Philipps University of Marburg
Department of Mathematics and Computer Science
Database Research Group

Wintersemester 2024/2025
11. 11. 2024

Prof. Dr. Bernhard Seeger
Amir El-Shaikh, M.Sc.
Marius Kuhrt, M.Sc.

Exercises for the lecture
Implementation of Database Systems

Submission: 15. 11. 2024,
no later than 8:00 p.m.
via the ILIAS platform

Exercise 2

Task 2.1: 2Q Replacement Strategies (3+1+5+6) **(15 Points)**

In the lecture you got to know the LRU-k strategy. In practice, $k = 2$ already achieves very good results. In this task you're required to implement a replacement strategy that is easier to implement than LRU-2, but still achieves similarly good results.

The 2Q strategy is based on two lists. The first list (A1) is managed using the FIFO strategy, the second (Am) using the LRU(-1) strategy. First, it is checked whether a requested page is already managed in one of the lists. If this is not the case, the page is sorted at the beginning of A1. If the list is already full, the oldest page is removed from A1 (A1 is a classic queue). If the page is in A1, it is moved to the beginning of Am.
Furthermore, you may find the full article on the 2Q procedure on the ILIAS platform.

We use the `Buffer` class to implement the replacement strategy. A buffer uses the (abstract) method `Buffer.Slot::victim()` to determine which page is to be swapped out next. Pages can be requested in the buffer by calling `fix()`. If there is no free frame left, the page returned by `victim()` is swapped out of the buffer.

a) First extend the class `Buffer` so that the current page fault rate can be queried. The page miss rate is the ratio of pages not in the buffer to requested pages. To do so, implement an abstract class `PageFaultRateBuffer` that inherits from `Buffer`. `PageFaultRateBuffer` maintains two counters: one for the total number of page accesses, and a second for the number of unbuffered page accesses. Furthermore, a method `getPageFaultRate()` is to be implemented that returns the current page fault rate.

Hint: The method `fix()` should be overridden accordingly.

b) Extend the class `LRUBuffer` to `PageFaultRateLRUBuffer` by adding the measurement of the page fault rate (similarly to part **a)**).

c) Create a class `SimpleTwoQueueBuffer` that inherits from `PageFaultRateBuffer` with the following data fields:

- `int kin`, where the size of `kin` is 25% of the total number of slots.
- `Queue<Slot> a1`.
- `Queue<Slot> am`.

The (simple) 2Q strategy's `victim()` method returns the page (slot) to be swapped out based on the lists `am` and `a1`. Hence, the 2Q strategy should be implemented using the `victim()` and `fix()` methods respectively.

d) The method implemented in part **c)** does not consider correlated access. However, this can be achieved by splitting `a1` into two FIFO queues. To do so, create a class `TwoQueueBuffer` that inherits from `PageFaultRateBuffer`. Now, instead of one queue `a1`, `TwoQueueBuffer` has two queues, `Queue<Slot> a1in` and `Queue<char> a1out`, plus a field `int kout`. The queue `a1in` maps the correlation time. As long as a requested page is in `a1in`, accesses are considered correlated. As soon as a page from the `a1in` queue is swapped out, a reference to it is inserted into the `a1out` queue. If a page in the `a1out` queue is referenced, it will be moved to the `am` queue. Implement the replacement strategy described above, taking into account the notes on part **c)**.

Test the buffers created in the previous parts of the exercise by simulating the scenario from task 2.2 and output the page fault rates for `LRUBuffer`, `SimpleTwoQueueBuffer` and `TwoQueueBuffer` on the console, respectively.

<u>Task 2.2</u>: Key-Value Stores vs Relational Databases (2+5+6+2) (15 Points)

In this task, the classes `Customer`, `Order` and `Product` provided in the code template in Ilias are to be persisted on the external storage using a key-value store and a relational database.

a) Create methods in the provided class `ShopGenerator` to generate random objects of the classes `Customer`, `Order` and `Product`. Make sure that the respective IDs are unique. When `Customer` is created, it should have a random, upper bounded number of `Order` objects (but at least one!). Created `Order` objects should also manage a random, upper bounded number of `Product` (but at least one!).

b) Complete the class `KVStoreImpl`, which, among other things, implements the `CustomerStore` interface using the `jdbm` library and persists `Customer` objects on the external storage using a `PrimaryHashMap`. Use the `customerId` as a key. Use the class `KVExample` as a guide for dealing with jdbm.

c) Complete the class `H2StoreImpl`, which, among other things, implements the `CustomerStore` interface using a `H2` database and persists *all* information of a `Customer` object on the external storage. The relational schema can be found in the `createTables` method. Use the `H2Example` class as a guide to dealing with `H2`.

d) Complete the `InsertionPerformanceTest` class and insert `10,000` randomly generated Customer elements into an instance of `KVStoreImpl` and an instance of `H2StoreImpl` in a `main` method. Measure the time it takes to insert all elements. Document and explain your results as a comment in the code or in a PDF file.

Note: The following tasks are optional but *highly* encouraged.

Task 2.3: Locality **(0 Points)**

The following is a reference string of a transaction:

`AABBECFCCCDAAFFGLGGIMIMIMEEFGEF`

Determine:

a) the current locality for `t=5, t=15, t=22` and window size `6`.

b) the average locality for window size `6`.

c) the LRU stack depth distribution. Create a diagram containing the number of accesses of each position in the stack.

Task 2.4: Replacement Strategies **(0 Points)**

The following is a reference string of a transaction:

`ABAABAACDBACED`

Assume a buffer with `3` frames. Sketch the modifications of the buffer for each of the following replacement strategies:

a) FIFO

b) LFU

c) LRU

d) CLOCK

Additionally, provide the page fault ratio for each strategy.

Task 2.5: LRU-2 **(0 Points)**

The following is a reference string of a transaction:

`BAABCDBEBAACDBE`

Assume page accesses happen at a rate of `1` per second, the database buffer holds four frames and uses the LRU-2 replacement strategy. The correlation time is `3` seconds. Sketch the status of the buffer and its structures `HIST(p, i)` and `LAST(p)` on each access of the transaction.