YOUR machine and MY database - a performing relationship!?

Martin Klier, Senior / Lead DBA
at Klug GmbH integrierte Systeme, Teunz, Germany

Introduction

This Paper

“YOUR machine and MY database - a performing relationship!?” is intended to be an information for Oracle DBAs, DB developers and system administrators who want to learn more about how databases, operating systems and hardware works together.

Databases affect machines, machines affect databases. Optimizing one is pointless without knowing the other. System administrators and database administrators will not necessarily have the same opinion - often because they know little about the opposite's needs. This lecture was made to promote understanding - showing how the database can stress the server, and how the server can limit the database. And why two admins sometimes don't speak the same language, not even with a developer as an interpreter.

- Recall the different needs of different technical layers underneath a database system.
- Understand the technical collaboration of hardware, operating system and database.
- Plot ways how to avoid collisions, competition and concurrency.
- Promote collaboration!

This white paper and its presentation were written in late 2013 and early 2014 from scratch for IOUG forum at COLLABORATE 14.

Author

Being in IT business as a specialist for Linux, **Martin Klier** (martin.klier@klug-is.de) has been involved in the administration of Oracle Databases for about ten years now, and works on senior level for more than four years. The integration of large Oracle-based high-availability solutions (MAA) with RAC and Data Guard were the first challenges. In the last six years he largely moved into performance analysis and tuning of complex systems with dynamically changing load profiles. Skeptical needs assessments, thorough architecture planning, decent system sizing, critical benchmarking and experienced load-focused system analysis are Martin's formula to success in avoiding production downtime.

Martin frequently blogs about recent issues and their solutions at http://www.usn-it.de/ and is known on Twitter as “@MartinKlierDBA”.

COMPANY

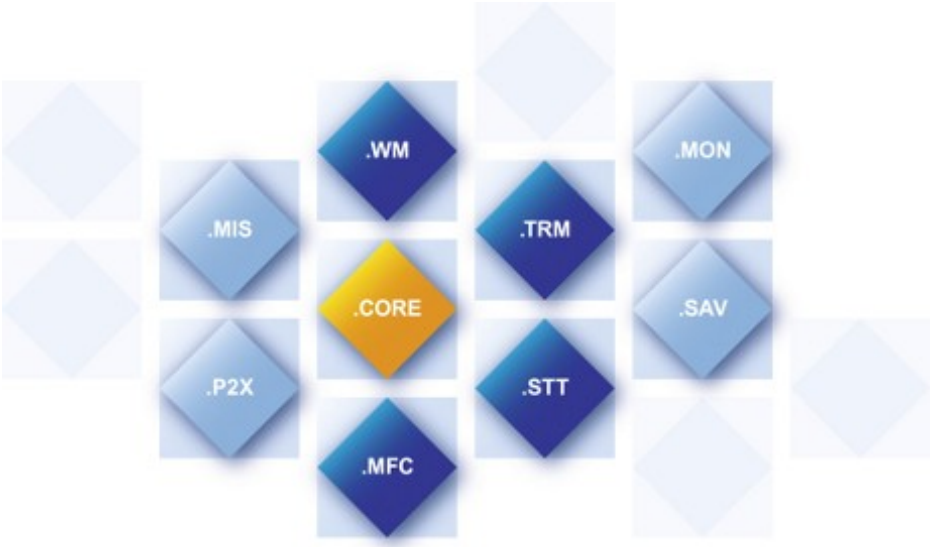
Klug GmbH integrierte Systeme (http://www.klug-is.de) is a specialist leading in the field of complex intra-logistical solutions. The core competence is the planning and design of automated intra-logistics systems with the main focus on software and system control.

CONTACT

Klug GmbH integrierte Systeme
Lindenweg 13
92552 Teunz
Germany
Phone: +49 9671 9216-0

PRODUCT

The company's product series, the “Integrated Warehouse Administration & Control System” iWACS® is a standardized concept of modules offering the tailor-made solution for the customer's individual requirements in the field of intra-logistics. With the iWACS® concept, customers do not only gain maximum flexibility, but also an optimal cost/performance ratio.



Developed with state-of-the-art software technologies, the iWACS® module .WM is an adaptable standard software granting flexibility and system stability. The warehouse management system .WM controls and organizes all tasks from goods-in to dispatch and is up to all requirements from manual warehouse to fully automated distribution center. To interlink .WM with any ERP system, customers can dispose of tried and tested standard functions.

ORACLE DATABASE

The desire of Klug GmbH integrierte Systeme is to deliver a completely rounded solution. This intent and promise includes, that Oracle databases used as an iWACS® backend are flawlessly integrated, stable and well-performing.

