OPS Database Configuration

Author: Arup Nanda
Initial Creation: 11/8/01 **Last Revision**: 1/30/02
Version: 2

Contents

General Configuration

The database is a two-node Oracle Parallel Server (OPS) 8.1.7.1 on a cluster of two Sun Ultra-80 platforms running Solaris 8. The datafiles are based on unix raw slices created on an EMC disk array with about 1 TB capacity. Although partitioning option is installed, it is not to be used. The nodes are named prodsvr1 and prodsvr2.

Important : Although this a two node cluster, only the prodsvr1 node is to be used. The other, prodsvr2 is to be used for failover from prodsvr1. This failover is effected by the specially constructed tnsnames.ora at the client machines.

The database name is PRODB. The oracle related files are stored in Oracle Flexible Architecture format with the base as /u01/oracle on each machine. The `initPRODB.ora` contains all initialization parameters common to both instances. It resides on `/u01/oracle/admin/PRODB/pfile` and it is not shared; but the files are identical in both servers. When a change is made in one, it must be copied to the other server. The instance specific parameters are in `/u01/oracle/admin/InstanceNum/pfile/init`*`InstanceNum`*`.ora` where *`InstanceNum`* is either *`PRODB1`* (for prodsvr1) or *`PRODB2`* (for prodsvr2).

Naming Convention of Oracle DB Related Files

Datafiles: There are several raw slices named in the format `data`*`NNN`*, where *`NNN`* represents a zero padded number starting with 1, e.g. `data001` to `data450`. Each datafile is exactly 2 GB in size. Please note that not all such raw slices are used now; they are reserved to be used when needed.

Redo Log Files: There are two sets of redo logs for the two instances of OPS. Each set has 4 groups and each group has 2 members of 100 MB. The names are in the format `redo_PRODB`*`InstanceNum`*`_g`*`GroupNum`*`_m`*`MemberNum`*, where *`InstanceNum`* is either 1 or 2 for PRODB1 or PRODB2 instances of OPS; *`GroupNum`* is between 1 to 8 and *`MemberNum`* is either 1 or 2. For example, `redo_PRODB1_g1_m1` indicates that it is used by instance PRODB1, belongs to group 1 and it is member 1. So there are a total of 16 redo log files. Groups 1 to 4 are used in PRODB1 and 5 to 8 are used in PRODB2.

Controlfiles: There are 3 controlfiles named in the format `cntl_PRODB_`*`N`* where N is between 1 and 3. Each raw slice is 300 MB but the actual controlfile in that could be much smaller. The size increases slowly but steadily.

Tablespaces

These are tablespaces and of these only three are allowed to grow as they contain application data. The others have been given pre-allocated storage. Each datafile resides in directory `/dev/vx/rdsk/oracledg`.

Tablespace	Size	Datafiles	Extent Management	Comments
SYSTEM	2 GB	data001	DICTIONARY	
RBS_PRODB1	8 GB	data002 data003 data004 data005	LOCAL	Contains rollback segments for instance PRODB1
RBS_PRODB2	8 GB	data006 data007 data008 data009	LOCAL	Contains rollback segments for instance PRODB2
STATSPACK	2 GB	data010	LOCAL	Contains the tables for STATSPACK.
TOOLS	2 GB	data011	LOCAL	Contains tables for miscellaneous users like OUTLN and DRSYS.
TEMP1	6 GB	data012 data013 data014	LOCAL TEMPORARY	Temporary segments for user MSCH
TEMP2	8 GB	data015 data016 data017	LOCAL TEMPORARY	Temporary segments for all other users.

Tablespace	Size	Datafiles	Extent Management	Comments
		data018		
USER_DATA	Variable, Currently 50 GB	data019 to data043	LOCAL	Application data tables
USER_DATA_2	Variable, Currently 50 GB	data044 to data068	LOCAL	Application data tables
INDX	Variable, Currently 50 GB	data069 to data093	LOCAL	Application data indexes
MLOG_SMALL	2 GB Fixed	data094	LOCAL	Snapshot logs of small tables. Please see the Replication Setup Manual for Complete Details.
MLOG_MED	2 GB Fixed	data095	LOCAL	Snapshot logs of the medium sized tables.
MLOG_BIG	2 GB Fixed	data096	LOCAL	Snapshot logs of the big tables.

