Most Common Database Configuration Mistakes

Jože Senegačnik, Oracle ACE Director, Member of OakTable
DbProf d.o.o.
Ljubljana, Slovenia

Keywords
Oracle Database, Configuration,

Introduction

The purpose of this paper is to discuss the most common database configuration mistakes I have encountered in my performance firefighting in recent years.

In older versions of Oracle database there were many parameters which were not set reasonably during the installation process. Like the database was improving also the database installer was improved and in latest versions of Oracle database (here I have in mind 10g and newer) there are less parameters which should be set after the initial installation. The most critical ones which depend mostly on the "size" of the underlying hardware are already set or could be set through the Oracle Universal Installer (OUI). If we think about the parameters which influence the database setup we can think about two major components: the amount of physical memory available and what kind of storage we have and also the number of connections (sessions/processes) we plan to have on our database. For the majority of other parameters usually the default values are a very good starting point. I would recommend changing them only when you are really sure that you have to change them. The most important thing is that you really understand the purpose of the parameter and what are the built-in algorithms behind. Unless you really understand how the parameter acts I would strongly suggest that you leave the parameter unchanged. Many parameters depend on the values of other parameters so you should really know what is their purpose. In recent years I have seen so many databases with strange values for certain parameters. When I have asked DBAs why they are set in this way, they have no good answer or one of the frequent answers was: "We found that setting and recommendation on the Internet." Because somebody wrote something on the Internet you should not apply without any thinking and reasoning. Sometimes I see so-called "Best Practices" as a kind of a framework which is meant in a good manner that one should develop his own practice for his particular case, but is unfortunately followed blindly without any reasoning. The most cases are some database parameters influencing the optimizer (CBO), which were introduced in old versions like 8i in order to resolve optimizer deficiencies, but have nothing to do in the database setup in latest version. A good example of such parameter which should not be set at all since Oracle 9i on is the OPTIMIZER_INDEX_COST_ADJ. Any non-default value of this parameter will have influence on the prepared execution plans by favorizing index access to table data although this might not be the optimal access path. Since Oracle 9i when the system statistics was introduced one can set the system statistics in order to tell the optimizer what are the real times for single/multi-block reads. This parameter was introduced in version 8 but was later on replaced with other algorithms and therefore should not be used any more.

Therefore it is very important that when we upgrade the database to a newer version we also check all non-default parameters which are set in our init.ora or spfile and simply comment them if they should not be set. The "after upgrade" testing is the right time to find out if such "obsolete" parameters should be set still in the newer version. It is well known that after any upgrade some of the SQL statements will run have changed execution plans and we notice this because they become the performance bottle neck. So it is a very good idea that during the upgrade process we get rid of such parameters and we don't react only on Oracle warnings about "deprecated" parameters.

And here is another warning about setting initialization parameters just to fix a certain problem. If you have to set a parameter, please be aware that this parameter will have influence on every SQL statement running on your database, therefore instance wide. So fixing execution plan of one SQL statement by changing the value of the initialization parameter will have instance/database wide scope. Just copying some values from Internet without knowing for the background might drive you to even bigger problems.

But let’s start our main topic – what kind of common misconfigurations I have seen in last five years during the performance firefighting. With no doubt I can say that I have seen two common problems, especially on large databases:

- Connection management (database session/processes misconfiguration)
- Memory misconfiguration

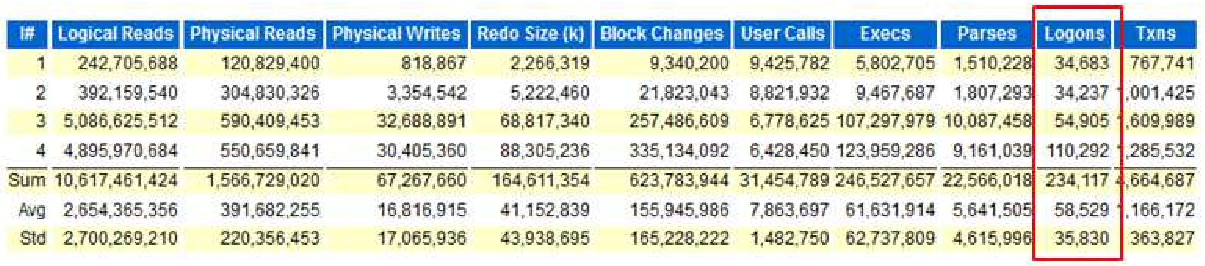
Of course I have seen some other problems, however most of them were related to schema/application design or database misuse. For instance, I was working on one database where there were roughly about 6000 identical schemas. By identical I mean that they had roughly about 30 tables with same names. Only one schema was common for all users. Can you imagine what happened in the library cache with all those identical tables which belonged to different schemas. The database encountered huge parsing problems as the text of SQL statements was identical but the objects referred to were owned by different schemas. The parsing mechanism could not reuse statements as they were not sharable and we had thousands of child cursors all pointing to different schema objects. This was an excellent example how an application design should not be performed.