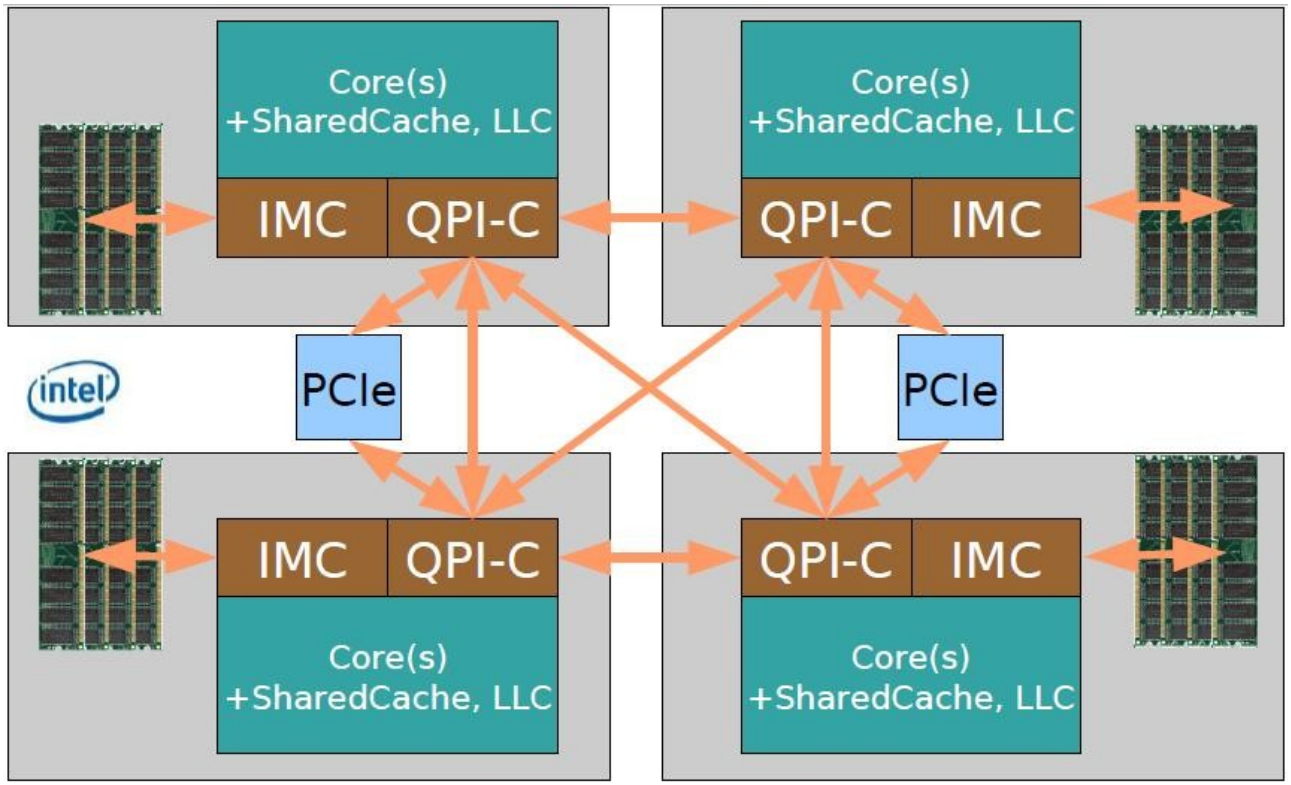
In early 2014, Klug was awarded the “Oracle Excellence Award 2013” for Independent Software Vendors (ISVs).

NUMA

BASIC

Non-Uniform memory Acces (NUMA) is a computer memory design used in multiprocessing, where typically each CPU has RAM locally attached. Accessing data in another CPU's memory (=remote), results in additional latency due to bus transfer times.

This graph shows an example layout, based on the Intel Nehalem EX architecture.

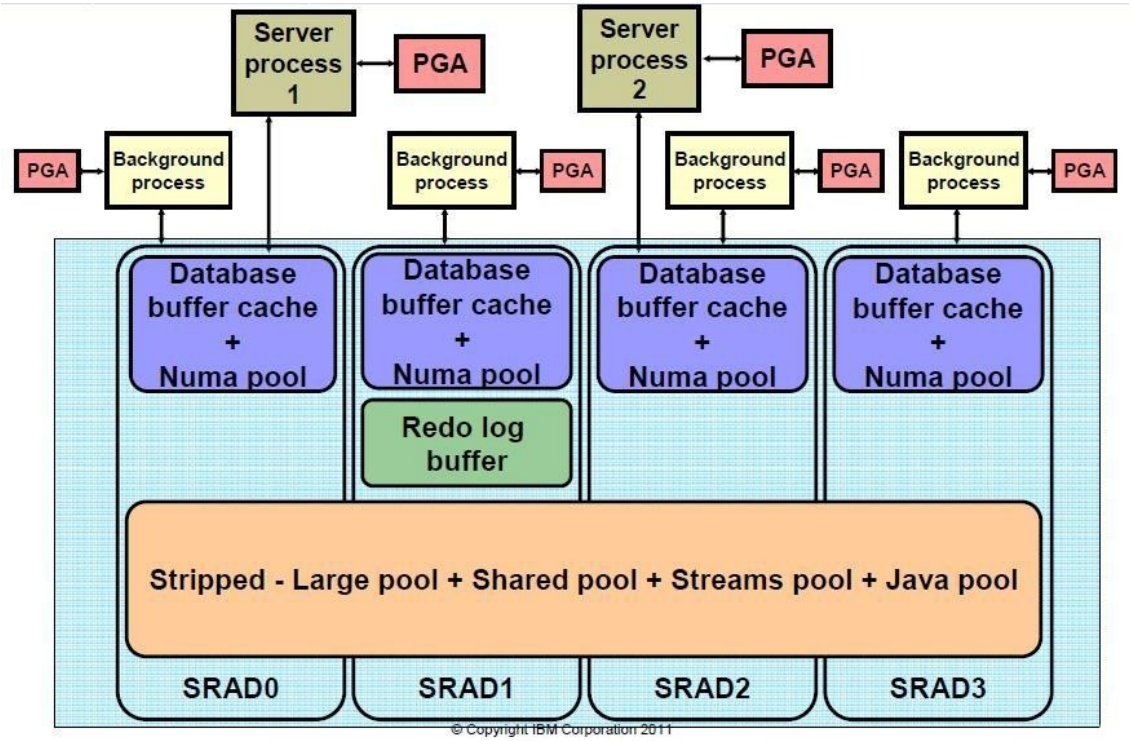


In this case, the QPI bus (“Quick Path Interconnect”) interconnects the CPUs. Each processor has an internal memory controller (IMC) to access its local RAM (in the “remote-case” on behalf of another CPU) . But last-level caches are dedicated to each CPU or core; in fact we see an example for cache-coherent NUMA, ccNUMA, here.

ORACLE AND NUMA

Oracle is NUMA aware, respectively supports NUMA with appropriate hardware and operating systems. In Oracle Database 11.2 and 12.1 we can enable working with this architecture by setting an instance parameter (“_enable_NUMA_support = TRUE”) as described in Doc ID 864633.1 at My Oracle Support.

