Database Systems II B: DBMS-Implementation

Chapter 7: The Buffer Cache

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2021/22

http://www.informatik.uni-halle.de/~brass/dbi21/

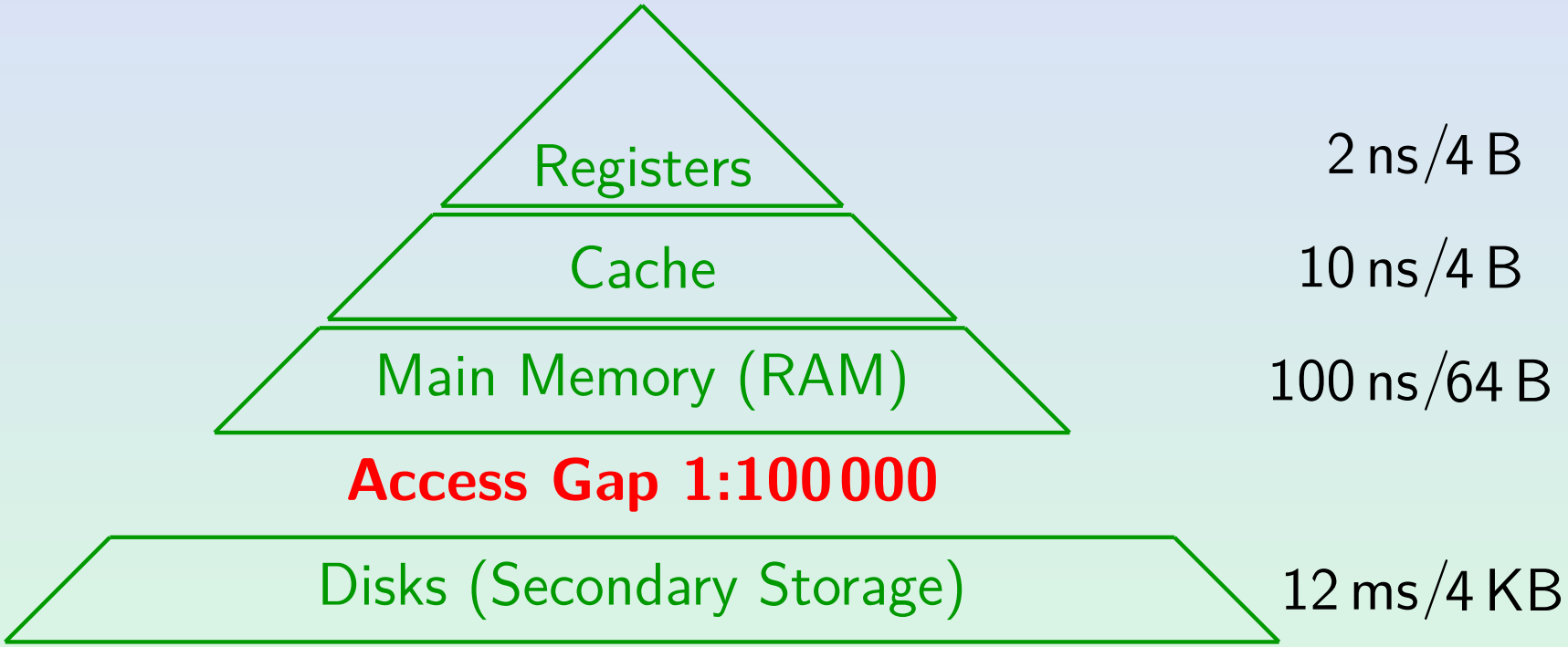
Objectives

After completing this chapter, you should be able to:

- explain the storage hierarchy and compare the characteristics of different storage media.
- explain how buffering (caching) works.
- find disk/buffer-related bottlenecks in Oracle.

Contents

Storage Hierarchy



Nearline External Memory (Tape Robot, Jukebox)

Offline External Memory (Tapes: Tertiary Storage)

Storage Characteristics (1)

- **Size**: Disks are usually much bigger than the main memory.

  PC (2021): 8 GB RAM, 1 TB disk.