Connection Management

In latest years most of the environments are running in multi-tier architecture. There are still some customers running in client/server mode (legacy Forms/Reports applications). With large number of end-users connecting to middle-tier many sites suffer due to performance problems in peek loads – like Black Friday, Christmas time, …
The scenario goes like this: the application server connects to database and executes SQL statements. Many times the connection pools are configured with a small initial number of connections and no practical limitations with the maximum number of connections. When more and more end users perform their work the load on the database increases what in long term means that with many concurrent users the database will start to respond slower as the concurrency in the database will increase. When the application detects that all current connections are busy due slower response time it starts creating new connections. Creating a new session (new connection) is one of the most complex tasks in Oracle database, killing the session is even more as it has to rollback uncommitted transactions and deallocate all resources. Now consider the following scenario: the database is heavily loaded and the application tier starts creating many new sessions what just increases the database load and the database will respond even slower. The spiral dive starts. Many DBA’s don’t know that the session creation is really not instrumented until the session starts executing SQL statements. Therefore one can’t monitor this kind of database activity, so periods when the database is going down to the knees are not debuggable and we can’t monitor them directly. The only possibility is to check the session creation statistics. **As a rule of thumb:** when there is more than 1 database logon per second this means that most likely you are suffering from bad connection management.

The symptoms for this kind of problem are:

- High number connections to the database (1,000s)
- Dynamic connection pool with a large number of logon/logoff to the database (>1/Sec)
- Periods of unexplainable/undebuggable poor performance/availability
- Inability to determine what is going wrong



I#	Logical Reads	Physical Reads	Physical Writes	Redo Size (k)	Block Changes	User Calls	Execs	Parses	Logons	Txns
1	242,705,688	120,829,400	818,867	2,266,319	9,340,200	9,425,782	5,802,705	1,510,228	34,683	767,741
2	392,159,540	304,830,326	3,354,542	5,222,460	21,823,043	8,821,932	9,467,687	1,807,293	34,237	1,001,425
3	5,086,625,512	590,409,453	32,688,891	68,817,340	257,486,609	6,778,625	107,297,979	10,087,458	54,905	1,609,989
4	4,895,970,684	550,659,841	30,405,360	88,305,236	335,134,092	6,428,450	123,959,286	9,161,039	110,292	1,285,532
Sum	10,617,461,424	1,566,729,020	67,267,660	164,611,354	623,783,944	31,454,789	246,527,657	22,566,018	234,117	4,664,687
Avg	2,654,365,356	391,682,255	16,816,915	41,152,839	155,945,986	7,863,697	61,631,914	5,641,505	58,529	1,166,172
Std	2,700,269,210	220,356,453	17,065,936	43,938,695	165,228,222	1,482,750	62,737,809	4,615,996	35,830	363,827

Figure 1: Inadequate Configured Middle Tier (from AWR report)

One can check the AWR report and see what the number of logons for a period of AWR report is. Figure 1. shows such a case. The AWR report was produced for the 3 hour period and one can see that on this 4-node RAC (Exadata) there was a large number of logons performed.

