



Redo Log In-depth

Gitesh Trivedi

Only for experienced Oracle DBA

NOT for Fresher DBA

(From Advance Performance Tuning Courseware)

Redo Log In-depth:

Objective: Depth study of Redo log and rectifying for tuning

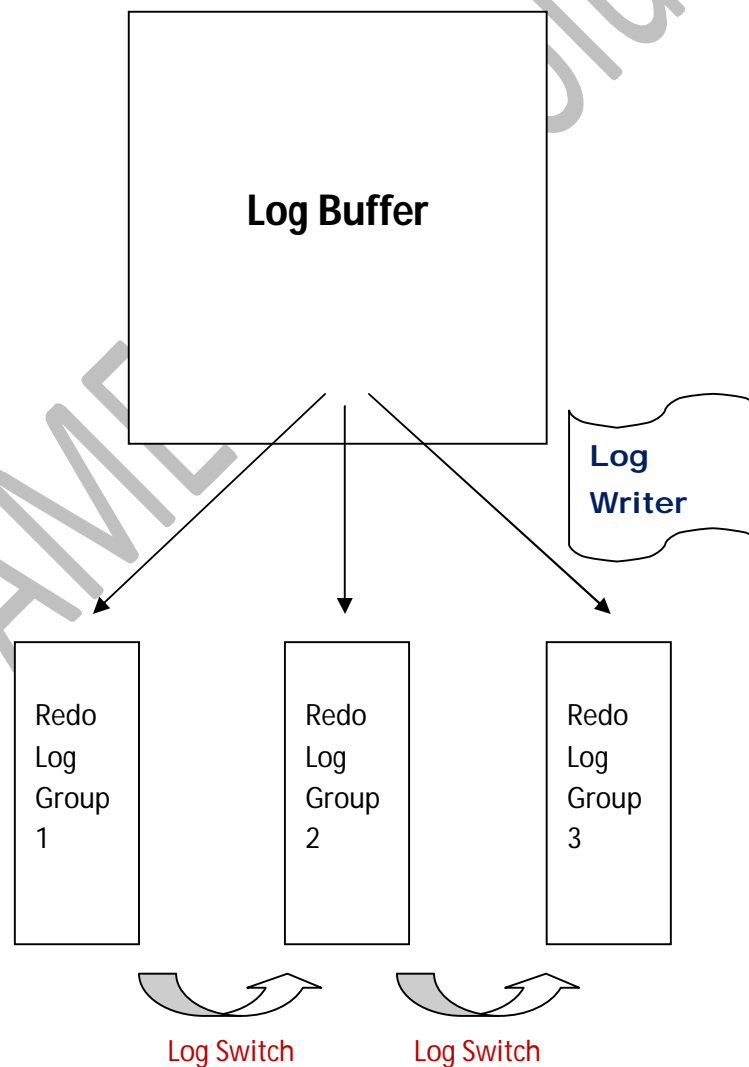
General Mechanism

Log Buffer:

Log buffer is part of Shared Global Area and populated in memory region of SGA. It is sub-component of Shared pool in instance object of Oracle Software.

Size of Redo log depends upon operating system physical block size.

Mechanism of Redo log Buffer:



Each and every DML and DDL statements writes in redo log buffers. And after committing transaction, all entries go in to redo log file. This writing process is done by Log writer back ground process in short it called as LGWR.

Select for update statement also generates redo change entries in redo.

As DBA you are aware of this scenario and knowing concept of redo log and archive processes.

What is Redo?

Keeping track and capture history of changes made in database.

Usage of Redo:

In Instance recovery

In Media recovery

Using Log Miner utility for auditing or recovery

Using Oracle Streams utility

Contains of Redo:

Redo record header

Thread - Thread number

RBA - Redo Byte Address

LEN - Length of Change Number

SCN - System Change Number, date and Time

What is RBA?

Called as Redo Byte Address. Consuming approximate 10 bytes. It represents/identifies start of redo record. RBA contains Log Sequence Number (4 bytes), Block Number within Redo Log (4 bytes) and Byte number within block (2 bytes) in started point.

There are different types of RBA available in SGA those are following.

Low RBA

Dirty buffer contains first redo change address called Low RBA. From x\$bh we can check low RBA.

