CSE 5331- DBMS Models and Implementation

Project 1: Buffer replacement policy implementation and BHR calculations Report

Group no.: 5

Group members:

1. Pratik Antoni Patekar (1001937948)
2. Tharun Reddy Nalabolu (1002069440)

Table of contents

1.0. Overall status:

In this project, we have completed implementing the two buffer replacement policies assigned i.e.

1. Least Recently Used (LRU)
2. Most Recently Used (MRU)

Each of these policies have been implemented for two different buffer sizes i.e. 15 and 45 each. Other than this, we have even implemented code in Buffer manager to calculate and print the Buffer Hit Ratio (BHR) and some other buffer page information such as total number of page hits, page loads and page faults.

2.0. Data structures, implementations and file descriptions:

2.1. Data structures and functions used for policy implementations: For implementation of policies, we were provided with a sample policy java file (Policy.java). In that file, we have modified two functions, namely,

1. pickVictim() - a function to pick a victim frame whose page should be replaced and return the frame number back to buffer manager class.
2. update() - a function to update the “frames” array (discussed in policy implementation explanation) as per the buffer replacement policy requirements. This function returns nothing.

Along with this we also used a few data structures which provided us with information about the buffer and buffer pages. These data structures are as follows:

1. framestab - This is an array whose size is the same as the buffer and each of its elements contain information corresponding to each frame in the buffer. Each element in the array contains buffer frame information such as index of the frame, page no, pin count, boolean bit indicating whether the corresponding frame is dirty or not. Along with these four parameters it also contains an integer value indicating the state i.e. whether the frame is available, referenced or pinned.
2. frames - This is a dummy array whose size is the same as the buffer. In LRU and MRU policy implementations, we have used this array to store the indexes of the framestab array as we had to maintain the order in which the policy should find the victim page. In case of LRU, the framestab must be treated as a queue whereas in MRU, it should be treated as a stack. (further discussed in policy implementation explanation).
3. nframes - This is an integer value that stores the index at which the page is stored the last time. This variable is used only till the time when all the pages are pinned in the buffer for the first time. (further discussed in policy implementation explanation).

2.2. Data structures and functions used for BHR calculation and printing: For implementation of BHR calculations we were provided with the Buffer manager java file (BufMgr.java). In this java class file, we have modified the following functions,

1. newPage() - This function is basically called when a new page is being allocated.
2. pinPage() - This function is basically called when a page needs to be pinned in the main memory i.e. buffer. We have used this function to record page hits, page loads and page faults.
3. printBhrAndRefCount() - This function is used to calculate and print the BHR and other values that are recorded in the pinPage function. This function also calls sort_func() function to print the top k referenced pages.
4. sort_func() - This is a function that we have added in order to sort and print the top k referenced pages in descending order of their hits.

Following are the variables and the data structures that we have used in the BHR calculation implementation.

1. totPageHits - Stores the total number of page hits.
2. totPageRequests - Stores the total number of page requests made.
3. aggregateBHR - Stores the BHR ratio calculated.
4. page_ref - This is an ArrayList that contains the hit count and load count of every page (pageno.pid).

2.3. Implementation:

2.3.1. pickVictim(): LRU buffer replacement policy chooses the least recently used page as the victim page and on other hand, MRU buffer replacement policy chooses the most recently used. As mentioned earlier, for doing so we have used information from framestab array and for treating the framestab array as queue (LRU) or stack (MRU), we have used frames array to restore the indexes of the framestab array and re-order the indexes as per policy.

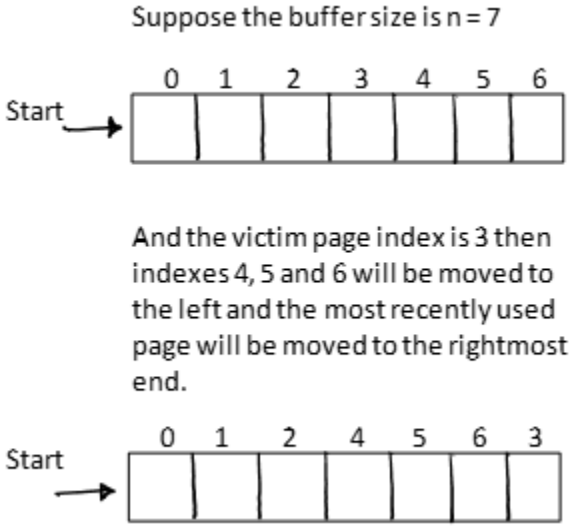
Let's assume that the buffer size is n. Following is the stepwise procedure that we have followed for implementing the pickVictim() function.