  PC (2001): 256 MB RAM, 36 GB disk.

  AltaVista (1997): 6 GB RAM, 210 GB disks

  32Bit computers cannot have more than 4 GB RAM.

- **Cost**:
  - RAM is currently priced at approx. \$0.15–\$1.00 per MB,
  - Disk: \$0.002–0.01/MB (\$2–15/GB).
  - Tape (DLT Cartridge): approx. \$0.001/MB.

Storage Characteristics (2)

- **Persistence**: The contents of main memory is lost in case of a power failure or system crash.

  The disk contents is only lost in case of a headcrash etc. The mean time between failure (MTBF) is today typically 500000–1Mio h (57–114 years). But this measures only the probability of a failure when the drives are still young.

- **Operations**: In order to work with the data, they have to be brought into main memory.

  Disks allow random access, tapes only sequential access, CDs only read access, etc.

Storage Characteristics (3)

- **Granularity**:
  - In main memory, every bit can be accessed.
  - On disks, one has to read/write entire blocks (e.g. 2KByte).
- **Speed**:
  - Accessing a word in main memory costs e.g. 100 ns ($1\,\text{ns} = 10^{-9}\,\text{s}$).
  - Reading a block on a disk needs e.g. 12 ms.

The CPU can execute e.g. 500 000 instructions during one disk access.

In main memory, the time needed to access a word is constant.

On disks, the time depends on the distance.

