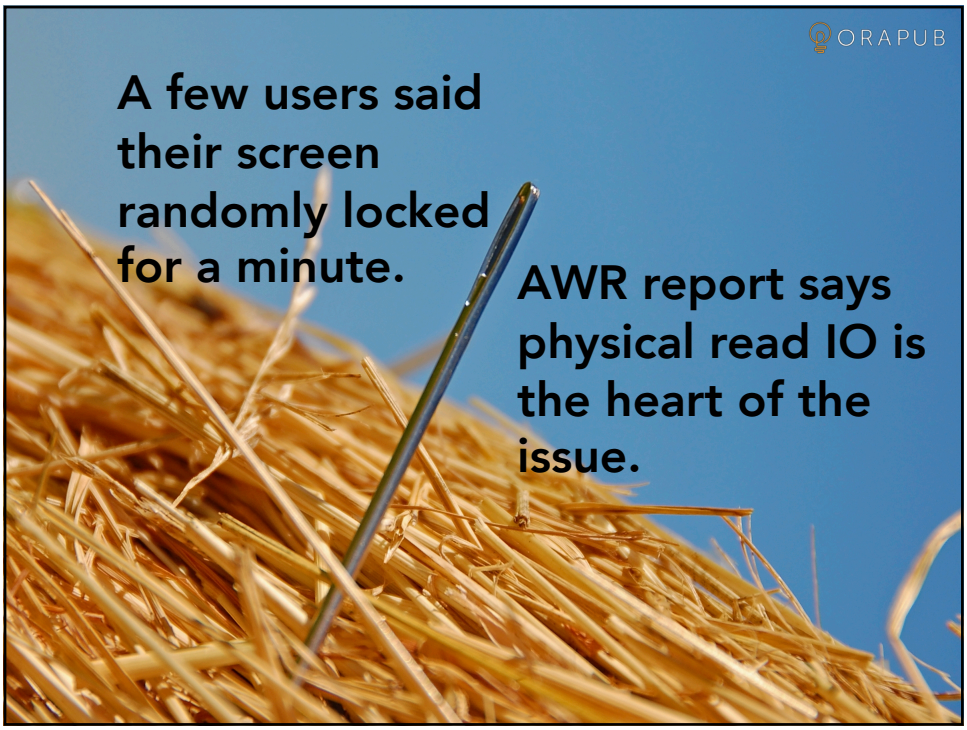
REGISTER NOW AT COLLABORATE.IOUG.ORG

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

For the most updated version of this presentation, go to:

www.orapub.com/ppts



This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

What SQL statements where involved in a deadlock between 9:47 and 10:00?

What is the top wait event between 9:10 and 9:20?

How many users where impacted by that incident?

Is there a program that is crazy active?

What top SQL for the time entry module?

Show me a timeline of the incident.

What is the most CPU intensive SQL_ID?

How many active sessions where there at 9:47?

Which session started the issue?

What is consuming all that CPU?

ORAPUB

But is there more?

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

But is there more?

Yes!

We can do the impossible
"log file sync" performance analysis
using ASH data

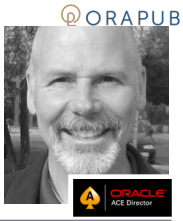
Here's our journey for today.

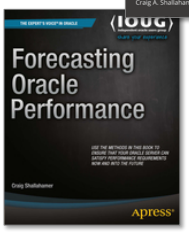
- What is a log file sync?
- Why is log file sync analysis so difficult?
- How do we diagnose log file sync incidents?
- How do we construct a story so people can understand our analysis and accept our solutions?

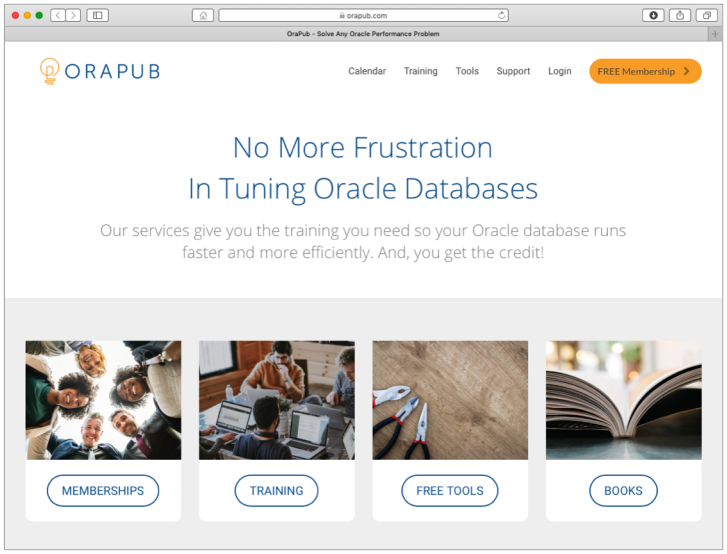
This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

About Me...

- Long time Oracle DBA
- Specialize in Oracle Database performance and performance engineering
- Performance researcher
- Blogger: A Wider View About Oracle Performance Tuning
- Author: Oracle Performance Firefighting and Forecasting Oracle Performance.
- Conference speaker
- Teacher and mentor
- Oracle ACE Director
- IOUG DBA Track Manager







OraPub works with Oracle DBAs empowering them to beat bots, AI, machine learning and autonomous anything.