An instance supporting NUMA will start up splitting its buffer cache between the recognized NUMA nodes. Large-, Shared-, Streams-, Java- and other pools are striped over the different local RAMs. Please seen the following IBM graph for details. It's an AIX example.



It's quite straightforward to see many possible limitations in the block diagram. One of the most obvious examples is the Redo Log Buffer. Let's follow a classical block manipulation from a Redo perspective:

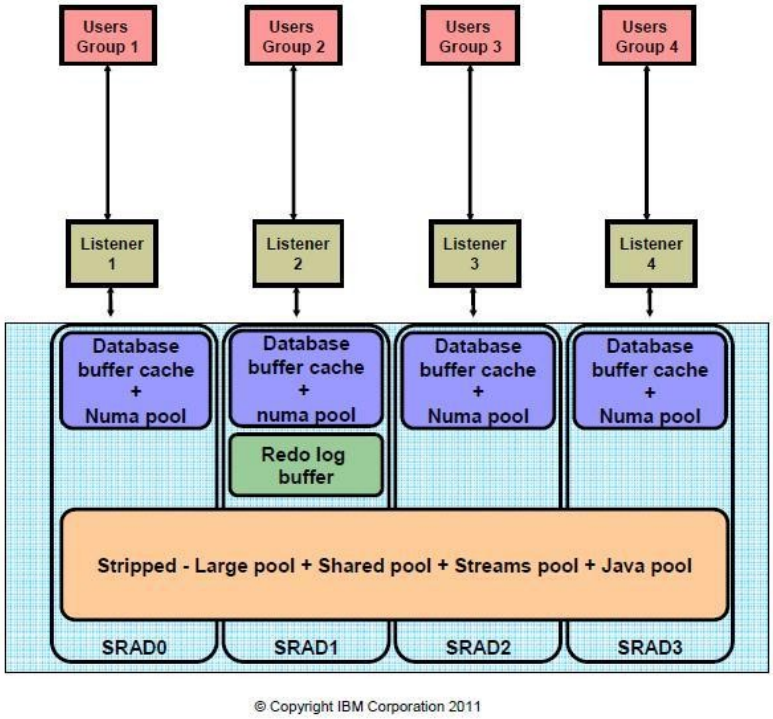
- Server Process 1 manipulates one buffer in buffer cache of node SRAD0. It's local, and quick.
- SP 1 has to write the Redo data for it into a private strand, that's in Shared Pool. Due to the striping, a statistical 3/4 of the data is written remotely and thus, slower.
- When SP 1 does its commit, data has to be transferred to the Redo Log Buffer, that's in remote node SRAD1's RAM: slow.
- During commit, the Log Writer has to read our data from Log Buffer to send them to the current Online Redo Log. If we assume that the background process attached to SRAD1 in fact IS the Log Writer, all is fine, and we have a fast local read. If not, it's a slow remote case.

What this theoretical example was meant to show is, that it's not easy to predict how things will settle after enabling NUMA support in your instance. It's necessary to check NUMA statistics, and value them carefully. In Linux, “numastat” gives absolute numbers counting upwards from last reboot. “numa_miss” or “numa_foreign” is what shows us, how many allocations failed to access locally, or how many foreign allocs happened.

```
[oracle[redacted]~]$ ([redacted]) numastat
                           node1           node0
numa_hit             2367087384      7595943305
numa_miss              57833           741376
numa_foreign           741376           57833
interleave_hit         111763           114861
local_node           2367040322      7595843897
other_node             104895           840784
```

Comparing how fast numa_miss is growing after switching Oracle to NUMA support, reveals the effect of the measure.

In theory, we should pin our users to NUMA nodes, maybe by having one listener on each node. In this case, they will fork dedicated servers on the local NUMA node, and thus, minimize buffer cache cross pooling. Another IBM graph:



REAL-WORLD-TEST

This test case was done on a live system, used for product development, in order to have a realistic long-term workload. The server has two physical CPUs, each one representing one NUMA node. The memory distribution between the nodes is visible from the graph.

```
[root@ora05 ~]# numactl --hardware
available: 2 nodes (0-1)
node 0 size: 32756 MB
node 0 free: 608 MB
node 1 size: 28672 MB
node 1 free: 1343 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

The buffer cache size is 26 GB:

```
select * from V$SGA_DYNAMIC_COMPONENTS;
```

frageergebnis

SQL | Alle Zeilen abgerufen:14 in 0,02 Sekunden

	COMPONENT	CURRENT_SIZE	MIN_SIZE	MAX_SIZE
1	shared pool	3489660928	2952790016	3489660928
2	large pool	67108864	0	67108864
3	java pool	67108864	67108864	67108864
4	streams pool	134217728	0	134217728
5	DEFAULT buffer cache	26038239232	26038239232	26709327872

When starting the instance with “_enable_NUMA_support = TRUE”, by executing “ipcs -ma” we can see that the buffer cache was split in two, as expected:

```
[root@ora05 ~]# ipcs -ma

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x740301e9 2457600    root       600        4          0
0x00000000 2752513    root       644        80         2
0x00000000 2785282    root       644        16384      2
0x00000000 2818051    root       644        280        2
0x00000000 2883588    oracle     640        4096       0
0x00000000 2916357    oracle     640        4096       0
0xed304ac0 2949126    oracle     640        4096       0
0x00000000 3735559    oracle     640        138412032  464
0x00000000 3768328    oracle     640        8388608    464
0x00000000 3801097    oracle     640        13555990528 464
0x00000000 3833866    oracle     640        13623099392 464
0x00000000 3866635    oracle     640        2684354560 464
0xc93391ac 3899404    oracle     640        2097152    464
```

After configuring the system, it was running for several days of production. But comparing between before, working without NUMA support:

node1 per second	node0 per second	
6364,2	16011,4	numa_hit
0,2	0,7	numa_miss
0,7	0,2	numa_foreign
0	0	interleave_hit
6364,2	16011,3	local_node
0,2	0,7	other_node

and after, working with NUMA support:

Node1 diff p s	Node0 diff p s	
3538,6	10179	numa_hit
0	0	numa_miss
0	0	numa_foreign
0	0	interleave_hit
3538,6	10178,9	local_node
0	0,1	other_node

there was not much of a difference. By enabling NUMA, so activating an alternative code path and

taking many risks by means of organization and possible bugs, we had no improvement. In fact, there was not even a problem to solve - less than one numa_miss per second is hardly an issue. But why did the system work well without Oracle NUMA support? Because the operating system takes care of that. Here Linux recognized a big shared memory object, and it fits into the RAM of one node. Trying to run all processes as local as possible, most of the database work is done at the node where the big buffer cache resides.

NUMA SUGGESTIONS

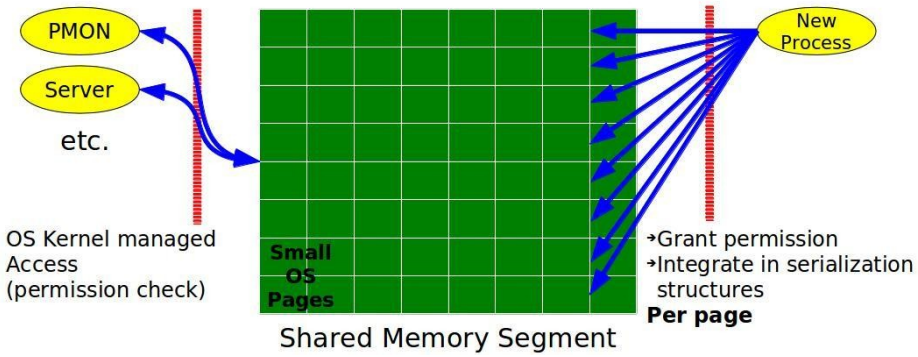
1. NUMA is mostly interesting for really big machines, for example in database consolidation systems. The partitioned approach of them, typically by user groups, applications or locations, is ideal for node-islands serving their consumers.
2. For big universal databases, it's on the edge, and needs, risks and benefits should be leveraged.
3. In any cases, NUMA support for Oracle has to be thoroughly tested, and quantified objectively.
4. Discussing the matter with the system administrator is mandatory: Both the DBA and the Sysadmin need to understand where the benefits and drawbacks are. DBAs should transfer knowledge about the application structure - usually they know a lot about it, since they are in contact with all the non-database tiers.

MEMORY PAGE SIZE

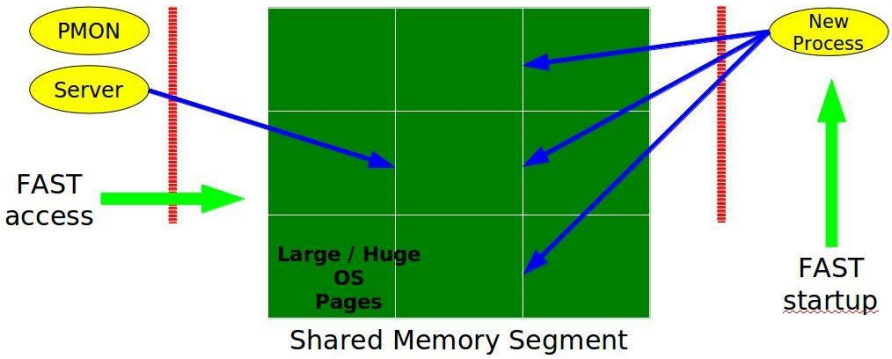
The operating system allows applications to access and allocate memory page wise. For each page access, specific system calls have to be executed.