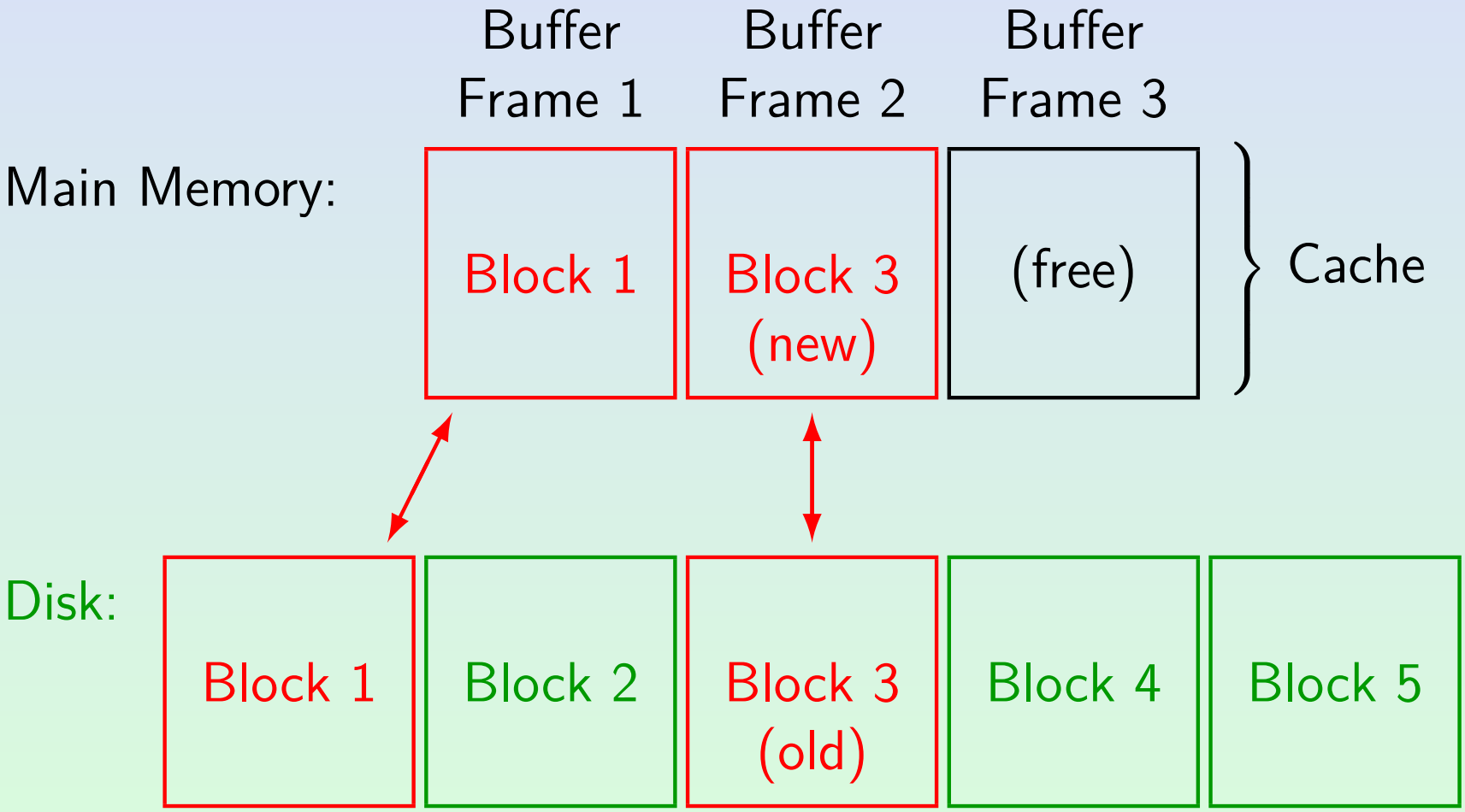
Contents

Buffering/Caching (1)

- Database blocks must be brought into main memory in order to work with them.
- The idea of buffering/caching is to keep the contents of the block for some time in main memory after the current operation on the block is done.

  Of course, if the block was modified, it might be necessary to write it back to disk. This can be delayed if other measures protect the data.
- If this same block is later requested again, it is much faster to use the copy in main memory instead of loading it again from the disk.

Buffering/Caching (2)



Buffering/Caching (3)

- The part of main memory that is used to keep copies of disk blocks is called the cache, the (disk) buffer, or the buffer cache.
- The cache is organized into pieces that can contain exactly one disk block, called (block) buffers or buffer frames.

  E.g. the block size might be 8 KB. Then each buffer frame is 8 KB large. When the cache consists of 1000 buffer frames, the cache size is 8 MB (plus some overhead for managing the cache, e.g. a table that states which disk block is contained in which buffer frame).

Buffering/Caching (4)

- If every second block request can be satisfied by using an already cached version (i.e. from main memory), the execution speed approximately doubles.

  This assumes a CPU and main memory access cost of 0, which is of course a simplification. However, because disk access is so much slower than main memory access, the result is already a relatively good approximation. If one fetched every block from the disk, the bottleneck would certainly be the disk, and the CPU would be idle for most of the time.

- In a well-tuned system, only 10% or less of the block requests really lead to a disk access.

  The remaining 90% can be satisfied from the buffer. This of course depends on the kind of queries that executed.

Buffering/Caching (5)

- One module of the DBMS software is the buffer manager (or cache manager). It gets "logical block accesses" from the upper layers of the DBMS:
  - Some of the requested blocks are contained in the cache ("cache hit"): No disk access needed.
  - Otherwise ("cache miss"), a real "physical block access" is required.
- The percentage of disk block accesses that can be satisfied from the cache is called the hit ratio. I.e. hit ratio = cache hits/(cache hits + cache misses).

Buffering/Caching (6)

- Of course, when the DBMS has just been started, the hit ratio is 0%, because the cache is still empty: Every logical block access leads to a physical block access.
- However, after some time there might have been 1000 logical block accesses and only 200 physical ones. Then the hit ratio is 80%.
- Depending on the author, good hit ratios are 80%, 90%, 95% (for normal OLTP databases).

Buffering/Caching (7)

- Normally the database is much bigger than the main memory. Therefore not all blocks can be kept in the buffer cache.
- E.g. suppose that the DB consists of 1 million blocks (8 GB), and the buffer cache consists of only 10000 blocks (80 MB).
- If the block accesses were randomly distributed, the hit ratio would be 1%. Then the possible speedup would be 1% or less, which is not worth the effort.

Buffering/Caching (8)

- But normally, a small part of the database is accessed very often, and a large part of the database only seldom.