High RBA

Dirty buffer contains last and most recent redo changes address called High RBA. From x\$bh we can check High RBA.

Checkpoint RBA

DBWR has written buffers from checkpoint queue are pointing to checkpoint RBA while incremental checkpoint is enabled. This RBA copies in to control file's checkpoint progress record. When instance recovery occurs that time it starts from checkpointing RBA from control file. We can check this RBA from x\$targetrba (sometimes from x\$kcrt).

Target RBA

It is representing RBA which DBWR want to done advance checkpoint for instance recovery (if enabled). We can check from x\$targetrba.

On-disk RBA

That RBA which was flushed in to online Redo Log File on disk. This RBA recorded in to control file record section. We can check from x\$kcpc for on-disk RBA (sometimes from x\$targetrba).

How Log Buffer writes:

While process goes to write in Log buffer, it takes redo copy latch. This latch protects log buffer from LGWR for flushing data from log buffer to redo log file. LGWR should wait up to copying be finished in Log Buffer. This latch protects Log buffers to flushing and I/O burden.

After getting redo copy latch, process must be taken allocation space in the log buffer. This redo allocation latch protects SGA heaps which are used and free allocation in redo log buffer.

Once getting redo allocation latch, process releases immediately latch and start writes change vectors from PGA or temporary buffer in to Redo Log Buffers.

After finishing copy of change vectors in to log buffers, all change vectors will be applied in to respective database blocks. All redo entries will be marked as VALID and redo copy latch will be released. If LGWR is waiting for complete redo copy in to log buffer then after finishing and releasing redo copy latch, process posts signal to LGWR to perform its task. This time process takes redo writing latch for checking LGWR performing task and active or not.

Detail explanation of Logwriter process:

While LGWR wakes up for writing data from log buffer to redo log file, It takes first redo writing latch and updates SGA variables for posting LGWR is active and checks used and free space allocation from SGA variables. After that LGWR takes redo allocation latch for checking log buffer is being written by foreground process or not (means any process is active with redo copy latch). If it finds any incomplete

writing process or any active process then LGWR sleeps for specific period. Wait event increased named "LGWR wait for redo copy". If it finds no active process in the log buffer then flushes buffers in to redo log file. After finishing writing process LGWR takes again redo allocation latch and update SGA variable again for frees and reused space allocation of Log Buffer and own inactive status.

How to determine Log Block Size:

Size of redo entries is bytes. LGWR writes in redo log files on disk in log blocks. Size of Log Block is operating system dependent. For windows, Solaris, AIX it is 512 bytes and for HP UX is 1024 bytes. Means smallest disk I/O done by LGWR depends on size of Log Block.

```
Select max(lebsz) from x$kccl;
```

Result of above query reflects size of Log Block.

Log block size is very important for disk I/O and tuning because following parameters directly affected by Log Block size.

Log_checkpoint_interval

Max_dump_file_size

_log_io_size

How Log Buffer flushed into Redo Log File (Disk):

Log buffer is divided into operating dependent block size. Each block of Log buffer is mapped to Redo log file blocks. It doesn't depend on oracle block size but depends on operating system block size.

While LGWR prepares for writing or flushing redo changes from Log Buffer to Redo Log File, there are two sga variables allocated. One maps start of Log Buffers from first block and second maps Redo Log File for end of Buffers. LGWR starts writing process from first sga variable to end sga variable to Redo Log File. Second sga variable also uses for index for free redo space allocation in Log Buffer for next redo generation. After finishing writing into Redo Log File, space between these variables will be freed for next redo generation. Both sga variables are protected by "redo allocation" latch. Means this latch is required for free up space in Log Buffer also.

What writes in Log buffers?

All changes are being written in log buffers. Single change entry calls as "Change Vector". Change Vector represents change in single block.

Detail of Change Vector:

Each Change Vector contains a header. Header of Change Vector contains change number, change type, operation code. Change vector also consists with physical file location, Segment id, Datablock address (DBA) and respective ROWID. It also contains block version number.

While any changes occurs in the database the set of change vectors are generated and logged into redo log buffers. There is no guarantee only one change vector will be generated for only single DML operation. Set of change vector number is depending on number of segment affected and involved in single change execution.

We can take example for better understand this.

