LIBERTAS MATHEMATICA, VOL. XIV (1994)

PRACTICAL CONSIDERATIONS ON A UNIX CLIENT / SERVER ARCHITECTURE

Viorel Vlad

ABSTRACT. The intention of the author was to present to the "new DBA starter" some of the classical steps that are needed to tune specific parts of the database. The paper focuses on the utilization of the Unix-SAR-tool to detect performance problems and to overview the Oracle concepts of: **extents, fragmentation and rollbacks.** A real case study is presented with its pros and cons.

INTRODUCTION

The final objective of each Software Development Department is to produce software applications that run smooth, correspond to user requirements and function at a minimum cost. Like any car on the road the "running smooth" perception differs from the point of view from which you analyze the problem. A lot of "echos" of "re-tuning" the system have been voiced, because the system had started to run slow and the response time from different screens was higher than normal acceptance. The general consensus was that the system must be tuned, and if possible with minimum production downtime. The application checked for tuning was developed in a client/server architecture in a Unix environment. The Oracle database resides on a 7040 Pyramid computer and development code is done from 5 different 386 AT&T computers via a STARLAN network. The conditions of the application environment change frequently due to a large number of "stress tests".

WHERE WE STAND ON THE UNIX SIDE? To analyze performance, a series of Unix - SAR tests have been done, especially during some critical parts of the day when many developers and testers were logged in. The **sar** is one of the quickest best tools to use if you want to see a snap-shot of the current days CPU activities. It samples cumulative activity counters in the operating system at "n" intervals of "t" seconds.

HOW TO USE "SAR": To use it at the operating system prompt type **$ sar**(most of the AT&T Unix Software has this package already incorporated).
This **System Performance Utility Package** has a lot of options providing CPU utilization, Buffer Activity etc.
A quick response of the system comportment is obtained by examining the **"idle time"** column. Any idle time for a period more than 10-15 minutes less than 25% implies that the system must be checked for tuning immediately. The idle time for our test case was varying between 20% and 30% for a period more than half an hour. Now, besides complaints of the system runing "slow" we have some confirmation.

WHERE WE STAND ON THE ORACLE SIDE - CHECKING THE NUMBER OF EXTENTS

Let's start by checking the number of extents. An extent is an allocation of contiguous database space, expressed as a number of bytes of data. The default is one and the maximum number for the Unix system is 121 (the maximum number varies by operating system type). Oracle dynamically allocates more extents based on the values of the storage parameters. An aggressive dynamic extention will have a big impact on the system performance. It is recommended as a general consensus that the number of extents not be greater than 3-5. Let's check the actual status of our test case. To do this we used the following SQL script called **findext.sql.**

```
spool findext.lst
break on segment type          skip 1
column segment_type            format a10          heading 'TYPE'
column segment_name            format a18          heading 'OBJECT'
column tablespace_name         format a25          heading 'TABLESPACE'
column counts                  format 999,999      heading 'COUNT(*)'
column space                   format 999,999,999  heading 'BYTES'

select segment_type, segment_name, tablespace_name, count(*), sum(bytes), space
from sys.dba_extents
where owner = 'OWNERNAME' and
extent_id  > 3
group by segment_type, segment_name, tablespace_name
spool off;
```

Run this script from SQLPLUS. The output will be written in the **findext.lst** file. A snapshot of this report file looks as follows:

NUMBER OF EXTENTS FOR YOUR SYSTEM

TYPE	OBJECT	TABLESPACE	COUNT(*)	BYTES
INDEX	INDEX1UXP	TBLSPACE5	5	1167360
	INDEX2UXP	TBLSPACE6	9	387072
TABLE	TESTTABLE1	TBLSPACE1	5	235520
	TESTTABLE2	TBLSPACE1	4	1234892

How can we get rid of these extents ? The only way is to increase the initial space in the "ddl" scripts which have been used to create the above indexes and tables. Of course, before starting to increase the space, you need to determine **how much space is available in any tablespace.** The answer to this question is obtained by running the following script , called **freespace.sql.**

```
column tablespace format a15
ttitle 'INITIAL ALLOCATED TABLESPACE' skip 4

select tablespace_name TABLESPACE, sum(bytes) TOTAL_BYTES, sum(blocks) TOTAL_BLOCKS
from sys.dba_data_files
group by tablespace_name
/

ttitle 'TABLESPACE FREE SPACE' skip 4
select tablespace_name TABLESPACE, sum(bytes) "Free Bytes", sum(blocks) "FREE B:LOCKS"
from sys.dba_free_space
group by tablespace_name
/
exit;
```

A snapshot of this report file looks as follow:

INITIAL ALLOCATED TABLSPACE		
TABLESPACE	TOTAL_BYTES	TOTAL_BLOCKS
TBLSPACE1	120586240	58880
TBLSPACE2	120586240	58880
TBLSPACE3	155189248	75776

TABLSPACE FREE SPACE		
TABLESPACE	FREE_BYTES	FREE_BLOCKS
TBLSPACE1	40519680	19785
TBLSPACE2	35604480	17385
TBLSPACE3	1046528	511

After you determine that you have enough space, then you must recreate the objects with extents by dropping and regenerating them with new space allocation. Two common mistakes always happen on dealing with the extents and must be avoided:

1. Dropping the object and recreating the object with the "hope" that Oracle will reorganize the space for you,
2. Increasing the space allocation for an object without checking the free space inside the tablespace

After all your objects with extents greater than 3 have been dropped and recreated with new space allocation, it is recommended to run the "findext.sql" script again. This will give you a second check on the number of extents, so you can verify your work.