As you can see the fixed size tablespaces have been places at the beginning of the sequences leaving the rest of the sequenced datafiles to the increasing size tablespaces. All these 2 GB raw partitions are spliced across the raid array to provide the maximum performance with the limited controllers available.

Whenever needed, the DBA should allocate the rawslices from the pool of available ones. The pool of available ones are recorded in a flat text file called `rawslice_list.txt` under directory `/u01/oracle/admin/PRODB1/adhoc` on prodsvr1 only, not on prodsvr2. As seen from above table, rawslices from data094 to data450, i.e. 356 slices equating to 712 GB is available right now, well heeled for the future expansion of the database for three years.

Here are the threshold guidelines for various tables. Please note that it's only a recommendation.

Tablespace	Recommended Percent Free space
SYSTEM	30
RBS_PRODB1	30
RBS_PRODB2	30
STATSPACK	20
TOOLS	20
TEMP1	0 (This is tempspace and there is no need to add space to increase free space)
TEMP2	0
USER_DATA	40

Tablespace	Recommended Percent Free space
USER_DATA_2	40
INDX	40
MLOG_SMALL	50
MLOG_MED	50
MLOG_BIG	50

Archive Logs

The archive logs are either stored on /archive1, or /archive2 or /archive3 filesystems on the storage array common to both the prodsvr1 and 2 machines. The init.ora parameter lists /archive1 as the archive log destination, but the backup software, DB BREEZE, makes the others the destination by issuing an ALTER SYSTEM command.

The format of the archive log is `PRODB_SequenceNum_ThreadNumber.arc`, where *`SequenceNum`* is the archive log sequence number and *`ThreadNumber`* is the thread numbr (instance number).

Parallel Query

Each instance has parallel query servers enabled. The current value is 12 Minimum and 50 Maximum.

Initial Load of Objects into Shared Pool

In order to reduce pinning, a database trigger puts several objects of the owners MSCH and TCOP into the shared pool by using the procedure DBMS_SHARED_POOL.KEEP().

Index Tuning Instructions

From time to time, some indexes in the database tend to get lopsided, a term used by the author due to lack of an official one to describe the phenomenon that affects indexes in an OLTP system. The b*tree indexes have branches and leaves to indicate structure. If the insertion of rows into the table is in such a pattern that more values starting with a particular character or number are inserted more frequently than the others. This makes the branching of the index on one side more prominent than the other, increasing the “height” of the index.

The other situation is period of time, when the rows get deleted from the main table, the index gets “holes” in the places of deleted rows. These two factors combined produce significant access problems for the index and the index needs to be rebuilt. The DBA has to make a judgment call based on the degree of lopsidedness and holes present. The author has provided a complete system to check the above metrics. It’s described in Appendix.