1. Initialize the nframes variable to 0. (Done in the constructor)
2. Now if the nframes values is less than the $(n - 1)$ then choose the frame with index = nframes, update the index in frames array, pin the selected frame and increment the nframes variable by 1 (so that it points at the next available frame in buffer).
   Now this step 2 will be repeated till the nframes variable value is less than buffer size. But once the buffer is full with pages then we need to find the victim page.
3. The frames array contains indexes which are arranged every time the update function is called as per the policy (as a queue for LRU and as a stack for MRU). Due to this reason, we iterate through the framestab array to find the victim page in the order of indexes stored in the frames array.
4. While iterating through the framestab array as per frames array, we check if that frame is pinned or not. If not pinned then update the frames array as per the policy, change the frame state to pinned and return the frame number to buffer manager class.

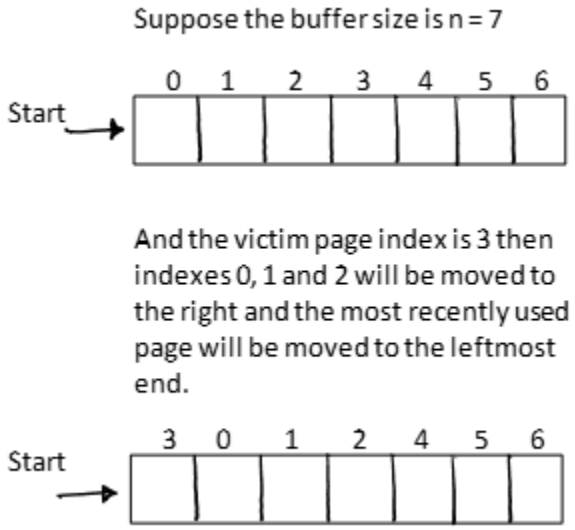
The implementation of the pickVictim() function is the same for both LRU and MRU policies.

2.3.2. update(): The main difference in LRU and MRU policy implementation is in update function i.e. how the update function modifies the frames array i.e. the order in which the frames are iterated to find the victim page.

In LRU, once a victim page is found, the update function is called. This index of the victim page now thus needs to be moved to the end of the array as this victim page is now the most recently used page. As in pickVictim() function the iteration is done from left to right direction (from index 0 to $n - 1$) we move the most recently used page to the end of the queue and as a result the least recently used pages remain on the left side. This is done using simple for loop logic and a temp variable. The same is illustrated in the figure below.



In MRU, similar logic is used. The only difference is that the victim page index is moved to the start of the frames array so that the most recently used frame is available at the start of replacement (if not pinned).



2.3.3. Changing the buffer size as per assigned values: To change the buffer size used, we had to make modifications in the Global constants java class file and BufMgr java class file. In global constants file we changed the value of NUMBUFS variable to the required buffer size value and in the buffer manager get that value in constructor (so that it is used for buffer creation).

2.3.4. BHR calculation and printing: For calculating the BHR, we need to record the number of page hits and page loads of every page. For doing so, we have used an ArrayList named page_ref. This array list consists of 3 array lists, each contain following information,

1. Pageno.pid - This is the page id of a page
2. Hit count - Number of times the corresponding page has been hit.
3. Load count - Number of times the corresponding page has been loaded.

In pinPage() function, there are two parts i.e. if the page already exists in the buffer and if it does not exist. So, in the first part we have added a code to increment the page hit count in the page_ref arraylist. And in the second part, we have added the code to increment the load count of a page in the page_ref arraylist.

Also while doing so, we have checked if the victim page is dirty or not. If it is dirty then we have set the reference count of that page to zero.

In the printBhrAndRefCount() function, we have calculated the sum of the page hits and the page loads in variables totPageHits and totPageRequests respectively. These variables are then used to calculate the BHR as ratio of totPageHits variable and totPageRequests variable.