DML Operation:

```
SQL> insert into dept values (50,'COMPUTER','MUMBAI');
```

Above insert statement create single row in to DEPT table. But when we are talking about "change vector" result is different than single insert.

- 1) Transaction is started and goes to Undo segment's header for checking and allocating space in undo segment. Undo header will be modified and change vector is generating for this task.
- 2) Spaces in the UNDO segment is allocated and insert entry stored in the segment for undo record. Change vector will be generated for performing this task.
- 3) Data block of segment is modified for storing transaction detail and locking data. Change vector will be generated.
- 4) If Dictionary managed tablespace contain affected segment than may be freelist and free extent will be modified. Change vector will be generated for doing these tasks.
- 5) If any index segment involves in this transaction then leaf block of index segment will be modified using index key generation. Change vector will be generated for performing this task also.
- 6) If any trigger exists on same table segment and it flushes data in another segment also then change vector will be generated for fulfilling this task.
- 7) If audit is enabling in database and performed DML is satisfy condition of auditing then audit entry also inserted in system's audit table. Means change vector will be generating.

Above example shows how change vector is generated during single DML operation.

Usage of Change Vector:

For undo and roll backing data changes.

It uses during archive log applying during recovery operation. In the log minor utility.

Log Miner utility:

This utility masks change vector and generates SQLs from bit and byte conversion. Can you check we don't get original SQL using log minor utility.

Example: Using single delete statement we deleted 1000 rows. For deleting these all rows we use only single DELETE statement. While we configure LOG MINOR we got 1000 SQL REDO and SQL UNDO statements from v\$log_contents not getting original single DELETE statement or INSERT statement.

Important Redo Latches:

There are three latches most important to protect Redo Log Buffer and Redo Log file.

Redo Copy Latch:

This latch is most important for writing in Log Buffer. This latch is required to write any redo record in to Log Buffer. It is only released after Log Switch occurs for freeing space and re-use of Log Buffer. Parameter log_simultaneous_copies represents number of redo copy latch in database.

Redo Allocation Latch:

This latch is required for allocation space in the Log Buffer. Number of redo allocation latch depends on parameter log_parallelism. If redo record is too small then this redo allocation latch copies redo records in to Log Buffer. Mean in this scenario "redo copy " latch doesn't required for copying redo/change vector in to Log Buffer.

Redo writing Latch:

This latch is requiring preventing multiple posting the LGWR process requesting log switch simultaneously. Process should be needed this latch before post the LGWR to write or execute log switch.

Tuning of latches in Redo:

Oracle 7.3.1 and Oracle 8.0.5

If parameter log_small_entry_max_size sets non zero value and redo entry is smaller than the value of log_small_entry_max_size then redo copy latch is not required to copy in to Log buffer. In this scenario only "redo allocation" latch performs this task means copies redo entries in to Log Buffer. If size of redo entry is more than log_small_entry_max_size then "redo copy" latch is must required to copying redo entry in to Log Buffer.

Oracle 8i and Oracle 9.0.1

Parameter `log_small_entry_max_size` is obsolete and "redo copy" latch is must required to copy redo entry in to Log Buffer in all case.

From Oracle 9.2.0.1

New parameter `log_parallelism` is introduced. It represents multiple redo allocation latches in the database. Log Buffer is deviding in to `log_parallelism` areas and each area is protected by unique redo allocation latch.

What is the Optimal size of Redo Log File?

Determine optimal size of Redo Log File, we should consider `V$INSTANCE_RECOVERY` dynamic view. Column of this view named `optimal_logfile_size` recommends size of Redo Log File. We can use Redo Log file advisory to check size of Redo Log File. But value is depending on existing `fast_start_mttr_target` parameter. Means it should decrease also can change value of `optimal_logfile_size` or Redo Log Sizing Advisory.

Related Wait events and statistics:***LGWR wait for redo copy***

While LGWR is waiting to flush data from log buffer to redo log file and finds a process holds redo copy latch and being active for copying change vector in to Log Buffer. LGWR waits until copy operation finishing, this event called "LGWR wait for redo copy".

Log file syncs

Process is waiting for finishing LGWR to flush data from Log Buffer to Redo log files. User session executes commit and server process send request to LGWR to flush required redo entries from Log Buffer to Redo Log File. After performing this task, LGWR will be sending message to server process that "request finished" (means flushed). Up to this message process has to wait. This waiting time event called as "log file sync".