BASIC

Memory allocation or access calls run through permission and protection checks, and have to be integrated in serialization structures. This is inevitable, but costs additional CPU cycles, and thus, time. When Oracle waits for a new process to spawn, we see “OS_THREAD_STARTUP“ in the wait interface.



One way to reduce process startup time, corresponding locks and the resulting resource consumption, is to reduce the number of pages for a given case. The need to deliver the requested amount of memory, leads to increasing the page size.

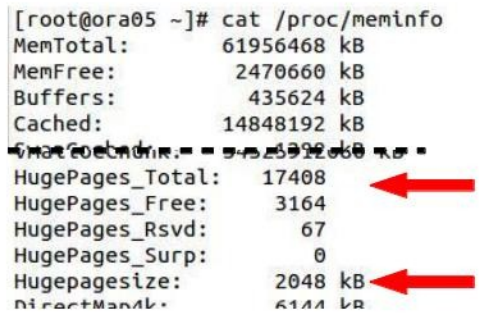


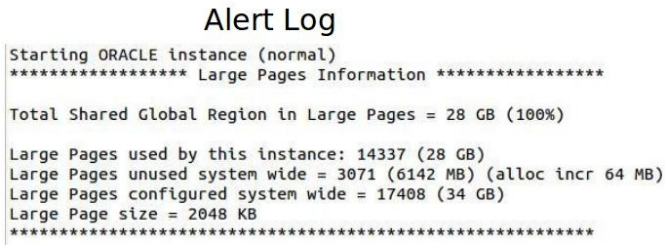
The standard page size for Linux, for example, is 4 kilobytes. With enabling large pages, the size increases to 2MB. Another example is AIX with base page sizes of 4k and 64k, and allows 16MB large pages or even 16GB huge pages. Other Unix systems have similar concepts.

A simple calculation: 64GB of SGA in 4k pages means 16.7 million page access calls. Changing the page size to 2MB means only 32 thousand pages. Thus, the overhead is reduced by factor 5,000.

REAL-WORLD-TEST

On the Linux machine some tests were done on for this paper, (working with Oracle 11.2.0.4, 32GB of SGA and approx. 1000 dedicated sessions) process start-up time reduced by 92% and OS_THREAD_STARTUP waits vanished below AWR reporting measure.





PAGE SIZE SUGGESTIONS

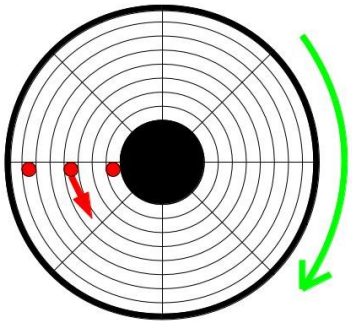
1. Larg(er) pages unfold their effect with 16GB of SGA or more, but they will do no harm on smaller setups.
2. The larger the page size, the greater the effect is. A big, but sane page size is the way of choice.
3. Some system admins don't like fumbling around here, especially if they don't have many database servers in their responsibility. But setting Large Pages up is quite straightforward. Talk them into just doing it.
4. Pre-paging the SGA is always a good idea, especially with large- / huge pages. Let the instance alloc all memory in one rush. The downside is a slightly slower start up (<30 seconds per 64GB), but on the pro side you have no more shared memory blockings for changing the size after going online with the database.

SSD STORAGE

Solid-state drives (SSD) are on the rise. They give unchallenged IO power, but are looked at sceptically when lifetime is discussed. For the DBA, they give a chance to get high performance from small servers.

SSDs and Hard-disk drives (HDD) differ in many technical aspects, but usually are connected via the same bus systems (FibreChannel, SAS or SATA).

HARD-DISK DRIVES (HDD)



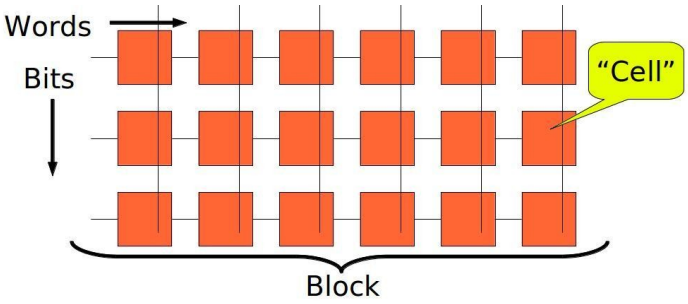
Hard-disk drives, with a rotating magnetic media, and a read/write head, have four physically limited reaction times:

1. How quick can the head move to the track?
2. How long does it take to rotate the right data under the head?