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

Image the questions I can ask!

Each **piece of paper** is like an active Oracle session.
Each **stack** contains all the active sessions captures during a single ASH sample, such as at 15:25:06.

ORAPUB

What is a Log File Sync?

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

What is a Log File Sync?

LFS is the **wait time** associated with a commit.

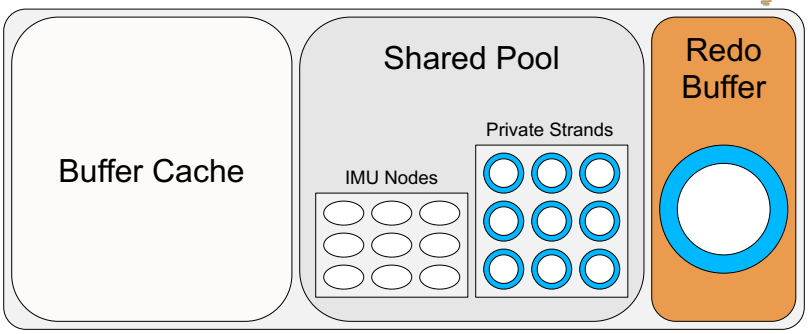
Keep in mind, there is also CPU time associated with a commit.

The log file sync wait event.

- Log file synchronization is about preparing for a commit and waiting for it to complete.
- For a commit to complete:
  - Both the server process and the LGWR are deeply involved.
  - They require CPU and IO resources to complete their tasks.
  - Their work must be synchronized to ensure everything is completed in the proper order.
  - When the foreground process is not consuming CPU but waiting on something else, probably the LGWR, it posts the `log file sync` event.
- Because significant CPU and IO resources are required to complete a commit, there can be **either** an operating system CPU or IO bottleneck.

Search Blog for "log file sync commit time"

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts



Multiple redo allocation latches allow multiple server processes to simultaneously copy redo into their private strand, also speeding redo into the redo buffer.

In 10gR2 "batch redo" was introduced. On **commit** (not just a change) both transaction IMU and redo (in a private strand) are copied into the redo general/central buffer and then the LGWR flushes the general/central redo buffer to the current on-line redo log.

When does the LGWR write?

- The LGWR writes when:
  - buffer fills to min(1/3 full, 1MB)
  - 3 second timeout (pre-10gR2)
  - Commit issued. Tests always show an immediate LGWR write, unless *commit write* facility is used.
  - DBWR tells LGWR to write when it wants to write to a dbf file. DBF file change history must be recorded in an on-line redo log, otherwise we won't know how the DBF file arrived at its current state.

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

LG00

```
$ strace -rp 37094
Process 37094 attached - interrupt to quit
     0.000000 clock_gettime(CLOCK_MONOTONIC, {18241951, 746042039}) = 0
     0.000055 gettimeofday({1400281130, 878476}, NULL) = 0
     0.000047 clock_gettime(CLOCK_MONOTONIC, {18241951, 746129707}) = 0
     0.000032 semtimedop(7307267, {{18, -1, 0}}, 1, {3, 0}) = 0
     0.054282 clock_gettime(CLOCK_MONOTONIC, {18241951, 800443453}) = 0
     0.000031 gettimeofday({1400281130, 932868}, NULL) = 0
     0.000031 gettimeofday({1400281130, 932899}, NULL) = 0
     0.000033 clock_gettime(CLOCK_MONOTONIC, {18241951, 800536977}) = 0
     0.000028 clock_gettime(CLOCK_MONOTONIC, {18241951, 800563872}) = 0
     0.000027 clock_gettime(CLOCK_MONOTONIC, {18241951, 800594344}) = 0
     0.000030 clock_gettime(CLOCK_MONOTONIC, {18241951, 800620822}) = 0
     0.000026 pwrite(256, "\1\"\0\0\32536\5\1\0\0\5"..., 1024, 10715648) = 1024
     0.007834 clock_gettime(CLOCK_MONOTONIC, {18241951, 808483126}) = 0
. . .
     0.000031 gettimeofday({1400281130, 941118}, NULL) = 0
     0.000030 semctl(7307267, 16, SETVAL, 0x7fff00000001) = 0
     0.000030 clock_gettime(CLOCK_MONOTONIC, {18241951, 808783134}) = 0
     0.000027 clock_gettime(CLOCK_MONOTONIC, {18241951, 808809667}) = 0
     0.000029 semtimedop(7307267, {{18, -1, 0}}, 1, {3, 0}) = 0
     0.035167 clock_gettime(CLOCK_MONOTONIC, {18241951, 844011841}) = 0
     0.000034 gettimeofday({1400281130, 976435}, NULL) = 0
. . .
     0.000028 clock_gettime(CLOCK_MONOTONIC, {18241951, 844186855}) = 0
     0.000031 pwrite(256, "\1\"\0\0\303Q\0\0\3120\0\0"..., 1536, 10716672) = 1536
     0.005887 clock_gettime(CLOCK_MONOTONIC, {18241951, 850103698}) = 0
. . .
```

o12.1

Sequence of events

ORAPUB

The key to solving the LFS mystery using ASH, is to understand the sequence of events. Then knowing the process state (cpu or wait) and understanding possible wait events.

```
sample task       sql_id  state    wait event
------ ---------- ------- -------- -----------
...
   100 dml
   101 dml
   102 dml
   103 commit
   104 commit
   105 commit
...
```

1 DML statement lasting multiple samples

1 long commit lasting multiple samples

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

Sequence of events

ORAPUB

There is no SQL_ID during the commit.

