Prof. Dr.-Ing. Dr. h. c. T. Härder
Computer Science Department
Databases and Information Systems
University of Kaiserslautern

Exercise 3– Solution proposal

Documentation of the lecture:
„http://wwwlgis.informatik.uni-kl.de/cms/courses/realisierung/“
(May 25, 2011, 3.30 pm, 36-336)

Exercise 3.1: Buffer Replacement with LRU-k

Please assume the following access pattern to pages A,B,C,D,E,F,G,H,I, and J:
AACDEFGHABGHCCCAGHIJABABCCCAAG

Moreover, the assume that a database buffer provides 6 frames.

Apply the LRU-k replacement algorithm for k=1,2,3 and discuss the results with respect to the total number of replacements. Which value for k is considered to be the optimal solution?

Solution:
k=1

t=8
Buffer: A, C, D, E, F, G
b(A,1)= 6 <---- selected for replacement
b(C,1)=5
b(D,1)=4
b(E,1)=3
b(F,1)=2
b(G,1)=1

t=9
Buffer: H, C, D, E, F, G
b(H,1) = 1
b(C,1) = 6 <---- selected for replacement
b(D,1)= 5
b(E,1)=4
b(F,1)=3
b(G,1)=2

t=10
Buffer: H, A, D, E, F, G
b(H,1)=2
b(A,1)=1
b(D,1)=6 <---- selected for replacement
b(E,1)=5
b(F,1)=4
b(G,1)=3

t=13
Buffer: H, A, B, E, F, G
b(H,1)=1
b(A,1)=4
b(B,1)=3
b(E,1)=8 <--- selected for replacment
b(F,1)=7
b(G,1)=2

t=19
Buffer: H, A, B, C, F, G
b(H,1)=1
b(A,1)=3
b(B,1)=9
b(C,1)=4
b(F,1)=13 <--- selected for replacement
b(G,1)=2

t=20
Buffer: H,A, B, C, I, G
b(H,1)=2
b(A,1)=4
b(B,1)=10 <--- selected for replacement
b(C,1)=5
b(I,1)=1
b(G,1)=3

t=22
Buffer: H,A, J, C, I, G
b(H,1)=4
b(A,1)=1
b(J,1)=2
b(C,1)=7 <--- selected for replacment

b(I,1)=3
b(G,1)=5

t=25
Buffer: H, A, J, B, I, G
b(H,1)=7
b(A,1)=2
b(J,1)=5
b(B,1)=1
b(I,1)=6
b(G,1)=8 <--- selected for replacement

t=30
Buffer: H,A, J, B, I, C
b(H,1)=12 <--- selected for replacement
b(A,1)=1
b(J,1)=10
b(B,1)=6
b(I,1)=11
b(C,1)=3

k=2

t=8
Buffer: A, C, D, E, F, G
b(A,2)= 7
b(C,2)= Infinity <--- selected for replacement
b(D,2)= Infinity
b(E,2)=Infinity
b(F,2)=Infinity
b(G,2)=Infinity

t=10
Buffer: A, H, D, E, F, G
b(A,2)=8
b(H,2)= Infinity
b(D,2)=Infinity <--- selected for replacement
b(E,2)=Infinity
b(F,2)=Infinity
b(G,2) = Infinity

t=13
Buffer: A, H, B, E, F, G
b(A,2)=11
b(H,2)=5
b(B,2)=Infinity
b(E,2)=Infinity <--- selected for replacement
b(F,2)=Infinity
b(G,2)=6

t=19
Buffer: A, H, B, C, F, G
b(A,2)=10
b(H,2)=7
b(B,2)=Infinity
b(C,2)=5
b(F,2)=Infinity <--- selected for replacement
b(G,2)=8

t=20
Buffer: A, H, B, C, I, G
b(A,2)=11
b(H,2)=8
b(B,2)=Infinity <--- selected for replacement
b(C,2)=6
b(I,2)=Infinity
b(G,2)=9

t=22
Buffer: A, H, J, C, I, G
b(A,2)=13
b(H,2)=10
b(J,2)= Infinity
b(C,2)=8
b(I,2)=Infinity <--- selected for replacement
b(G,2)=11

