6.830/6.814 — Notes* for Lecture 7: Buffer Management

Carlo A. Curino

September 30, 2010

1 Announcements

- Lab 2 is going to go out today... Same as before... do not copy! (I mean it!)
- Project teams were due last time. Anyone still without a team?

2 Readings

For this class the suggested readings are:

- Hong-Tai Chou and David DeWitt. An Evaluation of Buffer Management Strategies for Relational Database Systems. VLDB, 1985.
- If you are interested in simple current implementation of bufferpool management: `http://dev.mysql.com/doc/refman/5.5/en/innodb-buffer-pool.html` and

3 Recap

We are given many access methods. None of this is uniformly better than others across the map, but each has some particular case in which is best. B+Trees (both clustered and unclustered) and Heapfile are the most commonly used.

We made a big case about the fact that Disk accesses are very expensive and that the DBMS has 2 ways to reduce their impact on performance: i) reduce the number of disk accesses by having smart access methods, ii) do a good job at caching in RAM data from disk.

*These notes are only meant to be a guide of the topics we touch in class. Future notes are likely to be more terse and schematic, and you are required to read/study the papers and book chapters we mention in class, do homeworks and Labs, etc.. etc..

Last lecture we assumed every access was off of disk, and we tried our best to minimize the number of pages accessed, and to maximize the sequentiality of the accesses (due to the large cost of seeks) by designing smart access methods. Today we get into the investigation of "what" to try to keep in RAM to further avoid Disk I/O. The assumption is that we can't keep everything, since the DB is in general bigger than the available RAM.

4 Today's topic

Why don't we just trust the OS?

(DBMS knows more about the data accesses, and thus can make a better job about what to keep in RAM and what to pre-fetch, given an execution plan is rather clear what we are going to need next).

DBMS manages its own memory: *Buffer management / Buffer pool.*

Buffer pool:

- cache of recently used pages (and more importantly plans ahead of which one are likely to be accessed again, and what could be prefetched)
- convenient "bottleneck" through which references to underlying pages go useful when checking to see if locks can be acquired or not
- shared between all queries running on the system (important! the goal is to globally optimize the query workload)

Final goal is to achieve better overall system performance... often correlated to minimize physical disk accesses (e.g., executing 1 query at a time guarantees minimum number of accesses, but lead to poor throughput).

Good place to keep locks:

pg id	lock	ptr
1	R/W, TID 2	0xABCD
2	...	0xCDEF

Cache – so what is the best eviction policy?

USE SAM NOTES FROM HERE ON...

5 What about MySQL?

(From MySQL documentation online: `http://dev.mysql.com/doc/refman/5.5/en/innodb-buffer-pool.html`)

A variation of the LRU algorithm operates as follows by default:

- 3/8 of the buffer pool is devoted to the old sublist.
- The midpoint of the list is the boundary where the tail of the new sublist meets the head of the old sublist.
- When InnoDB reads a block into the buffer pool, it initially inserts it at the midpoint (the head of the old sublist). A block can be read in because it is required for a user-specified operation such as a SQL query, or as part of a read-ahead operation performed automatically by InnoDB.
- Accessing to a block in the old sublist makes it ?young?, moving it to the head of the buffer pool (the head of the new sublist). If the block was read in because it was required, the first access occurs immediately and the block is made young. If the block was read in due to read-ahead, the first access does not occur immediately (and might not occur at all before the block is evicted).
- As the database operates, blocks in the buffer pool that are not accessed ?age? by moving toward the tail of the list. Blocks in both the new and old sublists age as other blocks are made new. Blocks in the old sublist also age as blocks are inserted at the midpoint. Eventually, a block that remains unused for long enough reaches the tail of the old sublist and is evicted.

You can control:

- `innodb_old_blocks_pct` for the portion of new-old
- `innodb_old_blocks_time` Specifies how long in milliseconds (ms) a block inserted into the old sublist must stay there after its first access before it can be moved to the new sublist.

6 LRU Cache misses in typical scenarios

From the paper: *I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks An Analysis at the Logical Level* by Windsor W. Hsu, Alan Jay Smith, and Honesty C. Young. We report the LRU miss ratios for increasingly large bufferpool sizes, and for typical production databases and popular benchmarks. This should give you an idea of the fact that some portion of the DB are very "hot" while other are rather "cold", thus throwing more and more RAM at the problem will provide less and less returns. On the other side choosing the "right" things to keep in RAM is clearly vital!

6.830 / 6.814 Database Systems

Fall 2010