  An 80-20 rule applies to many things in the real world (Pareto principle). For database block accesses, it would mean that 80% of the block accesses go to 20% of the blocks. However, this would still not allow effective caching. Often, the distribution is much more uneven.

- When benchmarking a DBMS or measuring query runtimes, one must respect the cache: When the same query is executed for a second time, it usually runs much faster.

A Typical Buffer Manager (1)

- The following slides explain how the "Buffer Manager" module inside a DBMS might work.

  This is a hypothetical textbook DBMS. I believe that also Oracle basically works this way, but Oracle of course does not publish such internal details.

- This is mainly interesting for people who need to implement a DBMS.

  E.g. the Oracle development staff or students who want to work in my deductive database project.

A Typical Buffer Manager (2)

- The procedures explained on the following slides are called by the DBMS layers above the buffer manager.
- They are internal to the DBMS. The DBA or database user does not (and cannot) directly call them.

  Of course, when queries are executed, the DBMS software calls these procedures on behalf of the user.

- However, in order to do performance tuning, it is good (or even necessary) to have some understanding how the DBMS works internally.

A Typical Buffer Manager (3)

Procedure "Pin Block x":

- Determine whether block x is already in a buffer frame (i.e. in the cache).

  There probably is a hash table to quickly find the block.

- If yes (cache hit):
  - Return the memory address of this buffer frame.
  - Make sure that the block will remain in the buffer frame.

    I.e. is "pinned" there. The caller wants to work with this block in memory, and it would be fatal if suddenly the buffer frame is overwritten with another block. The caller will tell the buffer manager with "unpin" that he/she is finished.

A Typical Buffer Manager (4)

Procedure "Pin Block x", Continued:

- If the block is not in the cache (cache miss):
  - Get an empty buffer frame.

    Normally there is no empty buffer frame. Then an occupied one must be selected and made free, see "Replacement Strategy" below.
  - Call the disk manager to read the block into this buffer frame.
  - Return the memory address of the buffer frame.

    Again, the block must be pinned in the buffer frame.

A Typical Buffer Manager (5)

Procedure "Unpin Frame x" (block was not changed):

- The caller is done with this block (for the moment).
- Therefore, its buffer frame can be used for another block.

  Of course, since it is possible that the same block is requested again, it should not be immediately removed from the buffer frame — only when space is needed.
- Since the buffer frame was not changed, the disk still contains the same version.
- Thus, when space is needed, the buffer frame may simply be overwritten with another block.

A Typical Buffer Manager (6)

Procedure "Unpin Frame x" (block was changed):

- If the caller has changed the block, the version in the buffer frame is newer than the version stored on the disk.
- So before this buffer frame is reused, its contents must be written back to the corresponding block on the disk.

  The buffer is called "dirty" in this case.

A Typical Buffer Manager (7)

Delayed Writing:

- Writing a modified block back to the disk does not have to happen immediately.
- The persistence of the transaction is normally ensured via a different mechanism (log file).

  The log file is a transcript of all changes. Actually, before the buffer manager can write the block back to the disk it must ensure that the undo information was written to the log file (in Oracle via the rollback segments). This is called the WAL principle (write ahead log). If the system should crash, it must be possible to undo all changes by transactions that were not yet finished (committed). The easiest way to do this is by not writing back modified blocks that contain changes by unfinished transactions, but this has other problems.

A Typical Buffer Manager (8)

Multiple Pins for One Block:

- It is possible that multiple clients request the same block at the same time.

  In this case, one can use a "pin counter" which is incremented for every pin and decremented for every unpin. Only when this counter becomes 0, the block can be removed from the buffer frame.

- Of course, it must be avoided (by means of locks) that other processes access the block while one process changes it.

  Note that these locks should be held only for a very short time. They are something different than the locks which a transaction holds on a changed row until the commit.

Replacement Strategy (1)

- The first n requested blocks are loaded into the n available buffer frames. After that, all buffer frames are always "full".

  There is no advantage removing a block from the buffer without need (unless we know that it is not used again).

- Thus, when a new block must be loaded, a victim is selected among the blocks already in the buffer (by the "replacement strategy").
- This block is removed from the cache, and the new block is loaded into the same buffer frame.