AND, there is no LFS wait event when consuming CPU.

```
sample task       sql_id  state    wait event
------ ---------- ------- -------- -----------
...
   100 dml        abc123  CPU      -
   101 dml        abc123  wait     db file sequential read
   102 dml        abc123  CPU      -
   103 commit     -       cpu      -
   104 commit     -       wait     log file sync
   105 commit     -       cpu      -
...
```

Sequence of events

ORAPUB

What if the log file sync wait is
NOT sampled by ASH or NOT saved in the AWR ASH data?

```
sample task       sql_id  state    wait event
------ ---------- ------- -------- -----------
...
   100 dml        abc123  CPU      -
   101 dml        abc123  wait     db file sequential read
   102 dml        abc123  CPU      -
   103 commit     -       cpu      -

   104 commit     -       cpu      -
...
```


This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

Sequence of events

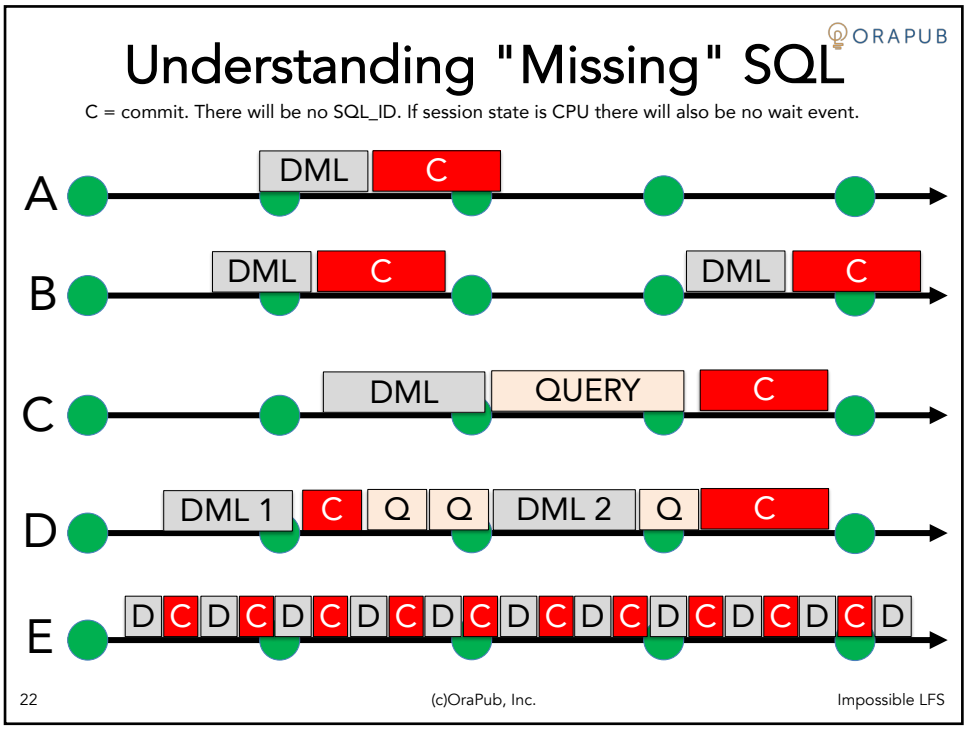
ORAPUB

- What if there is no task or application clues?
- Yes, we can get the type of SQL in most cases.
- What if the dml and/or commit occurred between samples?

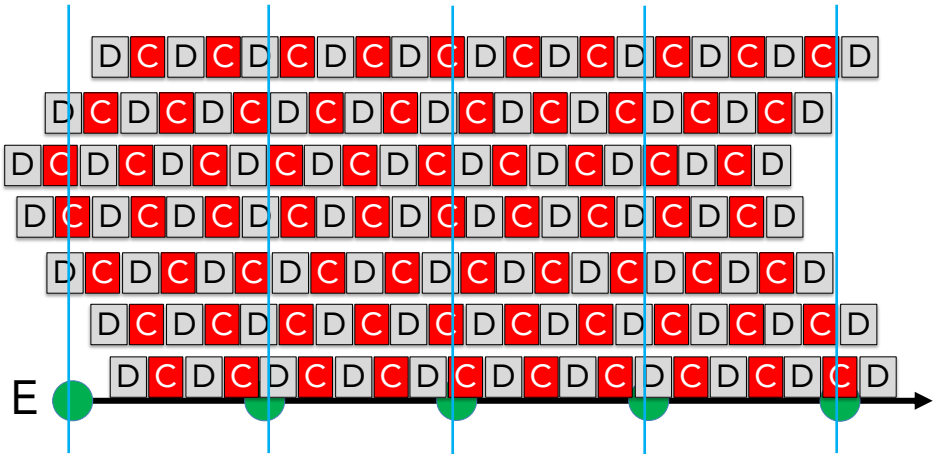
```
sample          sql_id  state    wait event
------          ------- -------- -----------
...
   100          abc123  CPU      -
   101          abc123  wait     db file sequential read
   102          abc123  CPU      -
   103          -       cpu      -

   104          -       cpu      -
...
```

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts



The Problem Of Rapid Commits.

