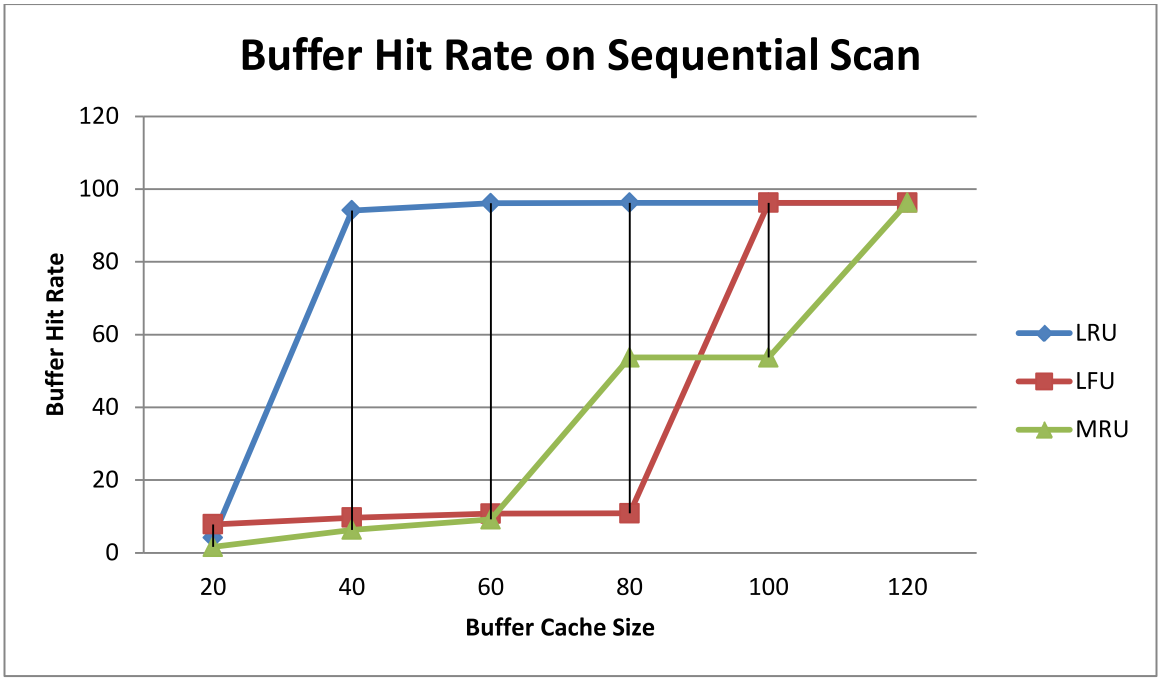
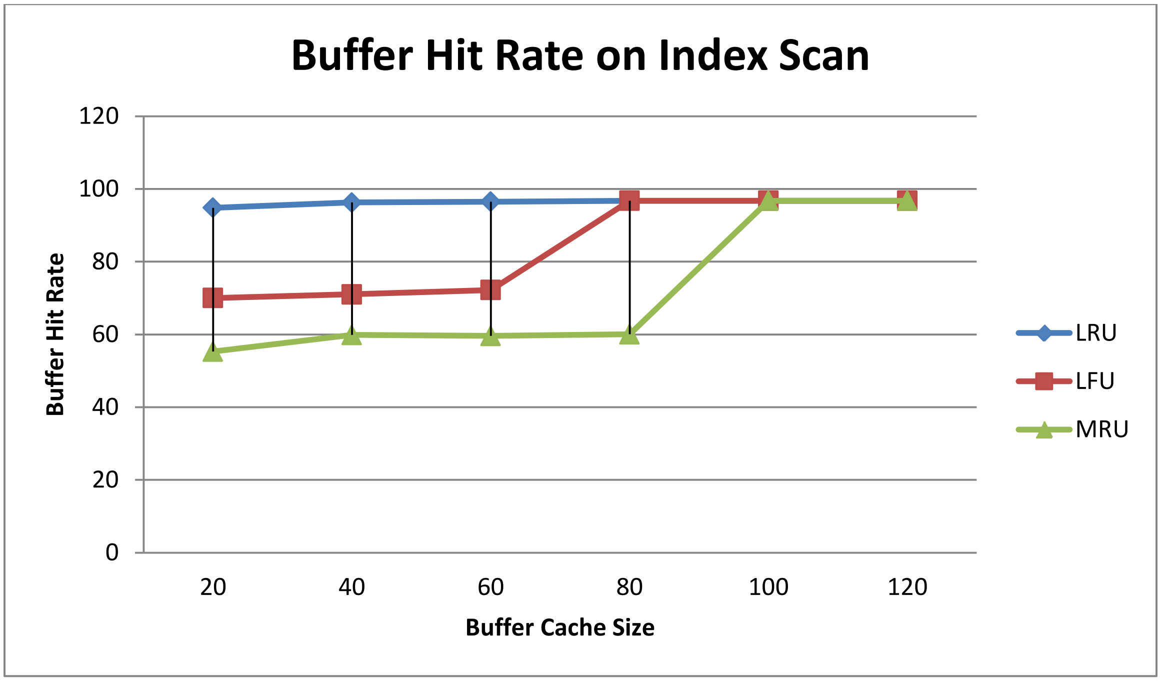
2011