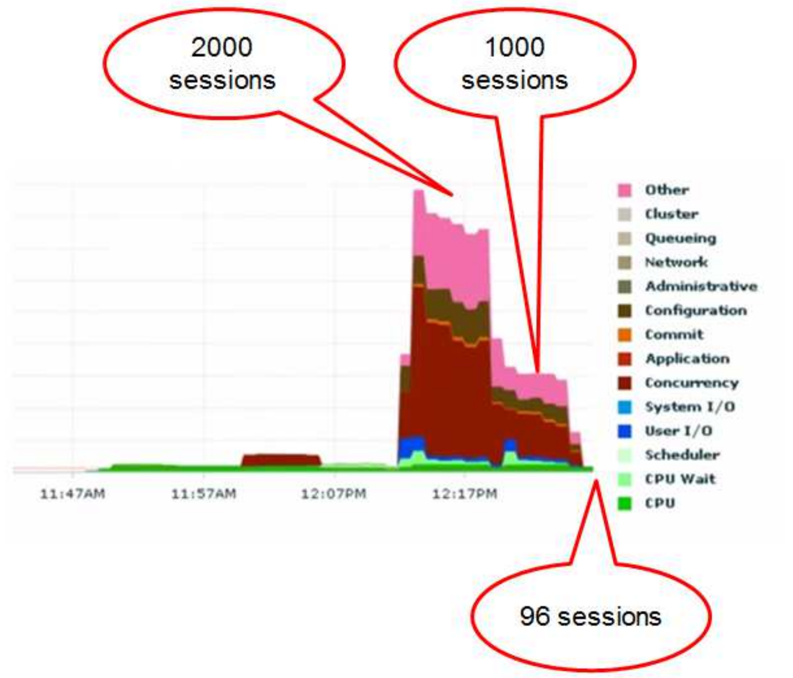


Figure 2: Concurrency at high number of concurrent sessions

Figure 2 shows a classical problem with too many concurrent sessions. Decreasing number of sessions from 2000 to 1000 and then to “only” 96 shows that the concurrency wait time vanishes while the actual work (the CPU time) increases.
The most important lesson to learn from this is:

- The database tier is not the tier for queueing
- Dynamic connection pools are dangerous and should not be used

Some other facts which should be always checked are:

- What is the number of configured sessions/processes on the instance level
- What was the maximum number of sessions during the performance problems

Typically one can see that on large systems the number of sessions/processes is defined completely wrong. In my cases I found that the number of sessions was defined as 30 000 or even 45 000 per RAC node. Immediately one should ask himself: how many active sessions can be running on 1 CPU – probably much less than 10 – usually only few. So when you see so highly defined number of sessions you should compare it with number of CPUs and you can easily spot the problem.

Another negative result of defining extremely high number of sessions is the overall performance degradation and increase of DB time even in "idle" periods. Sessions/processes parameters are static parameters because the number of sessions/database processes are fixed arrays defined in SGA. So if you define 10 000 sessions certain X$/V$ views will have to loop through huge arrays inside SGA. Remember that X$/V$ views are externalized C language SGA structures.

To resolve such problems one should:

- Properly define the number of sessions/processes
- When using dynamic connection pools the initial number of connections is also the maximum number of connections and therefore we disable possible spiral dive.

Oracle Real-World Performance Group has published two videos on Youtube where you can watch a possible scenario with this kind of problems.

See: Real-World Performance - 13 - Large Dynamic Connection Pools - Part 1
https://www.youtube.com/watch?v=Oo-tBpVewP4&list=PLKCk3OyNwIzvwEXdaubc6PQXwnQOAE9h2&index=16

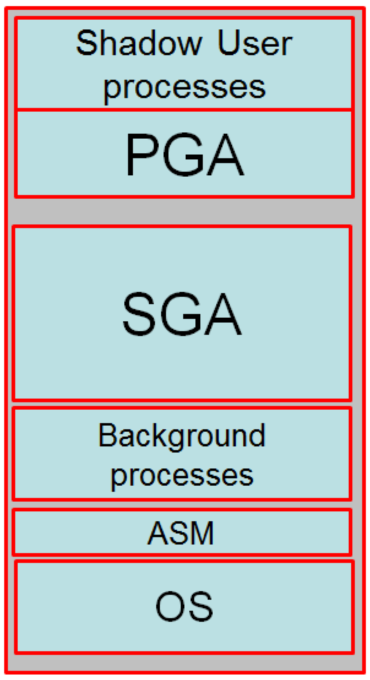
See: Real-World Performance - 14 - Large Dynamic Connection Pools - Part 2
https://www.youtube.com/watch?v=XzN8Rp6glEo&index=17&list=PLKCk3OyNwIzvwEXdaubc6PQXwnQOAE9h2

Memory Misconfiguration

Our next topic is memory misconfiguration. Although the amount of physical memory in the latest HW is dramatically increased, there are many case when the memory on the database server is misconfigured. Contributing factor to potential cases is virtualization which is now widely used either because of consolidation or to adjust the real situation with the actual number of Oracle database licenses. Currently there is hard to find appropriate boxes with just the right number of sockets/processors that would perfectly fit with the number of acquired database licenses in the past. Therefore many customers have to use "legal way" to "downsize" the new hardware with the licensing requirements. Most of the time this should not pose a problem to memory configuration, however when consolidation is in question, it can easily come to a situation when the amount of physical memory is not enough for all instances one wants to run on that server. One of possible solutions is multitenant option in 12c, which will significantly optimize the memory usage for several instances. Unfortunately this option is not for free.