Replacement Strategy (2)

- Note that if the block to be removed from the buffer was changed, it must be saved first.
- The system should save modified blocks from time to time so that there are sufficiently many buffer frames available which can simply be overwritten when needed.

  Oracle has one or more background processes "DB Writer" (DBW0, DBW1, etc.) for this purpose.

Replacement Strategy (3)

- Normally, a "least recently used" (LRU) strategy is used:
  - Whenever a block becomes unpinned, its buffer frame is entered into a queue (at the rear).
  - When a buffer frame is needed, the one at the front is taken.
  - If a block is pinned again while it is in the queue, it is removed from the queue.
- If all buffers are pinned, the caller must wait.

Exercise

- Suppose there are 3 buffer frames and 10 disk blocks. What actions does the buffer manager perform for these requests:
  - Pin block 1, unpin block 1 (not changed)
  - Pin block 2, unpin block 2 (changed)
  - Pin block 1, unpin block 1 (not changed)
  - Pin block 5, unpin block 5 (not changed)
  - Pin block 8, unpin block 8 (changed)
  - Pin block 1, unpin block 1 (not changed)
  - Pin block 8, unpin block 8 (changed)

Sequential Flooding of the Buffer (1)

- Suppose that the DBMS has n buffer frames and needs to read a table stored in $n + 1$ data blocks multiple times.

  E.g. for a nested loop join.

- Then the LRU strategy makes the buffer useless: A block is forced out of the buffer immediately before it is needed again.

  This problem is called "sequential flooding of the buffer". LRU is one of the worst possible replacement strategies here: Although we have n buffers, no block is buffered long enough to be accessed again from the buffer. If the table had $\leq n$ blocks, it would be read only once, and all following requests could be answered out of the buffer.

Sequential Flooding of the Buffer (2)

- One possibility to avoid this behaviour is zig-zag reading (used by some DBMS):
  - The first pass through the table is done forward,
  - the second pass backward,
  - the third pass again forward, etc.
- Oracle puts blocks read in long full table scans normally at the front of the LRU queue, so they are immediately reused.

DBMS vs. OS (1)

- Modern operating systems have virtual memory, which works quite similar to the decribed buffering scheme.
- So why repeat parts of the operating system in the DBMS?
- Operating systems are not very good in supporting the specific needs of DBMS, although they often do *nearly* the same thing.
- In the future there will be combined OS/DBMS.

DBMS vs. OS (2)

- In the OS, the file used for paging is initialized during every startup — it cannot be used it as the persistent database.

  One could of course request enough main memory to read every block from the DB into memory. But this would store most blocks two times on the disk: In the DB file and the swap file.

- On a 32bit-machine, virtual memory is limited to 4 GB. Databases can be terabytes large.

- Operating system calls take normally quite long. But pin/unpin are called very often.

  Since a block should be kept pinned only for a short time.

DBMS vs. OS (3)

- The DBMS might have information about future references to a block, which can be utilized in the replacement strategy.

  Also prefetching of blocks, e.g. in a sequential scan, is very effective.

- The buffer frames of the DBMS should be in real memory.

  If it should happen often that buffer frames are paged out ("double paging"), the best replacement strategy becomes useless. Choose a smaller number of buffer frames and do not run other memory-intensive processes on this machine.

Contents

Performance Monitoring (1)

- For performance tuning, the bottlenecks of the system must be found (i.e. the performance problems must be located).

  E.g. it is useless to increase the cache if it performs well.

- In Oracle, a lot of statistical information is available in the V$*-tables, e.g. V$SYSSTAT.

  The V$*-tables are called the "Dynamic Performance Views". They give access to data structures inside the Oracle server. They are not stored tables. Of course, the V$*-tables can only be accessed by the DBA.

Performance Monitoring (2)

- V$SYSSTAT contains 226 different performance related numbers (counters, average times, etc.). Its columns are:
  - STATISTIC#: Identifying number of the statistic.
  - NAME: Symbolic name of the statistic.
  - CLASS: Bit pattern to classify the statistic.

