Tuning Redo Log Buffer

Before Tuning Log Buffer, let's understand few basic things, I have firm this in Question and Answer form.

What are Log Buffer and what type of information resides in Log buffer?

Log buffer is memory area in SGA. The log buffer in the SGA is internally divided into blocks of the log block size. The size of log buffer is specified by the log_buffer init parameter.

Log buffer contains Redo entries. Redo entry is made up group of change vector. For example if you change salary in employee table, redo entry contains old and new entry of particular change vector.

What if the log buffer is too small?

If the log buffer is too small, then log buffer space waits will be seen during high redo generation. And LGWR may not begin to write redo entries to redo log file until it reaches the threshold. This may cause serious performance problem.
Ideally, the log buffer should be large enough to accommodate high redo generation.

What are the conditions where the log buffer is flushed?

- A session issues a commit or a rollback command.
- The log buffer becomes 1/3 full.
- A timeout (every 3 seconds) occurs.
- A checkpoint occurs.

How Oracle allocates space in the Redo Log Buffer?

- The session will acquire the redo allocation latch.
- The session will allocate the memory it needs from the redo log buffer for the copy of the redo.
- The redo allocation latch is released and a redo copy latch is acquired, if available. If there is not a redo copy latch available, then the redo allocation latch is held until the end of the operation.
- The redo is copied to the redo log buffer.
- The redo copy latch is released.

How oracle calculate Log buffer size?
The LOG_BUFFER size will be set by default, by Oracle internal algorithm.

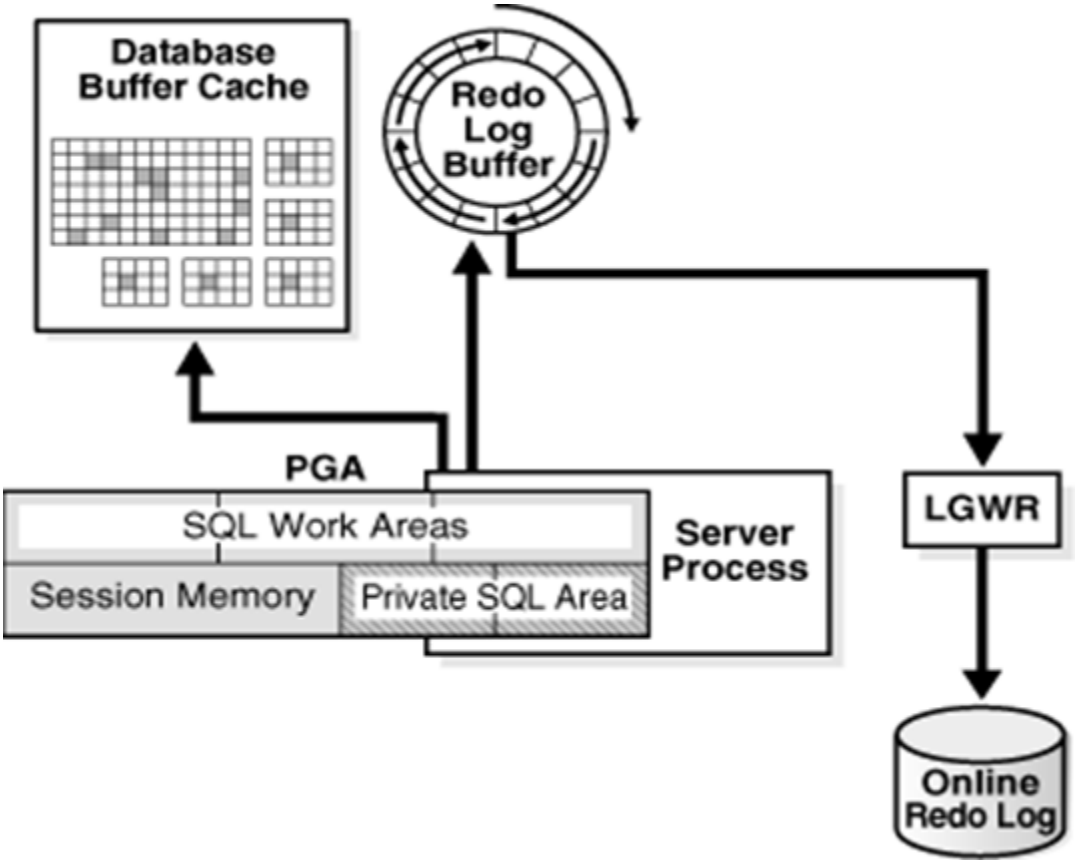
DB Version	Default Value	Calculation
9i & 10gR2	512 KB or 128 KB * CPU_COUNT, whichever is greater	Default is 512 KB or 128 KB * CPU_COUNT