In past years I had encountered so many memory misconfigurations and it was always a big surprise for me as I would never expect this especially from experienced DBA teams. However, I must mention here that memory misconfiguration sometimes can't be easily spotted when there is only a small OS level paging/swapping.
Traditionally on Unix OS (Linux,AIX, Solaris) there are system utilities which one can use to determine the amount of memory allocated by certain process. Unfortunately all these tools are displaying some values for memory usage which could be far from the real memory usage. The

problem are private parts of memory allocation besides the amount of memory used by shared libraries. To get a precise amount of memory used by a process one has to sum together all memory allocations made by this executable and also all dynamic shared libraries. Just looking at various sizes reported by OS level tools gives just a big number which is actually mostly represented by shared segments, but we can't figure out what is the actual private memory used by this process.



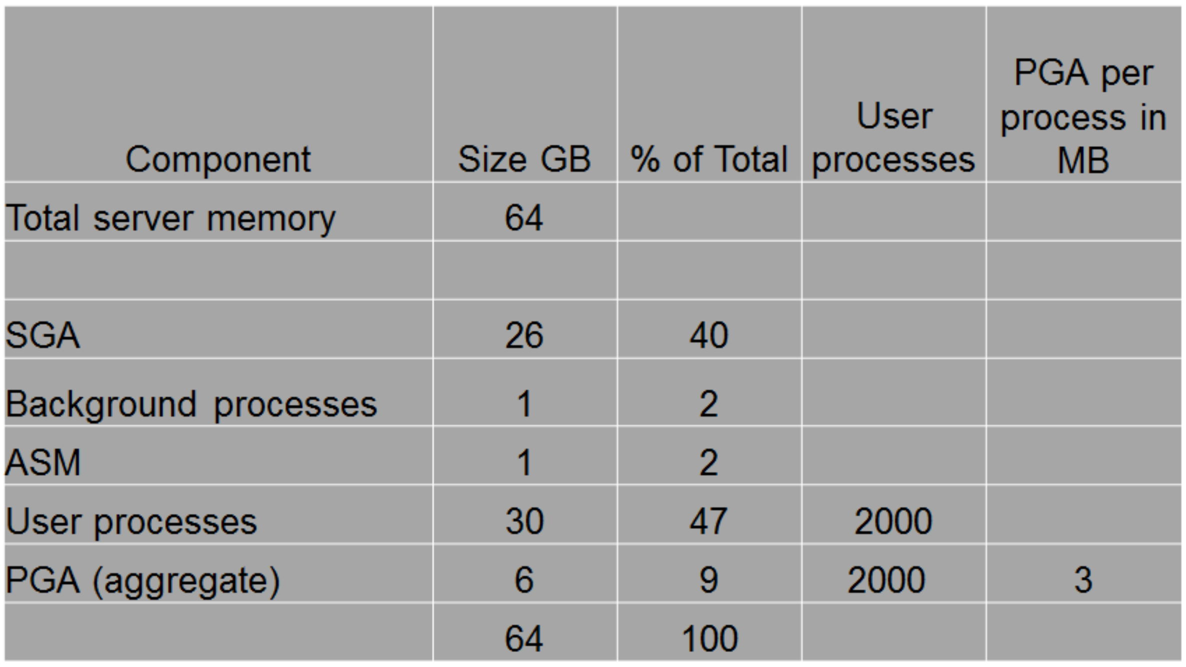
- Dedicated server mode – each session has its own server process
- Each process needs some memory space + all dynamic PGA allocations
- Each session (server process) needs some memory (PGA allocations are separate)
- Background processes need also some memory
- PGA_AGGREGATE_TARGET is only a target, not limit
- SGA (size easily detectable)

Figure 3: Typical memory map on a database server

Figure 3 depicts typical memory usage when we have hypothetically only one instance running on server. When we have large number of shadow user(server) processes (server dedicated mode) than this number could be quite high. In previous part of this paper we saw configurations when there are several thousands of sessions connected to the database what will result in several thousand shadow processes. For all these processes the majority of code segments is shared. However, beside PGA, which is dynamically allocated/deallocated, there are many private memory allocations performed by the oracle executable and also all dynamic libraries, which are completely private to this shadow server process.