WHERE WE STAND ON THE ORACLE SIDE ON FRAGMENTATION

The primary cause of database fragmentation is inaccurate space definition. It is very important from the design phase of a table to know the mechanics concerning the data and indexes. Oracle provides a formula to design the required space for table and indexes, but these estimates are good only for the first months after application implementation. In real life, the business conditions change so frequently that even the best "requirements analysis" become obsolete some months after implementation.

A secondary cause will constitute the "mechanics" of objects themselves. The frequency of deletions and insertions will have a big impact on fragmentation. When the objects are dropped or modified the contiguous free space will be divided. This will create a lot of chunks of blocks of data that cannot be used by Oracle because they are too small.

Before you decide to start the process of defragmentation, a clear response must be given to the following question:

1. Does your database need defragmentation on a whole scale, a specific tablespace or only for some specific objects(tables, indexes) ?

The way you answer these questions will determine the type of methodology to be used.

CASE A: ONLY A FEW TABLES NEED DEFRAGMENTATION

If you have only a few tables that are fragmented a quick solution can be obtained by using the following steps:

1. Identify the tables with a heavy fragmentation pattern.
2. Save the data in some temporary tables, drop the initial ones and recreate them using the initial "ddl" scripts.
3. Insert the data from the temporary tables to the newly created tables.

CASE B: THE WHOLE DATABASE NEEDS DEFRAGMENTATION

If the number of tables that are fragmented is high enough, then it is recommended to use a full **EXP/IMP** of the database.

CASE C: A BIG NUMBER OF TABLES FROM A SPECIFIC TABLESPACE NEEDS DEFRAGMENTATION

Let's check our database for the existence of those chunks. The study can be done by tablespace (our detail case) or for the entire database. For this, run the following script: **fretbl.sql**(reference #1).

```
set feedback off;
set term on;
set pagesize 60;
set newpage 0;
set linesize 78;
spool tblfrag.lst
ttitle 'TABLESPACE FREE SPACE REPORT'
col tablespace_name     heading 'TABLESPACE_NAME' format a15
col file_name           heading 'FILE' format a18
col block_id            heading 'START_BLOCK' format 999,999
col blocks              heading 'UNUSED_BLOCKS' format 9,999,999
break on report on tablespace_name skip page on tablespace_name
compute sum of blocks on file_name
compute sum of blocks on tablespace_name
compute sum of blocks on report
select sys.dba_free_space.tablespace_name, file_name, block_id,
    sys.dba_free_space.blocks
from sys.dba_free_space, sys.dba_data_files
where sys.dba_free_space.file_id = sys.dba_data_files.file_id
and
sys.dba_free_space.tablespace_name=sys.dba_data_files.tablespace_name
order by sys.dba_free_space.tablespace_name, file_name, block_id
/
spool off;
clear columns;
clear breaks;
ttitle off;
```

A snapshot of this report is the following:

TABLESPACE FREE SPACE REPORT			
TABLESPACE_NAME	FILE	START_BLOCK	UNUSED_BLOCKS
TBLSPACE2	/dev/rdisk/c1t1d0s3	32287	348
		32635	250
		35485	535
SUM			1133

It shows that TBLSPACE2 has 1133 unused free blocks. The next step is to find the list of all the objects pertinent to this tablespace. In our testing case this tablespace is comprised only from indexes (38 indexes). For defragmentation use the following steps:

1. Create and Execute an "export" file which include all the objects from "TBLSPACE2" with compress = 'Y'.
2. Take out TBLSPACE2 off line by executing the following alter command:
   SQLPLUS > alter tablesapce TBLSPACE2 off line;
3. Drop this tablespace using the following command:
   SQLPLUS > drop tablespace TBLSPACE2 including contents;
   For a tablesapce with about 50 objects usually takes around 15-20 minutes to execute.
4. After the tablespace has been dropped, use the initial script to recreate this tablespace.
5. Run the "IMP" utility. Make sure that the import file includes the same objects, as the export file.

WHERE WE STAND ON THE ORACLE SIDE ON ROLLBACKS

Some valuable lessons we learned from rollbacks was when our software application had been running on a 7040 Pyramid computer in a client/server environment and had to be moved to a AT&T-386 computer with a smaller memory capacity. This computer will be used only for developing and testing purposes, so the storage capacity of each database object can be diminished. Also, a request has been made that the configuration(except memory allocation) of the two computers be the same.

The AT&T-386 computer has been prepared for this transition, and a full reevaluation of file systems, tablespaces, tables and indexes has been done, to accommodate the new amount of data. Basically, the new storage capacity has been reduced.