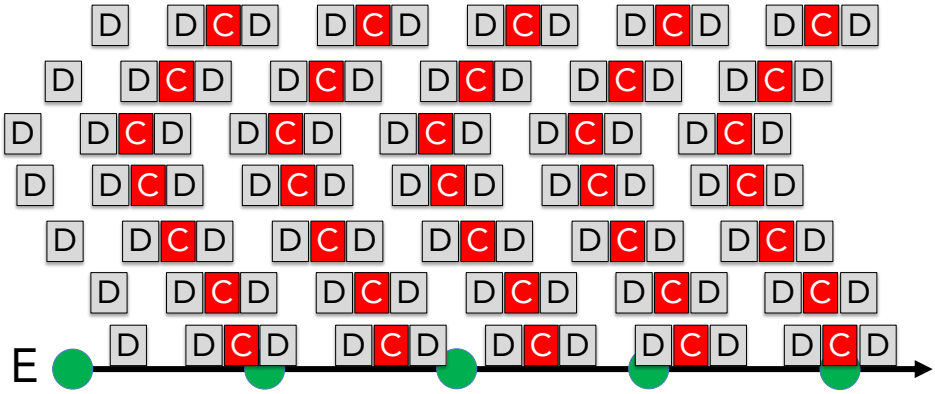


The more session doing rapid commits, the more likely we will see lots of LFS.

Search Blog for "batch commit"

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

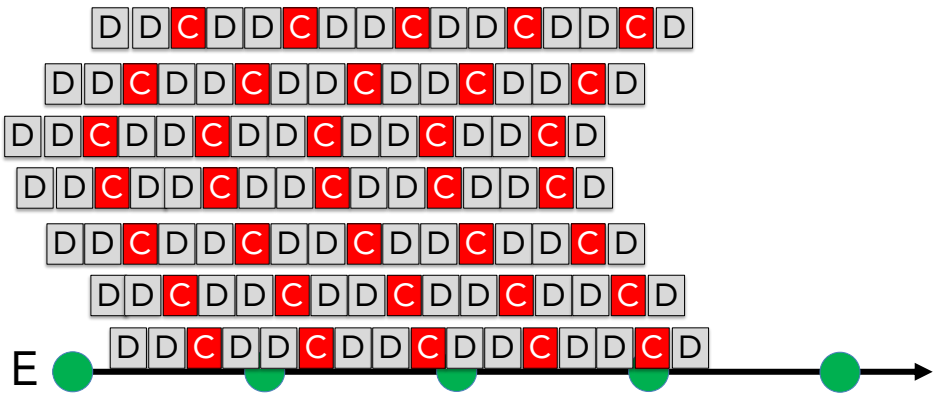
The Solution For Rapid Commits.



1. Less commits, means less commit time, means less db time for same DML work.
2. Grouping DML has a massive throughput gain.

Search Blog for "batch commit"

The Solution Rewards For Rapid Commits.

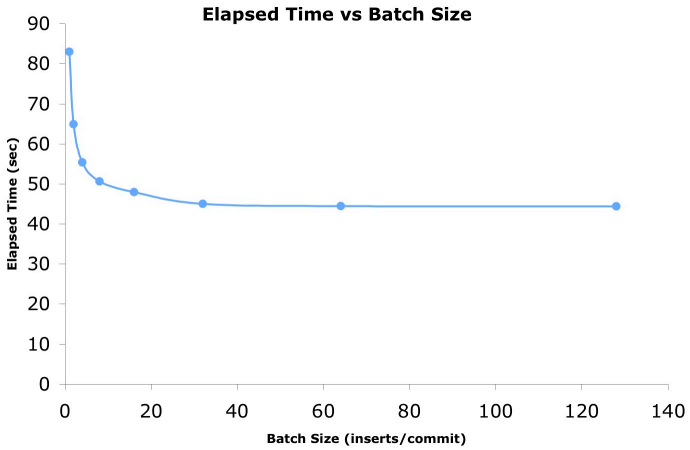


1. Less commits, means less commit time, means less db time for same DML work.
2. Grouping DML has a massive throughput gain.

Search Blog for "batch commit"

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

Experimental Results



Search Blog for "batch commit"

Solutions for LFS and LFPW.

ORAPUB

- Log file sync, posted by a server process.
  - Wait time is how long it takes a commit to complete plus perhaps 10% to 25% additional CPU time.
  - Classic cause is a rapidly committing application.
  - Common with *log file parallel write* waits.
  - The associated OS bottleneck can be either CPU or IO: lots to do in both areas.
  - Solutions:
    - First, find the rapid commit area of the application.
    - Don't be initially distracted by the IO subsystem.
    - Increase batch size.
    - Reduce commit frequency to reduce total "sync" activity.
    - Increase the LGWR OS priority, if CPU bottleneck.
    - Bind a CPU and the LGWR together exclusively.
    - Investigate using commit write facility; instance, session, or statement.
- Log file parallel write, posted by the LGWR.
  - This occurs when the LGWR waits…IO write response time from the IO subsystem.
  - Solution: Decrease redo requirements by application re/design, workload balancing, and specific Oracle features such as global temporary tables. Increase O/S I/O write capacity throughput for on-line redo logs.