3. How long does it take to read the data?
4. How quick can I transfer the data over the controller/bus system?

Typical overall-response times are 3 to 8ms, and a bandwidth of 80-150MB/s. All values depend on how scattered the data is on the media, which means that predicting the response time is difficult.

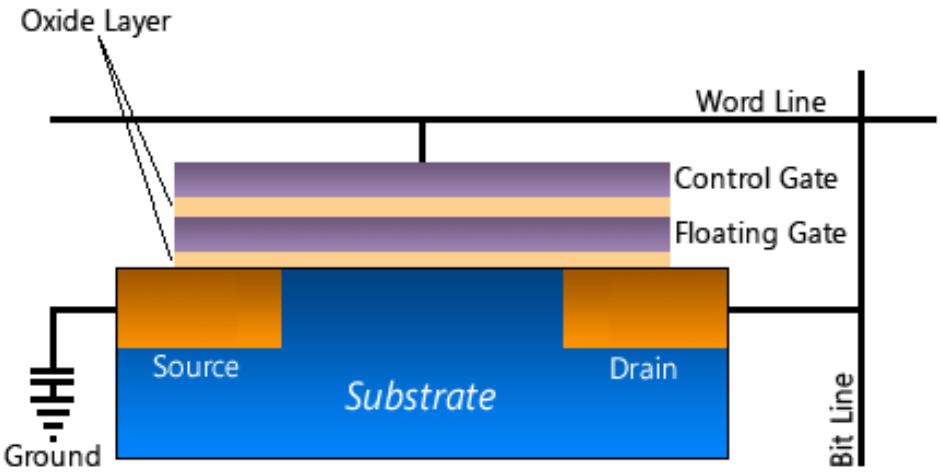
HDD lifetime is limited by mechanical deterioration and wear of the magnetic layer. Typically, the death-time of disks from the same charge and the same usage, spreads over several months or years.

SOLID-STATE DRIVES (SSD)



SSDs have a different technical approach, compared to Hard-disk drives (HDD). Data is

permanently stored in semiconductor “flash” cells. They are aligned in a grid, and allow parallel access to multiple bits at the same time, and thus, have no positioning times.

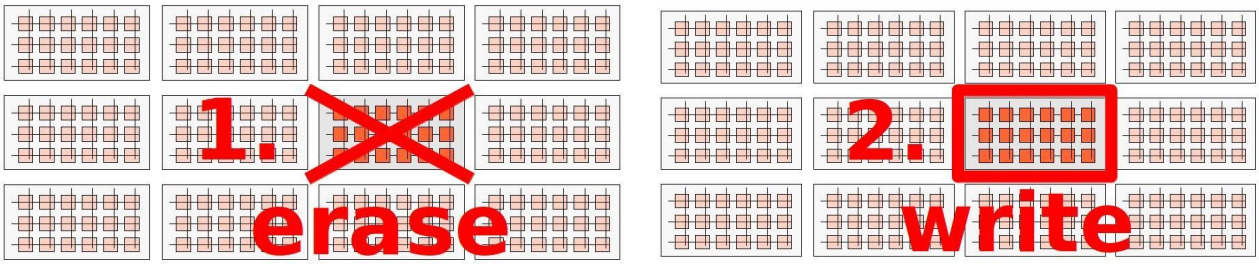


Each cell is set/reset by different voltage levels and/or currents:

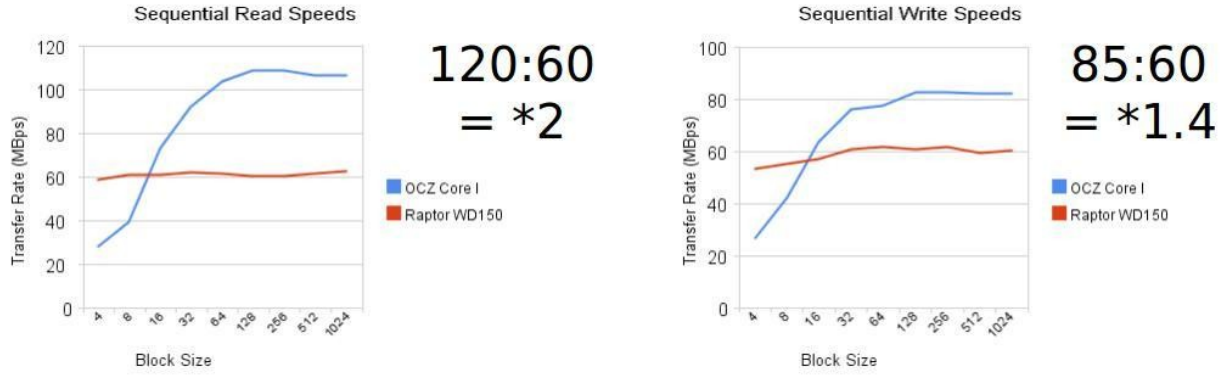
1. High voltage between Ground and the Word- / Bit Lines sets the cell to 1, tunneling hot electrons through the insulating Oxide Layer into the floating gate.
2. The possible current between Source and Drain is used to ask for the bit status without changing it. In fact, such a current is not possible with the Floating Gate having a negative potential. So “no current” means “bit is 1”
3. Grounded Word Line and high voltage on the Bit Line forces the electrons out of the Floating Gate into Drain, and sets the bit to 0. (Now S-to-D current is possible)
4. Modern flash cells can store more than one bit per cell by setting different potentials into the Floating Gate, and measuring the possible S-to-D current in a fine-grained matter.

Following this procedure, the SSD has no mechanical parts moving. But forcing the electrons through the insulating layer, wears off the Oxide, and thus, will steadily reduce the stability of the potential in the Floating gate each time the bit is set or re-set (“degeneration”). This makes the lifetime of the cell predictive, but this prediction has only a few counts of tolerance (“endurance”). Thus, SSDs of the same charge and the same workload die within short time.

DIFFERENT CHARACTERISTICS



The quite high voltage mentioned above limits the granularity of its use in the semiconductor chip. So SSDs are organized in blocks, that are always deleted and written together. Means: As soon the controller changes one bit, the whole block (16B - 512B) is copied to a cache, changed and flashed back to the medium. This, and the fact that setting the bit takes much more time than reading it, changes the load-scaling of the SSD compared to an HDD.



The comparison between an HDD and a SSD above shows, that SSDs are faster throughput-wise, when they can write bigger blocks, since this is more efficient for the block-wise writing described above.