Note one more point, when user process executes rollback that time also changes flushed from log buffer to redo log.

Redo writer latching time

Total cumulative time waiting of LGWR for flushing data from Log Buffer to Redo Log Files. It includes "redo allocation" and "redo writing" latches.

Rdbms ipc message

If LGWR is not active - waiting time. It is an idle wait event.

Redo wastage

Control structure of SGA contains index into log buffer for redo generation recent to next log buffer block before LGWR releases "redo allocation" latch. LGWR writes from start to end (low RBA to High RBA) and redo entries from Log Buffer to Redo log File. This Log Buffer is partially full and some space is un-used in the last block buffer. Due to this, redo wastage wait statistics reflected.

Sometimes high value of redo wastage represents high active LGWR. It also reflects high ratio of commit and rollback in the database or low sizing of Log Buffer.

log file single write

This wait event occurs due to updating header of Redo Log File. Because all contains are same in all members of single Redo Log Group except information of Header of each member. Contain of header of each member are different in single Redo Log Group.

log file parallel write

LGWR wait time for copying redo entries from Log Buffer to Redo Log Group. For tuning this event we should need to consider "log file sync" and "log file single write" also. After getting ratio and cost of redo writing we can consider to tune this wait event.

Errors find in Alert.log:

"Checkpoint not completed and can't allocate new log"

This error may occur due to DBWR is not finishing writing process to disk and LGWR process wants to reuse redo log file. While checkpoint occurs dirty buffers are flushed from Buffer Cache to respective datafiles on disk. DBWR process writes all dirty buffers to disk. But due to DML operations are constantly performing LGWR process wakes up to write redo entries from Log Buffer to Redo Log File. But found DBWR is not finishing writing process yet then LGWR has to wait up (process suspended) to finish written to disk by DBWR and frees online Redo Log File for new writing.

Some expert recommends solution for this error that increase size of Redo Log File or increase the number of Redo Log Group in database. But if we are increasing size of Redo Log File will cut-off the disk I/O from database. Indication of this error is high DML activity and slow disk I/O in the database system. This error may occur due to slow disk I/O of DBWR process or datafiles. Increasing size of Redo Log doesn't impact to boost performance of disk I/O. Slow disk I/O will be there in database because we didn't try to decrease it but we increase size of Redo Log File.

This error may be solved if we enable ASYNC I/O or increasing DBWR processes or DBWR slave processes. Chronic disk I/O may decrease due to asynchronous disk I/O starts or more process/slave can be done job parallel.

“Private strand flush not complete”

Strand is new technology used from Oracle 10g. Online redo recorded in PRIVATE instead of real time (it is like “in memory undo” introduced from Oracle 10g). Interpretation of error is same as “checkpoint not completed”. But as per Oracle recommendation this error may seen in alert.log and it is expected behavior. No tuning is required. We should need to worry for sequence# gapping while said error occurs. If sequence# gap finds high then size of Redo Log File is to be increased as per above solution or enable ASYNC I/O.

Recommendation for Tuning:

Put all Redo Log files in to faster disks

Use Raw device instead of file system

Enable ASYNC I/O in Database server

Don't put Redo Log File on RAID-5

Use NOLOGGING clause – this clause we use for following operations only

In direct load using SQL Loader

In direct load using Insert

create table as select (CTAS)

create index

alter table move partition

alter table split partition

alter index split partition

alter index rebuild

Check any datafile or tablespace contains BEGIN backup mode

In Oracle 8i and Oracle 9i automatic transaction auditing is enabled for Log Miner utility if you are not using Log Miner utility then disable it. Make false transaction_auditing parameter (transaction_auditing=false). Default value is TRUE.

From Oracle 10g onwards this parameter becomes obsolete and makes hidden (`_transaction_auditing`) and default value is TRUE.

Following things should be remember regarding REDO:

Redo is also generated during ROLLBACK operation.

Redo is also generated in "select for update" statement execution.

Redo is also generated during "delay block cleanout".

Most important tuning of Latch contention and Disk I/O.

Eliminate frequent COMMIT during large insertation.

Prepared by Gitesh P Trivedi

From courseware of Advance Performance Tuning (partial)



DBAMETRIX Solutions