Then we call sort_func() function to print the page_ref arraylist in descending order of the page hit counts. In the sort_func() function we have implemented the sort using basic sorting logic using 2 for loops.

3.0. File descriptions:

The zip folder submitted consists of 5 folders (other than the report) namely,

1. LRU_15
2. LRU_45
3. MRU_15
4. MRU_45
5. Outputs

The file naming terminology used is <policy_name>_<buffer_size> for the first 4 folders.

The first 4 folders consist of 2 folders i.e. bin and src along with a Makefile. The src folder consists of 2 folders i.e. bufmgr and tests. The bufmgr folder consists of java files with our implementations namely,

1. **BufMgr.java:** Modified for implementation of BHR calculation as mentioned in the previous section.
2. FrameDesc.java
3. **GlobalConst.java:** Modified for changing the buffer sizes
4. **<policy_name>.java:** Modified for implementation of policies using pickVictim() and update() functions as mentioned in the previous section.
5. Replacer.java

The ones that are highlighted above are the ones that we have modified as compared to what we were provided with.

The tests folder consists of 2 test files provided i.e. BMTest.java and BHRTest.java. These files have not been modified but these are the files on which the implementations were tested and ran successfully.

The Outputs folder consists of 4 text files consisting of command line outputs for each policy and buffer size combination. File name terminology used is <policy_name>_<buffer_size>.

4.0. Result Analysis:
The next 4 pages show results of BHR for different policies with different buffer sizes and top 5 pages and references.

4.1. LRU test results for buffer size 15 :

Policy/Test# LRU (buffer size = 15)	BHR	Top 5/10 pages and references		
		Page no.	No. of Loads	No. of Page Hits
Test 1 (just pin/unpin)	0	9	2	0
		16	1	0
		17	1	0
		18	1	0
		19	1	0
Test 2 (favors MRU)	2.6	31	1	8
		50	1	7
		41	1	6
		14	1	6
		54	1	6
Test 3 (multiple pins/unpins)	2.82	18	1	12
		29	1	9
		31	1	8
		53	1	8
		65	1	8
Test 4 (Random)	51	15	1	52
		10	1	52
		19	1	52
		9	1	51
		12	1	50

4.2. LRU test results for buffer size 45:

Policy/Test# LRU (buffer size = 45)	BHR	Top 5/10 pages and references		
		Page no.	No. of Loads	No. of Page Hits
Test 1 (just pin/unpin)	0	9	2	0
		31	1	0
		32	1	0
		33	1	0
		34	1	0
Test 2 (favors MRU)	2.68	13	1	9
		96	1	8
		54	1	7
		47	1	7
		138	1	7
Test 3 (multiple pins/unpins)	2.76	168	1	12
		215	1	10
		225	1	10
		13	1	9
		96	1	8
Test 4 (Random)	16.57	9	1	27
		38	1	23
		30	1	21
		11	1	21
		24	1	20

4.3. MRU test results for buffer size 15:

Policy/Test# MRU (buffer size = 15)	BHR	Top 5/10 pages and references		
		Page no.	No. of Loads	No. of Page Hits
Test 1 (just pin/unpin)	0	9	2	0
		16	1	0
		17	1	0
		18	1	0
		19	1	0
Test 2 (favors MRU)	2.94	12	1	12
		11	1	9
		10	1	9
		31	1	8
		9	2	7
Test 3 (multiple pins/unpins)	2.92	11	1	12
		12	1	12
		10	1	9
		31	1	8
		9	2	7
Test 4 (Random)	51	15	1	52
		10	1	52
		19	1	52
		9	1	51
		12	1	50

4.4. MRU test results for buffer size 45:

Policy/Test# MRU (buffer size = 45)	BHR	Top 5/10 pages and references		
		Page no.	No. of Loads	No. of Page Hits
Test 1 (just pin/unpin)	0	9	2	0
		31	1	0
		32	1	0
		33	1	0
		34	1	0
Test 2 (favors MRU)	3.76	13	1	16
		30	1	16
		35	1	15
		36	1	14
		37	1	14
Test 3 (multiple pins/unpins)	3.82	41	1	19
		30	1	18
		22	1	17
		13	1	16
		29	1	16
Test 4 (Random)	16.57	9	1	27
		38	1	23
		30	1	21
		11	1	21
		24	1	20