k=3

t=8
Buffer: A, C, D, E, F, G
b(A,3)= Infinity
b(C,3)= Infinity <--- selected for replacement
b(D,3)= Infinity
b(E,3)=Infinity
b(F,3)=Infinity
b(G,3)=Infinity

t=10
Buffer: A, H, D, E, F, G
b(A,3)= 9
b(H,3)= Infinity
b(D,3)= Infinity <--- selected for replacement
b(E,3)=Infinity
b(F,3)=Infinity
b(G,3)=Infinity

t=13
Buffer: A, H, B, E, F, G
d(A,3)=12
d(H, 3)=Infinity
b(B,3)= Infinity
b(E,3)=Infinity <--- selected for replacement
b(F,3)=Infinity
b(G,3)=Infinity

t=19
Buffer: A, H, B, C, F, G
b(A,3)=17
b(H,3)=11
b(B,3)=Infinity
b(C,3)=6
b(F,3)=Infinity <--- selected for replacement
b(G,3)=12

t=20
Buffer: A, H, B, C, I, G
d(A,3)=20
d(H,3)=12
d(B,3)=Infinity <-- selected for replacement
d(C,3)=7
d(I, 3)=Infinity
d(G,3)=13

t=22
Buffer: A, H, J, C, I, G
b(A,3)=22
b(H,3)=14
b(J,3)=Infinity
b(C,3)=9
b(I,3)=Infinity <--- selected for replacement
b(G,3)=15

Discussion

For k=1, 9 replacement operations must be performed, whereas for k=2 and k=3, only 6 replacements are necessary. Obviously, LRU-1 is equal to LRU. For k>1, recent page references indicate further accesses in the future, whereas older page references that are used rarely are selected as victims first.

According to O'Neil et al., in general, k=2 provides the best solution, because (1) the results for k>2 are not significantly better than for k=2, (2) the identification of victims is much simpler, and (3) reference variations are anticipated much better than for larger k.

Exercise 3.2: Locality and Sequentiality

Solution:

A transaction creates the following reference string:
AABBCEDABGHAAGGHHIIKKLKLKLKLMABABKLKLKLFMFAGHI
Please take into consideration that the following sequential order is assumed:
ABCDEFGHIKLMNO

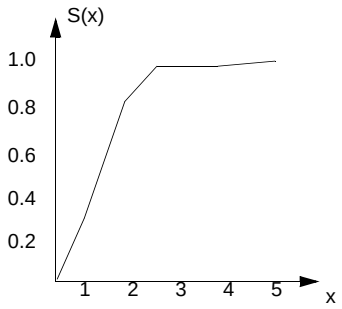
Please calculate the following numbers:

- The current locality at time t = 6, 18, 28 for the widow size w=6.
  AL(t,6) = W(t, 6) / 6
  AL(6,6) = 4/6
  AL(18,6) = 4/6
  AL(28,6) = 2/6

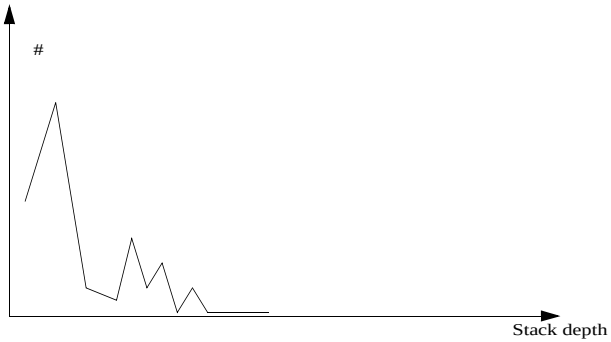
- The average locality
  L(6) = ((4+5+5+5+6+6+5+...)/41) / 6 = (158 / 41) / 6 = 0,642

- The length of sequential reference sequences (SRSs) and the cumulative distribution of SRS lengths
-

SRS	Length
AABBC	3
E	1
D	1
AB	2
GH	2
AA	1
GGHHIIKKL	5
KL	2
KL	2
KLM	3
AB	2
AB	2
KL	2
KL	2
KL	2
F	1
M	1
F	1
A	1
GHI	3



- The LRU-stack-depth distribution.