11gR2 and 12c onwards	5 MB to 32 MB, depending on the size of the SGA, CPU count, and whether the operating system is 32-bit or 64-bit	If ("log_buffer" is NOT defined by the user)
		log_buffer = max(2M, 128K * ncpus) * max(2, ncpus/16);
		The default min log buffer size = 4M;
		Else
		log_buffer = max(user-specified value, 2M);

In both the cases value will be rounded, based on the SGA granule size. Also with high cpu_count, the calculated value becomes large and it is advisable to manually set log_buffer to the desired value instead of using the defaults.
In such cases the log_buffer can be calculated to be a much larger value than 36MB.

Redo log Latches (Redo wait events)

When a change to a data block needs to be done, it requires to create a redo record in the redolog buffer executing the following steps:



'- Generated a higher SCN
'- Locate space to write the redo record. If there is no space available then the LGWR must write to disk or issue a log switch.
'- Allocate space in redo log buffer.
'- Write/Copy the redo record in log buffer and link it to the appropriate structures for recovery purposes.

The database has three main wait events for Redo Or we can say redo latches to handle above process:

latch: redo copy (Redo Copy latch)
The redo copy latch is acquired all the time of above process.
In 9i, 10g & 11g, init.ora LOG_SIMULTANEOUS_COPIES determines the number of redo copy latches. This latch released when a log switch happen to release free space and re-acquired once the log switch ends.
In 12c, LOG_SIMULTANEOUS_COPIES parameter comes under, underscore parameter i.e. _LOG_SIMULTANEOUS_COPIES

latch: redo allocation (Redo_allocation latch)

Before Oracle9.2, the redo allocation latch is unique and thus serializes the writing of entries to the log buffer cache of the SGA.
In Oracle 9.2., the number of redo allocation latches is determined by init.ora LOG_PARALLELISM.
In 12c onwards, LOG_PARALLELISM comes under, underscore parameter i.e. _LOG_PARALLELISM_MAX

Redo writing latch

This latch prevents multiple log switch request of LGWR simultaneously. A process which needs free space will acquire the latch before of deciding whether to post the LGWR to perform a write, execute a log switch or just wait.

How to monitor Redo wait events?

You can use following query to monitor redo wait events.

```
select  c.name,  a.gets,  a.misses,  a.sleeps,  a.immediate_gets imm_gets,  a.immediate_misses imm_miss,  b.pid
from  v$latch a, v$latchholder b, v$latchname  c
where a.addr = b.laddr(+)
and a.latch# = c.latch#
and c.name like '%redo%'
order by a.latch#;
```

Sample output,

```
SQL> select
  2     c.name,   a.gets,   a.misses,   a.sleeps,  a.immediate_gets imm_gets,  a.immediate_misses imm_miss,   b.pid
  3  from
  4     v$latch       a, v$latchholder b, v$latchname   c
  5  where
  6  a.addr   = b.laddr(+)
  7  and a.latch# = c.latch#
  8  and c.name like '%redo%'
  9  order by a.latch#;

NAME                                                             GETS     MISSES     SLEEPS   IMM_GETS   IMM_MISS        PID
---------------------------------------------------------------- ---------- ---------- ---------- ---------- ---------- ----------
redo on-disk SCN                                                    0          0          0          0          0
ping redo on-disk SCN                                               0          0          0          0          0
instance redo on-disk SCN                                           0          0          0          0          0
redo transport task                                             18113          0          0          0          0
redo writing                                                 93889527     228688      47496          0          0
redo copy                                                       9905       9596      10354  744523012    9219195
redo allocation                                              204971928   10137907     500961  740611933   36881768
real redo SCN                                                       0          0          0          0          0
redo gen encryption key structure                                  22          0          0          0          0
readable standby metadata redo cache                                0          0          0          0          0
readredo stats and histogram                                        0          0          0          0          0
KFR redo allocation latch                                           0          0          0          0          0

12 rows selected.
```

If the ratio of MISSES to GETS exceeds 1%, or the ratio of IMMEDIATE_MISSES to (IMMEDIATE_GETS + IMMEDIATE_MISSES) exceeds 1%, there is latch contention.