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

Let's review and demonstrate this using real ASH data.

ORAPUB

LFS is a foreground process wait.

```
SQL> l
  1  select count(*), session_type
  2  from   op_dba_hist_ash_sample_data_54
  3  where  event = 'log file sync'
  4    and  session_state = 'WAITING'
  5  group by session_type
  6* order by 1
SQL> /

  COUNT(*) SESSION_TY
---------- ----------
         3 BACKGROUND
     18468 FOREGROUND
```


This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

What's the big deal? Get the SQL_ID when the event is log file sync!

```
SQL> l
  1  select count(*), event, sql_id
  2  from   op_dba_hist_ash_sample_data_54
  3  where  event = 'log file sync'
  4    and  session_state = 'WAITING'
  5  group by event,sql_id
  6* order by 1,2,3
SQL>
```

What's the big deal?
Get the SQL_ID when the event is log file sync!

ORAPUB

```
SQL> l
  1  select count(*), event, sql_id
  2  from   op_dba_hist_ash_sample_data_54
  3  where  event = 'log file sync'
  4    and  session_state = 'WAITING'
  5  group by event,sql_id
  6* order by 1,2,3
SQL> /

  COUNT(*) EVENT         SQL_ID
---------- ------------- -------------
         1 log file sync 5u1vn3arapvv0
         3 log file sync 4c0c8nqtt8t44
        26 log file sync cwv71b61gsvmc
        28 log file sync g75cv70da0db0
        29 log file sync 59vjj34vugaav
       237 log file sync dmzhrtz46pzpq
     18147 log file sync     ??????????????????

7 rows selected.
```


This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

Are blank SQL_ID's always related to foreground processes?

ORAPUB

```
SQL> l
  1  select count(*), session_type, sql_id
  2  from   op_dba_hist_ash_sample_data_54
  3  group by session_type, sql_id
  4* order by 1
SQL> /

  COUNT(*) SESSION_TY SQL_ID
---------- ---------- -------------
...
     94263 FOREGROUND 8x02x20hpbykt
    124173 FOREGROUND g0uu9jbkk79kv
    125295 BACKGROUND
    138104 FOREGROUND 9r48h7rkx61gn
    170942 BACKGROUND az4y0d5bu9t5n
    246937 FOREGROUND 50vdbkbzm4abm
    260960 FOREGROUND
    315054 BACKGROUND 04pxuq9t9a4wa
    325424 FOREGROUND d7wjhr5fpk5bs

21224 rows selected.
```

ORAPUB

Where in the application is log file sync occurring?

```
select count(*), event,
       sql_id, MODULE
from   op_dba_hist_ash_sample_data_54
where  event = 'log file sync'
  and  session_state = 'WAITING'
group by event, sql_id, MODULE
order by 1,2,3
```


This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

Where in the application is log file sync occurring?

```
  COUNT(*) EVENT         SQL_ID        MODULE
  -------- ------------- ------------- --------------------------
...
        29 log file sync 59vjj34vugaav DBMS_SCHEDULER
        65 log file sync
       237 log file sync dmzhrtz46pzpq
      1905 log file sync               httpd.worker@module010 (T
      2012 log file sync               httpd.worker@module011 (TN
      2533 log file sync               httpd.worker@module005 (TN
      3201 log file sync               httpd.worker@module015 (TN
      4166 log file sync               httpd.worker@module014 (TN
      4243 log file sync               httpd.worker@module013 (TN

22 rows selected.
```

ORAPUB

Where in the application is log file sync occurring?

```
select count(*), event,
       sql_id, PROGRAM
from   op_dba_hist_ash_sample_data_54
where  event = 'log file sync'
  and  session_state = 'WAITING'
group by event, sql_id, PROGRAM
order by 1,2,3
```


This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

Where in the application is log file sync occurring?

```
  COUNT(*) EVENT         SQL_ID        PROGRAM
---------- ------------- ------------- ------------------
...
        69 log file sync dmzhrtz46pzpq oracle@zippi01 (J002
        71 log file sync dmzhrtz46pzpq oracle@zippi01 (J001
      1905 log file sync               httpd.worker@zippi08
      2012 log file sync               httpd.worker@zippi09
      2533 log file sync               httpd.worker@zippi03
      3201 log file sync               httpd.worker@zippi13
      4166 log file sync               httpd.worker@zippi12
      4243 log file sync               httpd.worker@zippi11

51 rows selected.
```