University of Toronto

Kobe Sun 996003756
Shane Jiang 996022281

[CSCD43 ASSIGNMENT 1]





Note: The buffer cache size is being kept as constant and its values are 20, 40, 60, 80, 100, 120.

1. Analyze/Interpret any “interesting” trends that you notice in the behaviour of the LRU, LFU, and MRU replacement policies. You have to explain when and why one policy is better than the other.

   Answer:
   During the sequential scan, we monitor the sequential flood when the buffer cache size is around 30, because there is a jump for the buffer hit rate from 0% (buffer cache size: 20) to 100% (buffer cache size: 40). Therefore, buffer management becomes useful after the buffer cache size reaches 30. We have learned from the lecture that MRU should perform better than LRU when sequential flooding happens. However, what we have seen here is quite different. After a deep investigation, I find that PostgreSQL occupies some buffers to initialize the system. By using MRU, there will be no way to get rid of these garbage buffers after they are unpinned. Therefore, in general, the performance of MRU is worse than LRU in our test, although the performance of MRU becomes better along with the increases of buffer cache size. LFU works just like MRU at the beginning, and is also affected by the system garbage buffers. However, for some of the system garbage buffers that are rarely used, they do have a chance to be evicted from the buffer pool. For those buffers with the same frequencies, they are arranged as FILO. Therefore, the performance of LFU is roughly the same as MRU and worse than LRU.

   During the index scan, the blocks are accessed randomly. Since the index block is small and can be easily placed in the buffer pool, the buffer hit rate for LRU, MRU and LFU are generally higher than those during sequential scan and there is no sequential flood. But because of the same reason, the performance of MRU is affected by those system garbage buffers again. LFU performs better than MRU, because index blocks are frequently accessed. However, being affected by system garbage buffers, LFU performance is worse than LRU.

   The reason that we can’t get 100% buffer hit rate for any of the replacement policy with any size of buffer cache size is that PostgreSQL needs to load some system blocks (i.e catalog) from the disk when it starts, so the buffer hit rate for the 1st query execution will never be 100%, which affects the average result.

2. Try to estimate the size of your table Data in number of blocks, based on the buffer cache hit rates, with increasing size of the table.

   Method 1:
   From the Sequential Scan graph, we can easily find the sequential flood. It happens when the buffer size < page size of the table. If we can find the marginal point, we can know the size of the table data. After several trials under LRU replacement policy, we get the following results:

   ```
           cat ScanQueries.sql | postgres -B 29 -D ~/data/ -d 1 -s mydatabase
       !     Shared blocks:       30 read,        0 written, buffer hit rate = 0.00%
           cat ScanQueries.sql | postgres -B 30 -D ~/data/ -d 1 -s mydatabase
       !     Shared blocks:       0 read,        0 written, buffer hit rate = 100.00%
   ```

   Therefore, the size of the table is roughly around 30.

   Method 2:
   The size of the table Data is roughly 30 blocks.
   I call the following command to do the analysis.

   ```
           cat ScanQueries.sql | postgres -B 60 -D ~/data/ -d 1 -s mydatabase
   ```

   The statistics I got under LFU replacement policy are as follow:

   ```
       !     Shared blocks:      112 read,        0 written, buffer hit rate = 61.38%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
       !     Shared blocks:       29 read,        0 written, buffer hit rate = 3.33%
   ```

   As we know, system blocks (catalog etc.) are loaded when the database starts, so the statistic of the 1st time read is not accurate. I will simply ignore it. For the rest data, we use the number of blocks it reads from the disk divided by (1 – buffer hit rate), then we can get the size of the table in terms of blocks. For example: 29 / (1 – 3.33%) = 30. We definitely need to change the buffer cache size to verify the result.