Redo log space request

```
SQL> SELECT name, value
   FROM v$sysstat
   WHERE name = 'redo log space requests';  2    3

NAME                                                             VALUE
---------------------------------------------------------------- ----------
redo log space requests                                          16215
```

This statistic "redo log space requests" shows the number of times a user process waits for space in the redo log file, not the buffer space.
The value of "redo log space requests" should be near 0. If this value increments consistently, processes have had to wait for space in the buffer. This may be caused the checkpointing or log switching. Improve thus the checkpointing or archiving process.

What are Redo Log Buffer and Latches Tuning Parameters?

Following parameters you can see in Oracle database 12c.

Parameter Name	Description	Remarks
LOG_BUFFER	Redo log buffers	Default 2mb to 32mb
LOG_CHECKPOINT_INTERVAL	This specifies the frequency of checkpoints in terms of the number of redo log file blocks that can exist between an incremental checkpoint and the last block written to the redo log. This number refers to physical OS blocks.	Default value 0
LOG_CHECKPOINT_TIMEOUT	This specifies time in second that has passed since the incremental checkpoint and last written of redo log. This parameter also signifies that no buffer will remain dirty (in the cache) for more than specified time.	Default value 1800
FAST_START_MTTR_TARGET	Number of seconds the database takes to perform crash recovery	Default value 0

LOG_CHECKPOINTS_TO_ALERT	Log your checkpoints to the alert file.	Default value false
_log_simultaneous_copies	number of simultaneous copies into redo buffer (# of copy latches)	Default Value CPU_COUNT*2
_log_parallelism_max	Maximum number of log buffer strands	Default 4
_log_switch_timeout	Maximum number of seconds redos in the current log could span	Default 0

When and How to Tune Log Buffer Parameters?

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Avg Wait	% DB time	Wait Class
DB CPU		6094.6		41.9	
log file sync	1,442,462	2219.9	1.54ms	15.3	Commit
latch: redo allocation	523,900	2084.5	3.98ms	14.3	Other
latch: redo copy	196,140	963.7	4.91ms	6.6	Configuration
buffer busy waits	659,417	808.7	1.23ms	5.6	Concurrency
cursor: mutex X	49,183	422.4	8.59ms	2.9	Concurrency
SQL*Net more data from client	754,078	253.4	336.03us	1.7	Network
latch: ges resource hash list	71,578	114.3	1.60ms	.8	Other
latch: enqueue hash chains	63,993	102.4	1.60ms	.7	Other
latch free	23,210	25.9	1.11ms	.2	Other

Latch: redo copy

Causes

latch is acquired whole duration until log switch happens

Solution

Check log_buffer parameter value and also check _log_simultaneous_copies parameter (value should be cpu_count*2).

Latch: redo allocation

Causes

latch is acquired to allocate memory space in redo log buffer.

Solution

Check log_buffer parameter value and also check _log_parallelism_maxparameter .

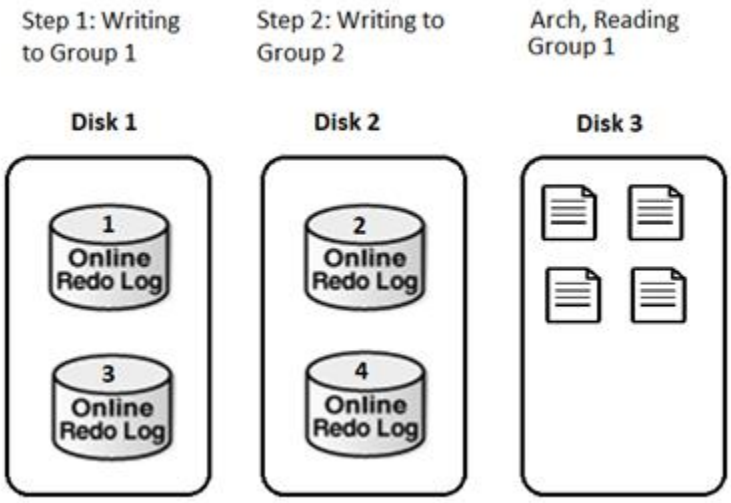
Log file sync
When you see log contention you will see log file sync wait event.
Causes

- Lower disk IO
- Small Redo log files
- Excessive application commits

Solution

- Put Redo log files in separate disk with high IO bandwidth.
- Proper sizing of redo log files.
- Application analysis to reduce commits (doing transactional based commits) or doing batch commits.

Putting Redo Logs in such a way to reduce contention in Online Redo Log Files.
Disk 1 will have odd numbers and Disk 2 will have even number. So that when LGWR is writing to Group 1, Group 2 will be archiving by ARCH.



This approach, you can ensure that Writing will happening on Disk 1 and Reading will happening in Disk2. So both the operations will be done on different disk by this way you can reduce contention.