Default Degree of Parallelism of Tables

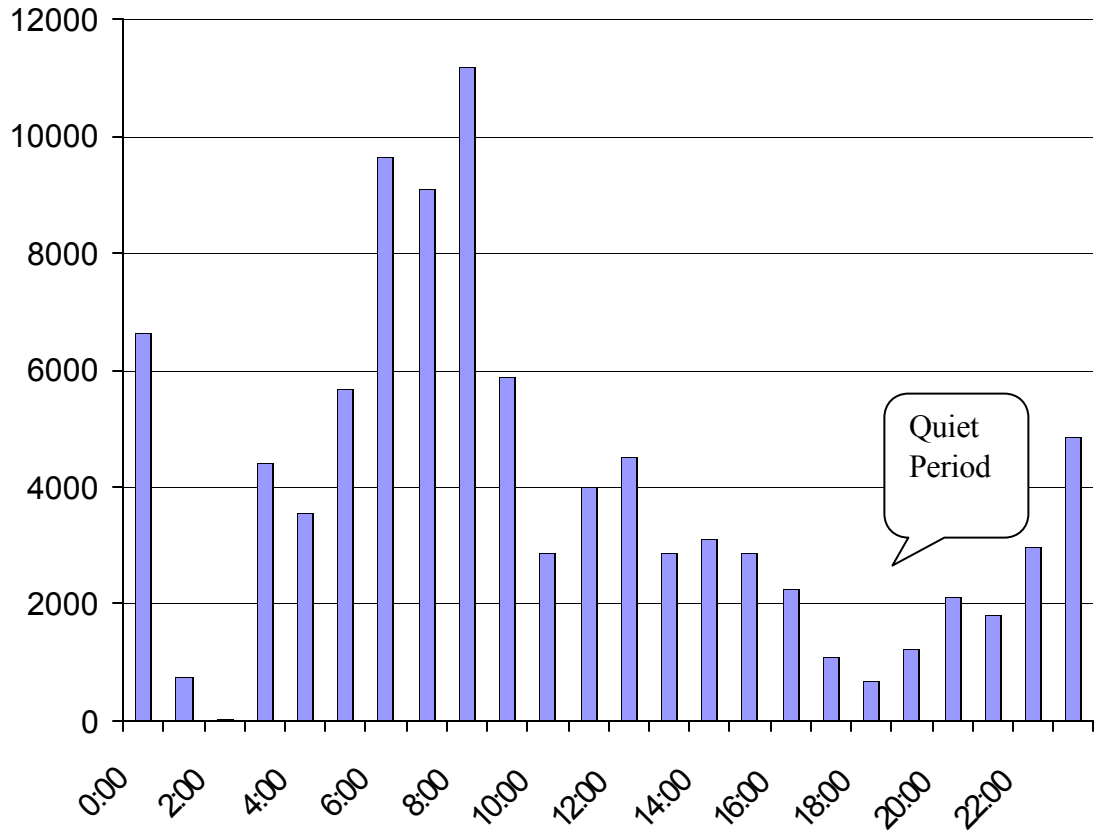
The prodsvr1machine has 4 CPUs and it make sense to have a default degree of parallelism for all big tables as 4; but do not set the default degree. Oracle uses cost based optimizer on queries even though there are no statistics on the tables or indexes and this created wrong access paths, sometimes making full table scans.

STATSPACK

The statspack data collections are invaluable tools in determining both the measurement and scope of the database tuning exercise. The schema is PERFSTAT and it uses the tablespace STATSPACK (2 GB). The snap() procedure is executed every hour on the hour.

Transaction Pattern Analysis

The statspack tables provided enough data to predict the transaction load on the system. The following graph provides a picture of the rate of transactions per hour every hour averaged over the first two months. As you can notice, the period between 17:00 to 19:00 is the quietest.



Maintenance Recommendations

Daily

- Free space in the tablespaces
- Alert.log for any errors, general.
- Between 08:00 to 10:00 in the morning, check for events and waits to see if any significant wait is happening.
- Check Hot Backup has completed all right.
- Make sure all the resources are properly configured as seen from the v$resource_limit view.
- Make sure parallel query servers are properly defined from the PX statistics for that day.
- Check the v$waitstat for that day and make sure no significant waits are occurring.
- Check for parallel server stats for locking conflicts, etc.

Weekly

None

Bi-weekly

Index Check and Rebuilding for the presence of Holes and Lopsidedness as described above.

Monthly

None

Appendix A Index Tuning Techniques

Symptoms of the Problem

1. The buffer busy wait wait event goes up significantly in the session_wait events and the suspected index segment shows up in the parameters of v$session_wait.
2. The query was working fine but suddenly takes a much longer time to complete and the execution plan has not changed.

Identifying Index Lopsidedness and/or Presence of Holes

The holes can be identified by the command analyze index … validate structure. However this command locks the tables in share mode preventing the insert/update/delete activities. So care must be chosen when to run this. From analysis of the statspack report, it has been identified that the transaction rate is lowest between 5 and 7 every evening and even lower on Mondays. This means the command has to be automated to be put in a cron job.