    E.g. all cache-related statistics have the third bit (8) set.
  - VALUE: The value of the statistic.

Performance Monitoring (3)

- E.g., this query prints the number of data blocks that were physically read since system startup:

```
SELECT VALUE
FROM   V$SYSSTAT
WHERE  NAME = 'physical reads'
```

- The Oracle server maintains a counter that is initialized to 0 when the system is started and incremented each time a block is read from disk.
- There are many different such counters.

  Some statistics are available only when the initialization parameter `TIMED_STATISTICS` is set to `TRUE` (because they cause some overhead).

Performance Monitoring (4)

- In addition, there is a table V$SESSTAT that contains statistics for each session. Columns are:
  - SID: Session identifier (more info in V$SESSION).
  - STATISTIC#: Identifying number of the statistic.
  - VALUE: The value of the statistic.
- Here, a join with V$STATNAME is necessary in order to decode the statistic numbers.

  V$STATNAME lists all available statistics, it has the columns STATISTIC#, NAME, CLASS. Some of the statistics are only meaningful in V$SYSSTAT, others only in V$SESSTAT.

Performance Monitoring (5)

- Two scripts in "`$ORACLE_HOME/rdbms/admin`" can be used to print a report containing many statistics:
  - `utlbstat.sql` (begin statistics) records the current values of the statistics counters.

    The scripts are executed with SQL*Plus. They log in as `INTERNAL`.

    It might be necessary to belong to the OS user group "`dba`".
  - Then there should be normal production usage of the DBMS for some time.
  - `utlestat.sql` (end statistics) computes the differences of the then current values with the stored ones and generates a report (in `report.txt`).

Performance Monitoring (6)

- SQL*Plus shows a few statistics for each executed query after

  `SET AUTOTRACE ON`

- In addition, the query execution plan is shown.

  `SET AUTOTRACE ON STATISTICS` shows only the statistics, `SET AUTOTRACE ON EXPLAIN` only the execution plan. Try also `SET TIMING ON`. If a user has rights on a view, but not the base tables, the execution plan is not shown. Before one can see the execution plan, a table for storing information about that plan must be created by executing the script `$ORACLE_HOME/rdbms/admin/utlxplan.sql`. Before a user can see the statistics, the DBA must grant the role `PLUSTRACE` to that user. The role is created with the script `$ORACLE_HOME/sqlplus/admin/plustrce.sql`.

Buffer Performance (1)

- In Oracle, the number of cache misses is the value of the counter "`physical reads`".
- The total number of requests are the sum of two statistics values (this sum is called "logical reads"):
  - `consistent gets`: Requests for block versions that contain only changes that were committed before the query started.
  - `db block gets`: Requests for the current version of a block.

Buffer Performance (2)

- In Oracle, the hit ratio is computed as:

$$\frac{\texttt{consistent gets} + \texttt{db block gets} - \texttt{physical reads}}{\texttt{consistent gets} + \texttt{db block gets}}$$

- Exercise: Write an SQL query for this.
- The hit ratio should be above 90% or 95%.

  At least for OLTP (online transaction processing) applications.

- If the hit ratio is below 60%, 70% or 80%, the buffering is not working well and something should be done.

Buffer Performance (3)

- E.g., in order to improve the hit ratio, it might be possible to increase the initialization parameter `DB_BLOCK_BUFFERS` (number of buffer frames).

  Total buffer memory: `DB_BLOCK_BUFFERS * DB_BLOCK_SIZE`.

- It is important that the entire SGA (system global area, includes the cache) remains in real memory.

  If the increase of the number of buffer frames leads to paging on the operating system level (i.e. "virtual memory" is used), the situation is worse than before ("double paging").

- If necessary, more memory must be bought.

Buffer Performance (4)

- However, before one tunes the buffer cache, there are many other things to check and improve.
- E.g., indexes might reduce the number of accessed disk blocks. Then the hit ratio will improve without adding more buffer frames.

  Oracle therefore recommends a specific sequence for tuning:
  Business rules, data design, application design, logical DB structure, DB operations, access paths, memory allocation, I/O and physical structure, resource contention, OS/hardware.