```
ORION VERSION 11.1.0.7.0

Commandline:
-run advanced -testname oltp-write -num_disks 1 -cache_size 8192 -size_small 8 -size_large 16 -type rand -simulate raid0 -write 80 -duration 30 -matrix basic

This maps to this test:
Test: oltp-write
Small IO size: 8 KB
Large IO size: 16 KB
IO Types: Small Random IOs, Large Random IOs
Simulated Array Type: RAID 0
Stripe Depth: 1024 KB
Write: 80%
Cache Size: 8192 MB
Duration for each Data Point: 30 seconds
Small Columns:,      0
Large Columns:,      0,      1,      2
Total Data Points: 8

Name: /media/KLMHIGHSPEED/o1_mf_sysaux_4zjblvr4_.dbf    Size: 1835016192
1 FILEs found.

Maximum Large MBPS=58.51 @ Small=0 and Large=2
Maximum Small IOPS=8171 @ Small=3 and Large=0
Minimum Small Latency=0.14 @ Small=1 and Large=0
```

But for small random IO's, the SSD concept is unbeatable. Top-rated HDD's are able to deliver 140 IOPS for a random workload. A simple desktop SSD (in my example, a Samsung SSD 840 PRO) gives 8100 IOPS without a problem with 8k/16k blocks. (Tested with Oracle ORION IO benchmark.)

SSD SUGGESTIONS

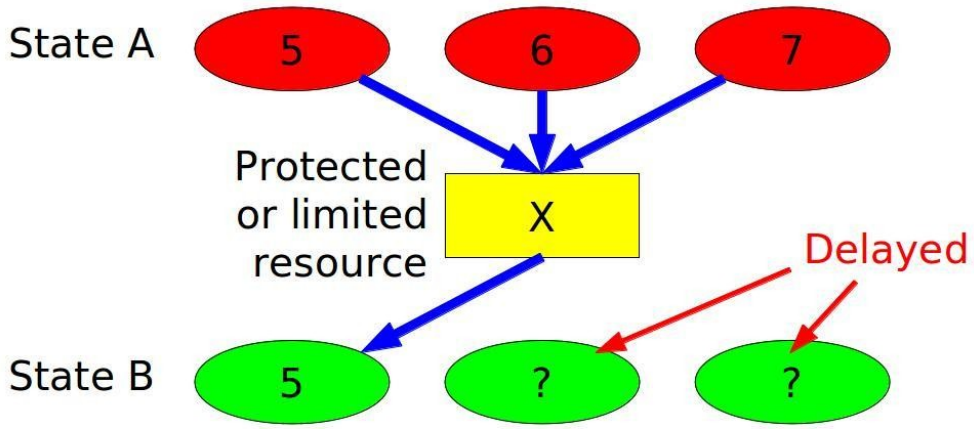
Using SSDs is ok. There are many industry-standard products with high reliability and very good performance. My suggestions for SSD in your place are:

1. Know your IO profile, from all possible sources (Oracle perspective from AWR or STATSPACK, OS perspective from nmon or other tools). Check where SSDs have the most and best effect, and play their strengths. You will have to help your system administrator with that. They usually don't know what databases need, so they tend to guessing.
2. Use enterprise-level devices in Single-level cell (SLC) format. Multi-level cells (MLC) are exponentially slower in re-writing.
3. Your sysadmin will know that: SSDs require different lifecycle handling. For reasons explained above, they die precisely after n cycles. Use S.M.A.R.T.- and SSD-aware controllers!
4. In doubt, consider and compare an array of HDDs of the same IO / throughput power.

CONCURRENCY

BASIC

Concurrency happens - that's a fact. Whenever a system has limited resources or needs to serialize for logical reasons (think: race conditions). In the first case, the serialization is natural, in the second case, a protection happens.



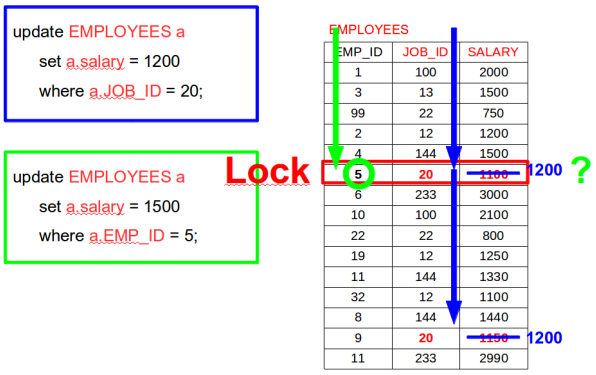
OCCURRENCE

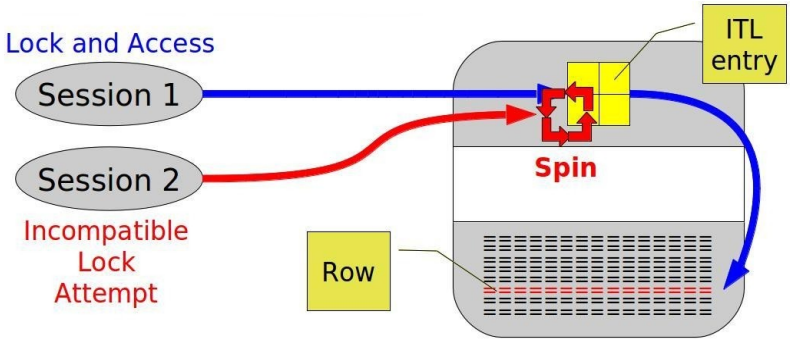
There are several situations, where concurrency occurs in a (database) system. Examples are:

- When accessing data, like row locking
- Memory protection in shared memory structures, or collisions in a block header
- Queuing for CPU, Disk IO or Network transfer

ROW LOCK

A row lock is the simplest case of a concurrency. For integrity reasons, Oracle does not allow two transactions to lock or manipulate the same row at the same time.





So when Session 1 starts manipulating the row, it creates an Interested Transaction List (ITL) entry in the block. It's valid, as long as the transaction Session 1 started, is not closed. Session 2 is interested in the same row, creates an own ITL entry, but has to check the other one repeatedly. It “spins” while contending for the right to pass along.

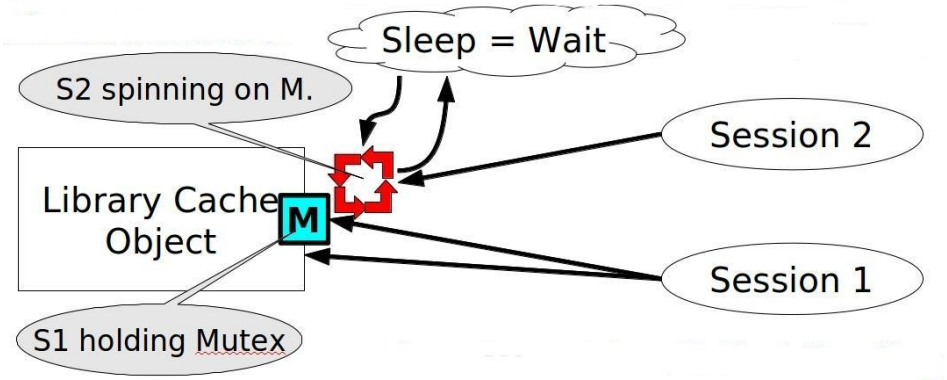
Spinning and Wait Events

Spinning means, to actively and frequently check a value in memory. This “wastes“ CPU time for non-productive work. To ease the threat of CPU starvation, and for the sake of diagnosability, Oracle only spins for a few hundred cycles, and then reports a Wait Event related to the measure that was blocked, and starts spinning again. If there was no stop in spinning, we would see just “CPU usage” without differentiation.

For the example with a row lock, the wait event would be “*enq: TX - row lock contention*”.

Mutex Waits

Mutexes are replacing more and more the latches in the database. At the moment, they usually are used to protect memory objects being under potentially high stress. This is typically the case in the Library Cache.

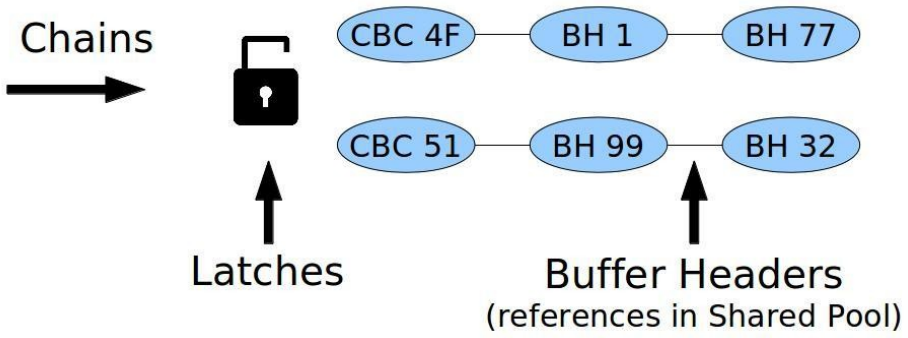