SQL Operation Names

```
SQL> l
  1  select distinct sql_opname, sql_opcode
  2  from   op_dba_hist_ash_sample_data_54
  3* order by 1
SQL> /

SQL_OPNAME           SQL_OPCODE
-------------------- ----------
ALTER DATABASE               35
ALTER INDEX                  11
ALTER PACKAGE                95
ALTER PACKAGE BODY           98
ALTER PROCEDURE              25
ALTER SYNONYM               192
ALTER TABLE                  15
ALTER TRIGGER                60
ALTER VIEW                   88
ANALYZE INDEX                63
CALL METHOD                 170
```

```
SQL_OPNAME           SQL_OPCODE
-------------------- ----------
CREATE INDEX                  9
CREATE PACKAGE               94
CREATE PACKAGE BODY          97
CREATE PROCEDURE             24
CREATE SEQUENCE              13
CREATE TABLE                  1
CREATE TRIGGER               59
CREATE VIEW                  21
DELETE                        7
DROP PACKAGE                 96
DROP TABLE                   12
DROP TYPE                    78
EXPLAIN                      50
INSERT                        2
LOCK TABLE                   26
PL/SQL EXECUTE               47
SELECT                        3
TRUNCATE TABLE               85
UPDATE                        6
UPSERT                      189
                              0

32 rows selected.
```


This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

Step-By-Step LFS Analysis Using ASH Data

- Understand commit is either consuming CPU or waiting on a log file sync wait event.
- Understand the "blank SQL_ID".
- Find a session, module or time of interest.
- Timeline the session(s):
  - With the serial number, showing the sql_id, op_name, module and/or program, execution start time and execution ID.
  - Be aware, there could also be other clues to where the issue is occurring.
- Find the log file sync in the timeline.
- Work backward, line by line, paying close attention to the sample time gaps, the op_name and the module.
- Remember that SQL and commits can complete extremely fast, so you may not see the actual incident or the incident detail. Why? Because ASH is sample based at a 1 second (v$) or 10 second (AWR) frequency.

"Timeline" all sessions related to key module and program.

```
def datasource=op_dba_hist_ash_sample_data_54
def dbahistdetails=' and dbid=3340869378 and instance_number=1'
def timingdetails="1=1"

def ashCols="event,SQL_ID,sql_opname,module,session_serial#"

def ashWhere="module like 'httpd.worker@module013%' and
              program like 'httpd.worker@zippi11%'"

select sample_id, to_char(sample_time,'DD HH24:MI:SS') DT,
       session_id sid, session_state,
       &ashCols
from   &datasource
where  &timingdetails
       &dbahistdetails
  and  &ashWhere
order by 1,3
/
```

"Timeline" ASH Scratchpad Script

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

SAMPLE_ID	DT	SID	SESSION	EVENT	SQL_ID	SQL_OPNAME	SERIAL#
60708902	05 18:50:22	2910	ON CPU		bcfz5cwrwy1fz	SELECT	20687
60708982	05 18:51:42	1028	WAITING	db file sequential read	fn5d0xhkyf3ap	SELECT	20053
60709102	05 18:53:43	931	WAITING	SQL*Net break/reset to cl	5892yqfmhkcdx	PL/SQL EXE	27447
60709312	05 18:57:14	931	WAITING	SQL*Net vector data to cl			27447
60709452	05 18:59:34	1196	ON CPU		9dcbm5ufmrccb	SELECT	18299
60709582	05 19:01:45	49	WAITING	db file sequential read	gq0pvxjtwwfx6	SELECT	1823
60709582	05 19:01:45	1028	WAITING	db file sequential read	gb8puygdu4hmn	INSERT	20053
60709602	05 19:02:05	1028	WAITING	db file sequential read	3vdra9wcmc8jn	INSERT	20053
60709612	05 19:02:15	1028	WAITING	SQL*Net vector data to cl			20053
60709622	05 19:02:25	1028	WAITING	db file sequential read	av61pb6qw72mg	SELECT	20053
60709632	05 19:02:35	1028	WAITING	db file sequential read	2kkq3zx6smajc	SELECT	20053
60709632	05 19:02:35	1437	WAITING	db file sequential read	dj424snf46dsq	SELECT	23561
60709652	05 19:02:55	49	WAITING	log file sync			1823
60709652	05 19:02:55	191	WAITING	log file sync			235
60709652	05 19:02:55	529	WAITING	log file sync			13127
60709652	05 19:02:55	1028	WAITING	log file sync			20053
60709672	05 19:03:15	1028	WAITING	db file sequential read	554w3nds3gzhv	SELECT	20053
60709692	05 19:03:35	529	ON CPU		3q98tvp0grj04	SELECT	13127
60709702	05 19:03:45	49	WAITING	log file sync			1823
60709702	05 19:03:45	529	WAITING	db file sequential read	avhg9a4162vwr	SELECT	13127
60709702	05 19:03:45	1028	WAITING	log file sync			20053
60709702	05 19:03:45	1196	WAITING	db file sequential read	55d61x1nq4vcj	SELECT	18299
60709712	05 19:03:55	49	WAITING	read by other session	554w3nds3gzhv	SELECT	1823
60709712	05 19:03:55	1028	WAITING	db file sequential read	554w3nds3gzhv	SELECT	20053
60709722	05 19:04:05	529	ON CPU		fxtjjmx55up8w	SELECT	13127
60709772	05 19:04:55	49	WAITING	log file sync			1823
60709772	05 19:04:55	266	WAITING	log file sync			23749
60709772	05 19:04:55	529	WAITING	log file sync			13127
60709782	05 19:05:05	191	WAITING	db file sequential read	a5rqs34fwaprj	SELECT	235
60709812	05 19:05:35	529	WAITING	SQL*Net vector data to cl			13127
60709822	05 19:05:45	1279	WAITING	log file sync			47955
60709842	05 19:06:05	1279	WAITING	Disk file operations I/O	f1thxv88g6dmm	INSERT	47955
60709882	05 19:06:46	1279	ON CPU		b9jnz00wywtmj	SELECT	47955
60709912	05 19:07:16	49	WAITING	db file sequential read	6n6m92z4szwuh	SELECT	1823