Just for a moment let's remember what are the defaults for memory setup when you run Oracle Universal Installer. The default is 40% of physical server RAM is dedicated for SGA. Do you think this default is a good number? The answer is as always: it depends what kind of a database you are installing. But let us consider the memory map depicted in Figure 3.

In this case we have up to 2000 user sessions (maybe these are Forms/Reports) sessions and typically on 11gR2 we can for Forms session calculate about 15MB of memory per session. In Oracle 9i this was only 5MB per session. And when we look at the amount of memory used we can see that just the non-shared allocations in user server processes are using 47% of total amount of server memory. If you look carefully there is no RAM available for OS on Figure 3.



Component	Size GB	% of Total	User processes	PGA per process in MB
Total server memory	64			
SGA	26	40		
Background processes	1	2		
ASM	1	2		
User processes	30	47	2000	
PGA (aggregate)	6	9	2000	3
	64	100		

Figure 3: When SGA size is 40% of server's physical memory

In order to detail analyze the memory usage by process one should use **pmap** on Linux or **procmap** on AIX.

The sample output on AIX is like this:

procmap <process id>

```
root# procmap 21430470

100000000           202626K  read/exec         oracle
110000070             6728K  read/write        oracle
9fffffff0000000         62K  read/exec         /usr/ccs/bin/usla64
9fffffff000fa7e          0K  read/write        /usr/ccs/bin/usla64
9000000004c6980         92K  read/exec         /usr/lib/libsrc.a[shr_64.o]
9001000a011f3e8         55K  read/write        /usr/lib/libsrc.a[shr_64.o]
90000000049dd00        157K  read/exec         /usr/lib/libcorcfg.a[shr_64.o]
9001000a03d9550         40K  read/write        /usr/lib/libcorcfg.a[shr_64.o]
9000000006be700        780K  read/exec         /usr/lib/liblvm.a[shr_64.o]
9001000a03a1188        221K  read/write        /usr/lib/liblvm.a[shr_64.o]
900000000486800         88K  read/exec         /usr/lib/libcfg.a[shr_64.o]
9001000a02b2028         29K  read/write        /usr/lib/libcfg.a[shr_64.o]
…
```

From the above output we can see that there are many private memory allocations in the category “read/write”. All such allocations should be summed together for a process and then summed together for all processes.

Searching through MOS and Internet one can find documents describing this problem in details:

- FAQ: How can I Investigate Memory Usage on my Unix/Linux Server (Doc ID 1447481.1)
- AIX: Determining Oracle Memory Usage On AIX (Doc ID 123754.1)
- How to Check and Analyze Solaris Memory Usage (Doc ID 1009500.1)

- Script for Checking the Grid Control Agent CPU, Memory & Threads Usage (Doc ID 464414.1)
- How to Calculate Memory Usage on Linux (Doc ID 1630754.1)
- **Tanel Poder - Using Process Memory Matrix script for understanding Oracle process memory usage**
  - http://tech.e2sn.com/oracle/performance/unix-performance-tools/process-memory-matrix
- **Oracle Instance Memory Usage**
  - http://www.pythian.com/blog/oracle-instance-memory-usage/

One commits a “mortal sin” in terms of configuration when the memory usage is going beyond the available physical memory of a database server, especially if this is production environment. Therefore careful planning of memory usage and later on monitoring is essential, especially when we have many database instances running on one physical server. The “top” number of instances running on a 2 socket box was 80 – you can imagine how this box was performing.

Conclusion

In this paper I discussed just two quite common, but not so obvious misconfigurations which could have enormous impact on the database performance. In excuse for finding a memory configuration problem DBAs defend themselves by saying: “But all those databases are just up, nobody is using them most of the time.” Unfortunately they forger that the database background processes are constantly working and there are many background tasks performed quite frequently (CKPT process, AWR snapshots, automatic diagnostics,….). Actually a database is virtually never idle and therefore we should configure it properly.

Contact:
Jože Senegačnik
DbProf d.o.o.
Smrjene 153
SI-1291 Škofljica
Slovenia

Telefon:	+386 41 72 44 61
E-Mail	joze.senegacnik@dbprof.com
Internet:	www.dbprof.com