4.5. BHR values side-by-side comparison:

Policy/Test #	LRU (buffer size = 15)	LRU (buffer size = 45)	MRU (buffer size = 15)	MRU (buffer size = 45)
Test 1 (just pin/unpin)	0	0	0	0
Test 2 (favors MRU)	2.6	2.68	2.94	3.76
Test 3 (multiple pins/unpins)	2.82	2.76	2.92	3.82
Test 4 (Random)	51	16.57	51	16.57

4.5.1. Analysis on basis of policies wrt hit ratios:

In test 1, we are just pinning a page and unpinning it after that. After pinning the page, the same page is not called again due to which the number of hits corresponding to each page is zero. And due to this reason, the BHR value will also be zero irrespective of the policy and buffer size. The same results can be seen above in the test 1 BHR value comparison as well.

For the same buffer size of 15 (or 45) in other tests we can see that the MRU BHR ratio is better than that for LRU. The MRU performs better when there is more chance that the page will be accessed, the older it is. That is what is happening in test 2 and test 3.

In test 4, the results of LRU and MRU both are the same as the pages are accessed in random order. In this case, both the policies performance parameter or BHR is the same.

4.5.2. Analysis on basis of different buffer sizes on different policies:

Ideally, when the buffer size is increased, the number of page hits should increase as there is more space for storing pages on the buffer. As there is more space to store the pages, the probability that the page requested will be found in the buffer itself will increase. We can see this similar result in case of test 2. As the buffer size increases from 15 to 45, the BHR increases slightly (for LRU and MRU both). But this is not always the case.

In the test 3 results for LRU, we can see that the BHR decreases slightly as buffer size increases. There are two possible reasons for this to happen. In LRU, as the least recently used page in the buffer is chosen as victim due to which it is possible that the older or frequently used pages in the buffer are replaced and thus contributing towards decreasing the hit ratio. The second possible reason is that, in the test files, the number of disk pages requested is proportional to (i.e. 5 times) buffer size. So when we increase the buffer size from 15 to 45 we increase disk pages from 75 to 225. Thus, when we increase the buffer size by 30, the number of disk pages increases by 150 and thus contributes towards decreasing the hit ratio again.

The test 3 results for MRU shows that the BHR increases as the buffer size increases similar to test 2 case.

In test 4, the pages are chosen at random (i.e. predictably random). As the pages are called at random and there are no frequently used pages. Thus, even if LRU and MRU policies try to store the pages that are most recently or most frequently used the BHR decreases as the buffer size increases.

5.0. Division of Labor: The division of coding jobs was straightforward. We had to implement two buffer replacement policies, so one policy for each member. Further more detailed information on division of work is as follows:

Though the assignment was one policy per group member, we spent the first 2 weeks on understanding the LRU implementation together as the MRU implementation would just have been a small modification in the LRU implementation.
After the LRU implementation was completed, the MRU implementation was completed in 2 days itself. On completion of policy implementations, we worked together for implementing the BHR printing function together till the last day of the project submission.

Group member name	**Policy implementation**	**Hours spent on project on weekdays**	**Hours spent on project on weekends**
Pratik Antoni Patekar (1001937948)	LRU (both buffer sizes), BHR calculation and report	Utmost 2 hours	4 to 5 hours
Tharun Reddy Nalabolu(1002069440)	MRU (both buffer sizes), BHR calculation and report	Utmost 1 hour 30 minutes	3 hours

6.0. Logical errors and how we handled/ resolved them: Following are a few logical errors that we faced during implementing the project.

1. Understanding the use of update and pickVictim functions and how they are called from the buffer manager class file. To overcome this issue, we had to study the codes given to us and contact TA in-person. Also, going through the helper slides provided by the TA helped.
2. Understanding the use of frames array. This logical error was the most time consuming one. As we were referring to the helper slides provided by the TA, it was mentioned in those slides that, for implementing the LRU policy we need to consider the frametab array as a queue. Though the understanding of the concept was clear, we were modifying the frametab array as a queue instead of using frames array to store indexes of frametab array and modify the frames array accordingly. This issue was resolved after the professor explained the use of the frames array in the first few minutes of a lecture.

3. Understanding the variables that were getting printed in the BHR print function. This issue was resolved after discussion with the professor during the lectures and it was further more clear when the sample output files were provided.