LFS "stacking"

ORAPUB

Let's "timeline" a session that has lots of LFS waits and is running the key module and key program

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

Timeline session 1028/20053

```
SAMPLE_ID DT            SID SESSION EVENT                      SQL_ID        SQL_OPNAME TO_CHAR
---------- ----------- ----- ------- ------------------------- ------------- ---------- ------
  60708512 05 18:43:51  1028 ON CPU                            3q98tvp0grj04 SELECT     43:50-
  60708982 05 18:51:42  1028 WAITING db file sequential read   fn5d0xhkyf3ap SELECT     51:41-
  60709582 05 19:01:45  1028 WAITING db file sequential read   gb8puygdu4hmn INSERT     01:44-
  60709602 05 19:02:05  1028 WAITING db file sequential read   3vdra9wcmc8jn INSERT     02:04-
  60709612 05 19:02:15  1028 WAITING SQL*Net vector data to cl                            -
  60709622 05 19:02:25  1028 WAITING db file sequential read   av61pb6qw72mg SELECT     02:24-
  60709632 05 19:02:35  1028 WAITING db file sequential read   2kkq3zx6smajc SELECT     02:31-
  60709652 05 19:02:55  1028 WAITING log file sync                                        -
  60709672 05 19:03:15  1028 WAITING db file sequential read   554w3nds3gzhv SELECT     03:14-
  60709702 05 19:03:45  1028 WAITING log file sync                                        -
  60709712 05 19:03:55  1028 WAITING db file sequential read   554w3nds3gzhv SELECT     03:52-

11 rows selected.
```

```
SQL> def ashCols="event,SQL_ID,sql_opname,to_char(sql_exec_start,'MI:SS'
SQL> def ashWhere="session_id=1028 and session_serial#=20053"
```

The module and program are:

```
httpd.worker@module013 (TNS V1
httpd.worker@zippi11 (TNS V1-V
```

Timeline session 2235/91

ORAPUB

SAMPLE_ID	DT	SID	SESSION	EVENT	SQL_ID	SQL_OPNAME	TO_CHAR(SQL_EXEC_S
62920780	31 12:39:33	2235	WAITING	db file sequential read	5x5fz9fc91j99	SELECT	39:28-16781023
62920790	31 12:39:43	2235	WAITING	log file sync			-
62920800	31 12:39:54	2235	WAITING	log file sync			-
62920810	31 12:40:04	2235	WAITING	db file sequential read	fvgqj6rd5pht7	SELECT	40:03-16777889
62920820	31 12:40:14	2235	WAITING	db file sequential read	6n6m92z4szwuh	SELECT	40:08-16777890
62920860	31 12:40:54	2235	ON CPU		c937pbscnfv06	INSERT	40:51-16777227
62920870	31 12:41:04	2235	ON CPU		11cfcd5n3w9bk	PL/SQL EXE	41:04-16882727
62920890	31 12:41:24	2235	WAITING	db file sequential read	11u3xp8y4y1gu	INSERT	41:23-16778995
62920910	31 12:41:44	2235	WAITING	db file sequential read	a3cw3qb200kmk	INSERT	41:39-16779678
62920920	31 12:41:54	2235	WAITING	log file sync			-
62920970	31 12:42:45	2235	WAITING	db file sequential read	7yrtt58uuahzr	SELECT	42:40-16780822
62920980	31 12:42:55	2235	WAITING	TCP Socket (KGAS)	50vdbkbzm4abm	PL/SQL EXE	42:54-16777245
62920990	31 12:43:05	2235	WAITING	db file sequential read	9acjr0ftq10g4	PL/SQL EXE	42:59-16777216
62921000	31 12:43:15	2235	WAITING	db file sequential read	2azkyyck47pf7	SELECT	43:08-16777219
62921010	31 12:43:25	2235	WAITING	db file sequential read	1j3gnfv93u2zr	SELECT	43:25-16777682
62921090	31 12:44:46	2235	WAITING	log file sync			-
62921110	31 12:45:06	2235	WAITING	db file sequential read	f1thxv88g6dmm	INSERT	45:05-16783373
62921120	31 12:45:16	2235	WAITING	db file sequential read	6xy8fhm4u0d5r	SELECT	45:10-16780006
62921130	31 12:45:26	2235	ON CPU		6xy8fhm4u0d5r	SELECT	45:10-16780006
62921140	31 12:45:36	2235	WAITING	read by other session	6xy8fhm4u0d5r	SELECT	45:10-16780006
62921150	31 12:45:46	2235	ON CPU		a72cx3rwzfdnm	SELECT	45:37-16777542
62921180	31 12:46:16	2235	WAITING	cursor: pin S wait on X	0k60vyv9whhfp	SELECT	-
62921190	31 12:46:26	2235	WAITING	log file sync			-
62921200	31 12:46:36	2235	ON CPU		ftwyc7w2h1r02	INSERT	46:30-16777420
62921230	31 12:47:07	2235	WAITING	latch: redo writing	f1thxv88g6dmm	INSERT	46:56-16783905
62921250	31 12:47:27	2235	WAITING	read by other session	5uvjagrh9zrwv	SELECT	47:25-16777594
62921260	31 12:47:37	2235	WAITING	db file sequential read	fvgqj6rd5pht7	SELECT	47:33-16778239
62921270	31 12:47:47	2235	WAITING	db file sequential read	6xbxd3qgyrraw	SELECT	47:46-16777878
62921310	31 12:48:27	2235	WAITING	log file sync			-
62921320	31 12:48:37	2235	ON CPU		241sna55f1dbk	PL/SQL EXE	48:36-16778793
62921350	31 12:49:07	2235	WAITING	db file sequential read	c4ug00unmjv6x	INSERT	49:05-16777288
62921390	31 12:49:48	2235	WAITING	log file sync			-