In addition to that, the command itself needs a share lock on the table and if another transaction has held a lock, the command cannot progress. Thus a repeat try is needed when issuing the command.

Both the objectives have been achieved by creating a system to validate the index structures. The system comprises of three tables and a pl/sql procedure to execute the command. Here is a brief description of the components involved.

INDCHK_INDEXES This is a driver table initially populated from DBA_INDEXES WHERE OWNER IN ("MSCH','TCOP'). Its columns are described below. When a new index is created, please add it here.

OWNER	The owner of the table/index.
TABLE_NAME	The name of the table.
INDEX_NAME	The name of the index.
ANALYSIS_OPTION	This is either "REPEAT" or "SKIP". When val_index tries to analyze the index, if there are locks on the table, or index, the command will fail. If this field is REPEAT, then val_index will repeat the command, till RETRY_LIMIT or until it is successful, whichever comes first. If this field is SKIP, then for that index, val_index will skip it when it can not acquire a share lock.
ANALYZED_STATUS	The status after analysis. When you run val_index for the first time, make it NEW. When val_index picks up the index to analyze, it becomes PROCESSED. If the lock on the table cannot be obtained, it becomes, FAILED, or if successfully analyzed, it becomes VALID.
RETRY_LIMIT	The maximum number of times val_index will try to acquire the index lock after an unsuccessful try.
RETRY_COUNT	The current number of retries val_index has had for that index.
STATUS_CHANGE_DATE	The timestamp of the last status change.

INDCHK_INDEX_STATS This stores the index stats after the validate structure command. Please note that the when ANALYZE INDEX VALIDATE STRUCTURE command is issued, the table index_stats is visible only on that session and holds the data for the last analyzed index only. So we need to preserve the rows from that to some permanent space and this is the table for that. The structure is exactly same as the INDEX_STATS table.

INDCHK_HISTOGRAM	This is similar to INDEX_HISTOGRAM table except that the index name is added. Due to the same reason described above, the index_histogram is stored in this table.

The pl/sql component is the program called **val_index.sql**. This program drives off the INDCHK_INDEXES table and issues ANALYZE INDEX <*index_name*> VALIDATE STRUCTURE for each index on that table with the retry options as indicated. After running this, the INDCHK_INDEX_STATS and INDCHK_INDEX_HISTOGRAM tables are populated and are ready for further analysis.

This program val_index.sql should be run as the schema owner that owns the above tables and through a cron job at around 5 PM on a Monday, if possible, every to weeks. The following queries give the indication where or not the index needs reorganization.

```
SELECT INDEX_NAME, ROUND(100*DEL_LF_ROWS/LF_ROWS,0), HEIGHT
FROM INDCHK_INDEX_STATS
WHERE
(DEL_LF_ROWS > 0
AND LF_ROWS > 0
AND DEL_LF_ROWS/LF_ROWS > 0.1)
OR
HEIGHT > 3
```

This gives the deletion hole percentage. If the second column is more than 10, then the index may be considered for reorganization, but may be skipped based on the perception of the hole factor on performance. A value of 20 or above certainly needs reorg.

If the height is more than 3, then it may be a candidate for rebuilding; However the action may be postponed till the height > 4 when it definitely needs rebuilding.

Rebuilding the Indexes

After identifying an index for rebuilding, the index may be rebuilt by the following command.

```
ALTER INDEX <index_name> REBUILD ONLINE PARALLEL 4 NOLOGGING;
```

while connected as the schema owner, MSCH or TCOP. Please note that you do not have to find a quiet time for this operation; it can be done under full load.

After the rebuild, make sure the hole percentage is reduced and height too, if applicable. This can be done by the running the following query

```
UPDATE INDCHK_INDEXES
SET ANALYZED_STATUS = 'NEW', RETRY_COUNT = 0
WHERE INDEX_NAME = '<index_name>';
COMMIT:
```

to make that particular index candidate for val_index script and then running val_index.sql. However, this needs to done at a quiet time as it locks the underlying table.