Discuss the dynamic allocaction of buffer pages for this transaction. How can such dynamic buffer allocations be efficiently determined in DBMSs?

Exercise 3.3: Search in the DB Buffer

In the lecture notes, several approaches for supporting fast search of DB pages in a DB buffer are described. Please analyze the following methods regarding their maintenance and search costs:

a) Unsorted table

b) Sorted table

c) Table with chained entries (LRU order)

d) AVL-tree

e) Hash table with overflow chain

Assumptions

We assume that in one out of ten times, a page fault occurs and, as a consequence, a replacement must be performed.

Let method i have m_i cost units of maintenance costs w and for each search, s_i cost units of search costs c.

Let there be $n = 2^{10}$ entries per table or data structure.

To keep the table sorted, we assume that n/2 entries must be moved. This raises costs of n/2 cost units of maintenance costs w.

For LRU, we assume that finding a page, which is already in the DB buffer, raises n/10 cost units of c. In the case of a hash table, in average, 1.5 comparisons are assumed.

Please calculate the costs C_i which are caused by the ten search operations and the replacement operation.

Please appropriately estimate the various cost factors!

Which method prevails, if we assume w = 5 * c?

Do the fundamental statements still hold, if we do not assume an average locality of 90 % (10/1), but instead assume x% ((x+y)/y)? Discuss the case where x=y, i.e., x=y=5, to allow for an immediate comparison.

Solution:

a) Unsorted table

$C_1 = 9 * c * n/2 + 1 * c * n + w * 1 = (11 * 512 + 5) * c = 5637 * c$

b) Sorted table

$C_2 = 10 * c * \log_2(n) + 1 * w * n/2 = 10 * c * \log_2(2^{10}) + w * 2^{10}/2$
$= (100 + 2560)*c = 2660*c$

c) Table with chained entries (LRU order)

$C_3 = 9 * c * n/10 + 1 * c * n + 1 * w * 2 = 1440 * c$

d) AVL-tree

$C_4 = 10 * c * 1{,}44 \log_2(n) + 1 * w * 1{,}44 \log_2(n) = 10 * c * 1{,}44 * 10 + w * 1{,}44 * 10$
$= 216*c$

e) Hash table with overflow chain

$C_5 = 10 * c * 1{,}5 + 1 * w * 2 = 25 * c$

There is definitely a winner of this competition: The appropriateness of methods is given by the following sequence: (C_5, C_4, C_3, C_2, C_1).

Please note, this sequence is not only a result of the chosen parameters. Furthermore, its outcome is very common and holds in real-world scenarios.

Do the fundamental statements still hold, if we do not assume an average locality of 90 % (10/1), but instead assume x% ((x+y)/y)? Discuss the case where x=y, i.e., x=y=5, to allow for an immediate comparison.

a) Unsorted table

$C_1 = x * c * n/2 + y * c * n + w * y = (15 * 512 + 25) * c = 7705 * c$

b) Sorted table

$C_2 = (x+y) * c * \log_2(n) + y * w * n/2 = 10 * c * \log_2(2^{10}) + 5 * w * 2^{10}/2$
$= (100 + 12800) * c = 12900 * c$

c) Table with chained entries (LRU order)

$C_3 = x * c * n/10 + y * c * n + y * w * 2 = 5682 * c$

d) AVL-tree

$C_4 = (x+y) * c * 1{,}44 \log_2(n) + y * w * 1{,}44 \log_2(n)$
$= 10 * c * 1{,}44 * 10 + 5 * w * 1{,}44 * 10 = 504 *c$

e) Hash table with overflow chain

$C_5 = (x+y) * c * 1{,}5 + y * w * 2 = 65 * c$

In this case, the sequence of appropriate methods is given by (C_5, C_4, C_3, C_1, C_2), but the cost unit w for movements in the table is probably much lower than assumed!