32 rows selected.

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

Timeline session 1700/95

ORAPUB

SAMPLE_ID	DT	SID	SESSION	EVENT	SQL_ID	SQL_OPNAME	MODULE
62920960	31 12:42:35	1700	WAITING	read by other session	7yst1v6dqc2sx	DELETE	httpd.wo
62920970	31 12:42:45	1700	WAITING	enq: RC - Result Cache: C	gwxv7b8unzv58	PL/SQL EXE	httpd.wo
62921000	31 12:43:15	1700	WAITING	log file sync			httpd.wo
62921030	31 12:43:45	1700	WAITING	log file sync			httpd.wo
62921090	31 12:44:46	1700	WAITING	db file sequential read	3vdra9wcmc8jn	INSERT	httpd.wo
62921100	31 12:44:56	1700	WAITING	db file sequential read	f1thxv88g6dmm	INSERT	httpd.wo
62921120	31 12:45:16	1700	WAITING	log file sync			httpd.wo
62921150	31 12:45:46	1700	WAITING	log file sync			httpd.wo
62921190	31 12:46:26	1700	WAITING	log file sync			httpd.wo
62921200	31 12:46:36	1700	ON CPU		4avn39hxnfajw	SELECT	httpd.wo
62921230	31 12:47:07	1700	WAITING	latch: redo writing	a3cw3qb200kmk	INSERT	httpd.wo
62921240	31 12:47:17	1700	ON CPU		1brf7pa4xja6x	SELECT	httpd.wo
62921250	31 12:47:27	1700	WAITING	log file sync			httpd.wo
62921310	31 12:48:27	1700	ON CPU		4xz29t6wfxxu1	PL/SQL EXE	httpd.wo
62921330	31 12:48:47	1700	WAITING	db file sequential read	6v28h5d08pfky	UPDATE	httpd.wo

15 rows selected.

ORAPUB

With all these challenges, "*Timeline*ing" a session provides many details we need.

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

What I learned…

- Know **when** to look.
  - User, log, lots of LFS by hour by [session, program, client_id, module, etc.]
- Discover **where** the LFSs are occurring.
  - module, program, SQL_ID, etc.
- Timeline "**all sessions**" at key time(s) for the key module, program, etc.
  - This is useful to understand the application flow.
  - LFSs tend to "stack up" bursting with waits, which we can't see when focused on one session.
- Timeline **key** session(s).
  - This is useful to find the DML SQL, long running or multiple execution SQL and where in the application (think: module, program) the LFSs are occurring.
  - You want to find **multiple** situations confirming the "story."
- Never forget we are looking at sample data so, **expect missing details**.

ORAPUB

So awesome!

However…

This is just the beginning.

There is so much more we can do with ASH analysis.

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB

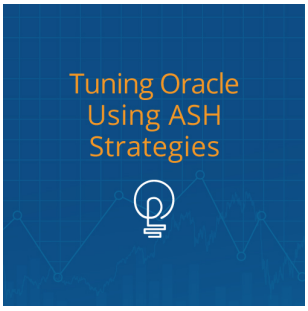
ASH Scratch Pad

www.orapub.com/tools

www.orapub.com >
Login >
TOOLS & PRESENTATIONS >
Tolls

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

ORAPUB



OraPub LVCs Are Different

You learn to master the topic at a deeper, higher confidence and more practical level compared to any other teaching method I offer.

This occurs because the class is spread out over multiple weeks, each session is 2 hours with a day in between, you do homework/activation on your real systems and I personally work with you through the entire class.

Details & Registration: https://www.orapub.com/lvc
Event Calendar: https://www.orapub.com/events

ORAPUB

Thank You!

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts

How To Do Impossible "Log File Sync" Performance Analysis With ASH Data

https://www.orapub.com/ppts

Craig Shallahamer
craig@orapub.com

ORACLE ACE Director

ORAPUB

COLLABORATE19
TECHNOLOGY AND APPLICATIONS FORUM FOR THE ORACLE COMMUNITY

IOUG

APRIL 7-11, 2019
SAN ANTONIO

REGISTER NOW AT COLLABORATE.IOUG.ORG

ORAPUB

For the most updated version of this presentation, go to:

www.orapub.com/ppts

This presentation was given by Craig Shallahamer (craig@orapub.com) at the April 2019 Collaborate/IOUG Conference in San Antonio, TX USA.
There is likely a more recent version at www.orapub.com/ppts