Database System Implementation (CSE507) : Homework 4

Anshuman Suri : 2014021
Rounaq Jhunjhunu Wala : 2014089

Buffer Size	Query							
	Q1		Q2		Q3		Q4	
	LRU	MRU	LRU	MRU	LRU	MRU	LRU	MRU
10	101000	2082964	100100	1920064	50500	1032464	50100	960064
20	101000	2062829	100100	1719929	50500	1012329	50100	859929
50	101000	2001824	100100	1118924	50500	951324	50100	558924
75	101000	1950299	100100	617399	50500	899799	50100	307399
100	101000	1898149	100100	115249	50500	847649	50100	55249
200	1100	1688150	100100	85349	600	637650	50100	35349
500	1100	1058150	100100	55649	600	7650	50100	5649
1000	1100	8150	100100	6149	600	600	600	600
1500	1100	1100	1100	1100	600	600	600	600
2000	1100	1100	1100	1100	600	600	600	600
2500	1100	1100	1100	1100	600	600	600	600

The number of page faults for each of the specified cases are present in the spreadsheet file in the submission. An image is also shown above. We will discuss the trends that we got and the possible explanation for those trends.

Table 1: Table Statistics

Table Name	# records	#disk blocks
Employee	10000	1000
Department	1000	100
Project	5000	500

Q1 : Block nested loop join between Department and Employee with Department as the inner loop
Q2 : Block nested loop join between Department and Employee with Employee as the inner loop.
Q3 : Block nested loop join between Department and Project with Department as the inner loop.
Q4 : Block nested loop join between Department and Project with Project as the inner loop.

1. **The number of page faults always saturate down to the number of unique blocks accessed during the query :** We can see that this happens *at most* when the buffer size increases to an amount which is greater than or equal to the number of unique blocks accessed. All the page faults in this case are due to the **compulsory misses** of loading a block for the first time during the procedure, and hence can't be removed at all. We can also see that on increasing the cache size further, we are just wasting the

resources because the number of blocks accessed by a particular query is fixed.

2. **The number of page faults in the LRU are same for all cases when it is not at the saturation page faults :** For each case in LRU mode, we see that the number of page faults = (<<blocks in inner loop>> + 1) * <<blocks in outer loop>> for all the queries. The number is same to the number of block accesses (ignoring tuple join loops) we have for nested-block join. The reason for a page fault on every access, is because of the sequential access of inner relation, there will be an page fault on every index since LRU would have removed an inner block. The the additional #outer-blocks are for loading of new blocks for each iteration. The saturation point is reached whenever the whole inner relation fits into the RAM, and since we are just doing a sequential scan on outer loop, no working page is evicted and the number of page faults is essentially the same as the full saturation case.

3. **The Page faults in MRU are very high for small cache size :** Since, during tuple join, we access the same 2 blocks repeatedly. But, since MRU evicts the most recent page, there are continuous page faults for every tuple join. This results in very high number of page faults. This number decreases on increasing cache size as the capacity of the cache to hold more blocks would lead to lesser page faults.

4. **Comparing MRU with LRU :** On overall comparison, we can see that LRU performs better than MRU. The main reason for that being the **Clustered Sequential** scan order of the blocks. Since a range of the block space is looped very frequently, it suffers from multiple block-evictions due to the ever-increasing timestamp.