Buffer Performance (5)

- The hit ratio can also be improved by caching only blocks from certain tables.

  E.g., if blocks from a very large table are accessed at random, they do not profit from the valuable buffer space, but push other blocks out of the cache.

- Besides the `DEFAULT` buffer pool, Oracle can manage two other buffer pools: `KEEP` and `RECYCLE`.

- One can distribute the available buffer frames between these three buffer pools and assign database objects to a specific buffer pool:

```
CREATE TABLE ...(...)
STORAGE(BUFFER_POOL KEEP)
```

Buffer Performance (6)

- Oracle normally places blocks read during a full table scan at the front of the LRU queue, so that the buffer frames are immediately reused.
- For small lookup tables one should request that they are even if read in a full table scan:

```
CREATE TABLE EXERCISES (CAT CHAR(1), ...)
                TABLESPACE USER_DATA
                CACHE
```

Small tables are nearly always read in full table scans.

See also: V$BUFFER_POOL, V$BUFFER_POOL_STATISTICS, V$BH.

The Five Minute Rule (1)

- Sometimes disks must be bought not because more disk space is needed, but because more accesses per second are required.
- In such a situation, caching can save not only time, but also money (see next slide).
- This rule was originally known as the five minute rule: If a block is accessed every five minutes, it should remain in the cache.
- However, as technology advanced, it is now really the ten minute rule.

The Five Minute Rule (2)

- Simplified/Naive calculation:
  - A disk costs about \$200 and allows e.g. 70 accesses per second. Suppose that each access for 8KB (in average).
  - If the data is accessed only every ten minutes from the disk, it costs $\$200/(70 * 600) = \$0.005$.
  - So it is still cheaper to buy 8KB more buffer (8KB RAM cost \$0.004 if price/MB is \$0.50).

Disk Performance (1)

- V$FILESTAT contains performance statistics for each file. It has the following columns (continued below):
  - FILENO#: File number.

    V$DATAFILE relates FILENO# and NAME.
  - PHYRDS: Number of read operations for this file.

    Physical reads, i.e. real reads (not from buffer).
  - PHYWRTS: Number of write operations for this file.
  - PHYBLKRD: Number of blocks read.

    PHYBLKRD can be larger than PHYRDS since sometimes a chunk of several consecutive blocks is read in one call.
  - PHYBLKWRT: Number of blocks written.

Disk Performance (2)

- Columns of V$FILESTAT, continued:
  - READTIM: Time spent in reading (in 1/100s).

    Timing information is collected only when the initialization parameter TIMED_STATISTICS is TRUE.
  - WRITETIM: Time spent in writing.
  - AVGIOTIM: Average time for an I/O operation.
  - LSTIOTIM: Time for last I/O operation.
  - MINIOTIM: Minimum time for an I/O operation.
  - MAXIOWTM: Maximum time for a write operation.
  - MAXIORTM: Maximum time for a read operation.

References

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Edition. Section 5.1–5.4.
- Ramakrishnan/Gehrke: Database Management Systems, 2nd Ed. Section 7.1, 7.2, 7.4.
- Härder/Rahm: Datenbanksysteme, Konzepte und Techniken der Implementierung.
- Mark Gurry, Peter Corrigan: Oracle Performance Tuning, 2nd Edition (with disk).
- Oracle 8i Designing and Tuning for Performance, Release 2 (8.1.6), Oracle, 1999.
- The PC Guide: Hard Disk Performance
  http://www.pcguide.com/ref/hdd/perf/index.htm
- Storage Review: http://www.storagereview.com, http://198.76.30.88/jive/sr/
- Gray/Putzolu: The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time. Proc. of SIGMOD'87, Pages 395–398.
- J.N. Gray, G. Graefe: The Five Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb. ACM SIGMOD Record 26:4, 1997, pages 63–68.
- Patterson/Keeton: Hardware Technology Trends and Database Opportunities. SIGMOD'98.
  http://www.cs.berkeley.edu/~pattrsn/talks/sigmod98-keynote-color.pdf