ROLLBACK SEGMENTS: These segments are used by Oracle to allow transactions to be "*rolled back*" prior to commit. A rollback segment records unconfirmed actions issued against the database that should be undone during recovery. These rollback segments are stored inside the tablespaces, and the way you calculate the ideal number of how many your application needs, the storage space, minimum and maximum number of extents, will have a big impact on the functionality of your application.

HOW MANY ROLLBACKS DO I REALLY NEED? The easy way to answer this question was to accept the same number of rollback segments that initial application had. But, having memory capacity problems (very limited space) the intention was to reduce their number or to reduce their storage capacity.

Some Oracle professionals with experience in the field claim that "*one rollback segment is needed for every three users of the database*" (see reference #2). I found this number relatively small. If I extend this "rule of thumb" to the applications I worked on for the last years, this number will grow to 6-8. Of course these applications had to support a lot of concurrent activity transactions. The final decision was to go with the same number of rollbacks(8) but with a diminished storage capacity.

A full *export file* of the initial database had been executed, and the AT&T-386 computer with the new storage requirements was ready to "import" this file.

The first big problem encountered was moving the export file (70M) using the **uucp facility** via the network. Although the file was "**packed**", after three unsuccessful attempts to move it on the network, it was decided to "**split**" the file into 10 different small files using the "**Unix split**" command. Even this didn't help us. After the file was "recreated" after the "split", we couldn't execute the "pack". The only way to move the file was by increasing the value of "ulimit" to 2113674. This helped us move the file to the new location. After the file was put together in its initial state it was ready to be imported. During the "import" process we encountered Oracle errors related to rollback segments.

The first wrong assumption was to believe that the import transaction would pick up the first rollback, create extents, and when full, the next rollback would continue etc...This was wrong. After the first rollback is acquired, it will grow and will produce more extents, these extents will grow as long you have space in the tablespace. By Oracle standards this is considered one of the "rule" that govern the functionality of the rollback segments:

If a rollback segment is not large enough to accommodate a big transaction, then the RDBMS will extend the rollback segment like it extends any other object in the database".

Three attempts have been performed to increase the tablespace and the memory capacity of those rollback segments, but without any success.

The next logical step was that, besides the 8 initial created rollback segments, create a "big" rollback segment(70M) to accommodate our "big" "import" file. Using the standard procedures this big rollback segment had been created. By checking the **monitor** in the SQL*DBA the actual status of the system indicated that at this time there were available:the SYSTEM rollback, eight small rollback segments plus the "big" 70M rollback segment.

The import procedures had been started again, and unfortunately the same type of Oracle error appeared, this time for rollback #5. What happened ? During the import file the transaction selected one of the small rollback segments **instead** of selecting the "big" rollback segment with enough memory (that's what we expected to happen, but again we were wrong..). Later, we realized that another "rule" is governing the rollback behavior:

You cannot control which rollback segment will be chosen for a given transaction(Version V6).

So even if you have a big rollback segment with enough memory, **it is not guaranteed** that the transaction will pick up the one you expect.

At this point, the eight rollback segments have been dropped, leaving intact only the big rollback. Finally our import transaction performed O.K, and all the tables have been populated. The eight rollback segments have been recreated again and the database has been started after the line which designated the big rollback segment in the **init.ora** file. has been blocked(#).

The database was running relatively smooth and after one week of usage a checkup had been

performed to the number of extents for eventual tuning. We were surprised to discover that our big rollback was still there with a big number of extents(67). We realized that the big rollback which was blocked to be selected by the Oracle instance, was still there eating memory space. Then we understood the next "rule":

A rollback segment will never disappear (or become smaller) without the manual intervention of the DBA.

This being the case, the rollback was immediately dropped out.

Conclusions and Recommendations

1. In your daily routines, try to examine the number of extents. Any increase in this number represents a warning signal for DBA.
2. Check at a minimum once a month the database for fragmentation and act accordingly. A big fragmented database can lead to a halt of your application.
3. Moving a database application from a powerful computer to a smaller one needs a lot of planning and strategies. Besides the fact that all tablespaces, tables, indexes etc must be reevaluated, attention must be given to the rollback segments.
4. If you have to import a "big" file (40M-70M) always use, for the "import phase" a "big" rollback segment to accommodate the transaction.
5. After this rollback segment is created, if your configuration still has rollback segments, it is **NOT guaranteed** that the transaction will pick up the one with the biggest memory space.
6. A rollback created will stay forever (by occupying space). The fact that you restart the database by blocking a rollback in the **init.ora** file(#) means only that the rollback will not be used for that particular instance.
   You need to manually drop a rollback if you do not need it anymore.

By writing this paper the author's intention was to share with some of the readers the problems encountered during the transition process. Perhaps obvious for some readers, but maybe there are still some who will benefit from this experience.

REFERENCES

1. Rachel Caramichael, Marlene Theriault, All the things they never told us. Proceedings ECO-92.
2. Alan Shafer, "So You're the New DBA". Proceedings IOU-1990.
3. Rollback Segment Configuration for ORACLE RDBMS Version 6.0. Oracle Corp.