Exercise 3.4: Hot Set Model

Solution:

Let there be three relations R1, R2, and R3 consuming 10, 20, resp. 30 pages. Every page encompasses 10 tuples. For the calculation of both 1:n joins ((R1 |X| R2) |X| R3), a n*ested-loops join* operator is used.

The intermediate results are stored in temporary tables with 20 resp. 30 pages.

```
foreach i in R2
    find j in R1
    calculate i x j
    put result in R'
end
foreach i in R3
    ... (as above with R' instead of R1, result in R'')
end
```

How many page faults occur over time when attention is paid to the number of available buffer frames and LRU is used a page replacement strategy?

There are 3-11 pages available:

Every page access causes a page fault	
Reading of R2 = 20 pages with 10 tuples	
plus, for each tuple of R2 (200), 10 page accesses for R1	
Storing every internemdiate result (200) in a page of R' = 20	20 + 200 * 10 + 20 = 2040
Reading of R3 = 30 pages with 10 tuples	
plus, for each tuple of R3 (300), 20 pages accesses for R'	
Storing every intermediate result (300) in a page of R'' = 30	30 + 300 * 20 + 30 = 6060
	8100 page replacements

There are 12-21 pages available:

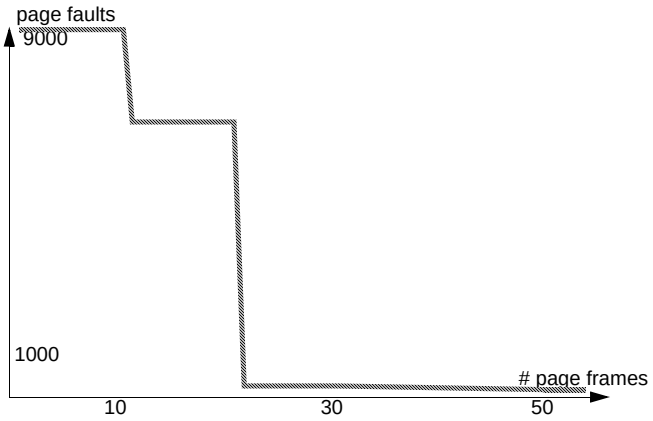
The 10 pages of R1 remain in the DB buffer during the first loop	
10 page faults for R1	
20 page faults for R2	
20 page faults for R'	50
Reading of R3 = 30 pages with 10 tuples	
plus, during each iteration (300), reading 20 pages of R'	
Storing each intermediate result in pages for R'' = 30	30 + 300 * 20 +30 = 6060
	6110 page replacements

There are 22-51 pages available:

First loop: as described above	50
20 pages of R' remain in the buffer, but must be read first	20 + 30 + 30
	130 page replacements

There are more than 51 pages available:

First loop: as above	50
Second loop: as above, but R' is already in the buffer	30 + 30
	110 page replacements



Exercise 3.5: Analysis of Paging Behaviour

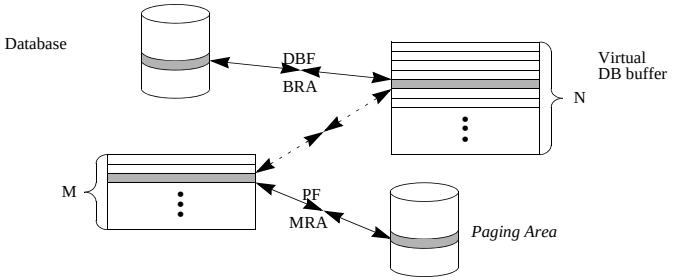
Solution:

Main-memory database buffers serve for data processing in DBMSs. If an access to a database page is requested, the buffer manager checks whether this page is already present in a DB buffer. If this is true, the requesting component can immediately access the page. Otherwise, the buffer manager must load the requested database page from the external storage and has to put it into the database buffer. Therefore, it might become necessary to replace another page in the DB buffer, if there are no free buffer frames left. In operating systems with virtual memory, DB buffers reside in a virtual address space, whose pages are also replacable. The buffer manager of the DBMS controls the DB buffer independently of the operating system. Since the operating system uses its own replacement strategies for the complete virtual address space, which also includes the DB buffer, the problem of *Double Paging* can occur.