The concept is the same: One has it, the other wants it. One holds, the other spins, and reports wait events. In this case, “*cursor: mutex X/S*” and similar.

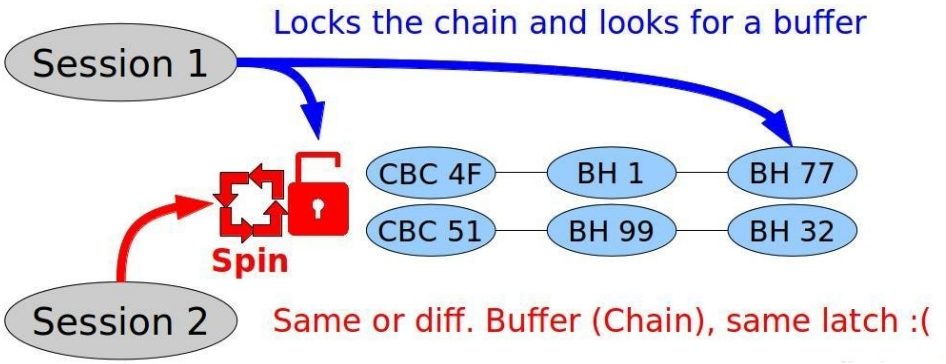
For details, and special issues with cursor sharing, see my Collaborate 2012 paper “Resolving Child Cursor Issues Resulting In Mutex Waits”.

Cache Buffer Chains

The database buffer cache is organized in “buffers”, means data blocks loaded into memory. Before accessing one buffer, it has to be clarified if it's in the cache at all, or has it to be loaded from disk? Looking through the full cache is not efficient, so Oracle introduced the concept of Cache Buffer Chains.



They are based on hash methods: The hash value of a buffer decides, to which chain the buffer will belong. As long as the asking process walks the chain, it holds a latch, protecting the chain from being accessed a second time.



When spinning/waiting for the same latch, Session 2 burns CPU, and sometimes reports “*latch: cache buffer chain*” to show us, what's going on.

Unfortunately, there are more chains than latches, so multiple chains share a serialization instrument. By the way, this also sometimes leads to false contention:

- Session 1 looks for BH77 in CBC4F
- Session 2 looking for BH99 in CBC51 is locked out, spinning.

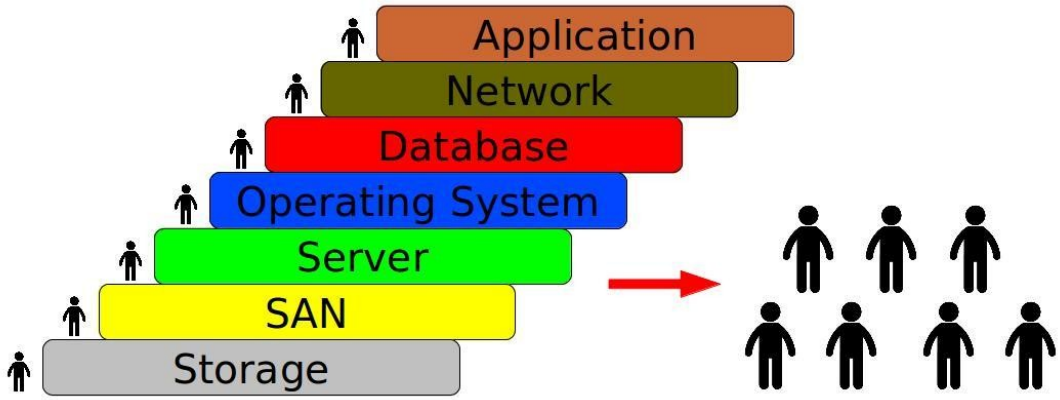
CBC suggestions

The key for the analysis of Cache Buffer Chain waits is to know what's going on. A decent diagnosis of all buffer cache access in the area.

In very rare cases, there are really “hot” blocks that are accessed intensively. But in many cases, just the sheer number of buffer gets rises the number of CBC latch collisions. In this case, the efficiency of the SQL in question is the first thing to approach.

COLLABORATE!

When talking about system design or re-design, a team comes to play. One will be the most-motivated guy who WANTS the new environment, others will understand, others will complain, and a few may block it, because they have different interests or politics in mind. This is the game, be ready for it. In the end, nothing will work if you are not able to form a team of all layers!



These are “engineers to work together” - and sometimes a manager has to be worked with, too. :)

SUGGESTIONS

Collaboration is very personal. IT only works when you become a unit. So:

1. Try to have a wider view. Understand the other layers as far as remotely possible.
2. Help the other guys or girls to understand you, make yourself clear. Don't storm them with database- or Oracle-special vocabulary. They might reciprocate accordingly!
3. All bridges are based on personal relationships. Don't act the big shot just because you are the one making the deal today. Don't try to impress, don't try to suppress. Let them tell their opinion.
4. In the end, build a solution TOGETHER. They will help to maintain it much more willingly when it's “THEIR” box as well, not “just another database server the DBA wanted this way”.
5. Avoid contentions and collisions - if there is team beer, make sure everybody can have at least one in decent time! :)

FURTHER READINGS

- Kevin Closson, on NUMA and Huge Pages
  https://kevinclosson.wordpress.com/2010/03/18/you-buy-a-numa-system-oracle-says-disable-numa-what-gives-part-i/
  http://kevinclosson.wordpress.com/2010/09/28/configuring-linux-hugepages-for-oracle-database-is-just-too-difficult-part-i/

- Craig Shallahamer, on Cache Buffer Chain visualization
  http://shallahamer-orapub.blogspot.de/2010/09/buffer-cache-visualization-and-tool.html

- Arup Nanda, on ITL / Locks
  http://arup.blogspot.de/2011/01/more-on-interested-transaction-lists.html

- Andrey Nikolaev on Mutexes
  “Exploring mutexes, the Oracle RDBMS retrial spinlocks”

- Ronan Bourlier & Loïc Fura, IBM
  “Oracle DB and AIX Best Practices for Performance & Tuning”

- My Oracle Support
  Doc ID 864633.1 “Enable Oracle NUMA support with Oracle Server Version 11gR2”
  Doc ID 1392497.1 “USE_LARGE_PAGES To Enable HugePages”
  Doc ID 361468.1 “HugePages on Oracle Linux 64-bit”