Let a database consist of D pages and let us assume that the DB buffer can hold N pages. Let us furthermore imply that the virtual DB buffer pages are assigned to M real pages and M < N < D holds.

If an access to the database becomes necessary, we call this situation a *Database Fault* (DBF). If the requested page resides in virtual memory, but an access to a paging area is required, we refer to this situation as *Page Fault* (PF). The DB buffer is managed by a *buffer replacement* algorithm (BRA) and the main-memory pages are managed by a *memory replacement* algorithm (MRA).

- Provided that there is a random assignment of database pages to virtual buffer pages and from virtual buffer pages to main-memory pages, please calculate the probability of a DBF and a PF.
- Develop a simple model for calculating the IO costs per database access as a function f(D, N, M). Please introduce a constant factor describing the ratio between PF costs and DBF costs. ..



We will use the following abbreviations:

DBF: *buffer fault* (*database fault*)
PF: *memory fault* (*page fault*)
BRA: *buffer replacement algorithm*
MRA: *memory replacement algorithm*

The following constants are used:

D: # of pages allocated by the database
N: capacity of the database buffer (DBP)
M: # of assigned main-memory pages

Probability d that a database page resides in the virtual DB buffer (if uniform distribution is assumed):

d = 1 (for D < N)
d = N/D (for D >= N)

Probability n that, for a randomly chosen replacement algorithm, a virtual buffer page resides in the main-memory:

n = 1 (for N < M)
n = M/N (for N >= M)

Precondition: We assume random assignments of database pages to the DBB and of virtual buffer pages to the main memory.

Page faults can occur, if:

(1) a database request finds the required page in the DBB, which is not in the main memory

(2) a database request is mapped to page in the DBB, which is not in the main memory

Probability of a *page fault* (*memory fault*): PF = 1 - n

Probability of a *database fault*: DBF = 1 - d

Expected I/O costs (T) per database request:

$T(N,M,D) = (1 - n) * C_{PF} + (1 - d) * C_{DBF} = (1 - M/N) * C_{PF} + (1 - N/D) * C_{DBF}$
for $1<=M<=N<=D$, where C_{PF} and C_{DBF} are the cost units for *page faults* resp. *database faults*.

The I/O cots may vary per fault, because modified pages need to be flushed to disk. We assume for all fault types the following average costs: $C_{PF} = x * C_{DBF}$ for $0 < x < oo$

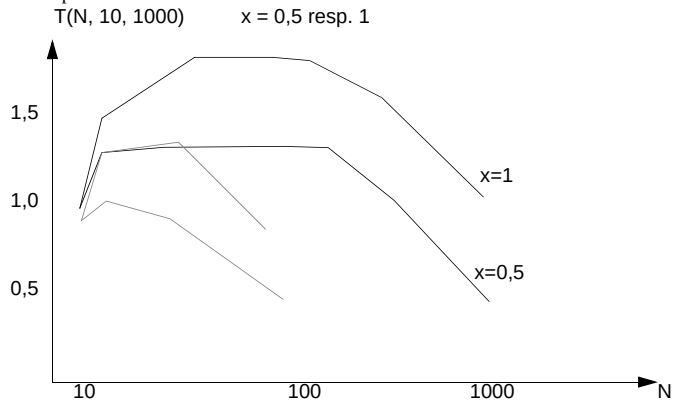
Then:

$T(N, M, D) = (x - x * M/N + 1 - N/D) * C_{DBF}$
with constant values for M and D, we get the following formula for the I/O costs:

$\Delta T(N) = T(N + 1) - T(N) = (x * M/(N^2 + N) - 1/D) * C_{DBF}$

If D (= |< | >) $(N^2 + N) / x * M$, then $\Delta T(N)$ is a (constant | increasing | decreasing) function of the virtual buffer size.

An Example



=> In real-world scenarios, choosing a DB buffer capacity larger than the actual number of available pages is not effective.

The dashed line represents the values for D=100. If there is a high degree of locality, this trend is realistic.