SIMULATING BUFFER STRATEGIES

Objective:

The objective of the given project is to simulate a buffer pool that can be used for simple Join/Selection queries on a few small tables. The buffer pool supports buffer manager strategies like LRU/MRU/CLOCK blocks. The main purpose of the project is to compare the performance of these buffer manager strategies in terms of the number of disk I/O operations required. By simulating different buffer manager strategies and analysing their performance, this project aims to provide insights into the most efficient way of managing buffers for small tables in a database system.

Assumptions:

Only one block is transferred in one disk i/o
Size of block in buffer pool is same as size of block in page file
We assumed the block (or page) size
All records of a table are in single page file
Considered only simple selection and join queries for this simulation
Join query contains only two tables

Methodology:

SQLite was used to find the number of blocks (or pages) in a table (a page file)
We declared two structures namely buffer_block and buffer_pool. Buffer pool will contain buffer blocks (implemented as linked list).

buffer_pool structure contains fields namely head, tail (Pointer to the head and tail of the linked list), size (number of blocks in the buffer pool), num_valid_blocks (Number of currently loaded blocks), num_disk_reads (Number of disk reads performed).

Buffer_block structure contains fields namely table_name (Name of the pagefile this block belongs to), page_number (page number in a pagefile), pin_count (number of times the block is pinned), clock_referenceBit (it is used in clock replacement strategy), next, prev (pointers to next block and previous block in linked list)

First, we initialise the buffer_pool using function buffer_pool_t *init_buffer_pool(int size) ,take the Sql query as input and then extract table names present in the query using function vector<string> extractTableNames(string query).

Now, we iterate over all tables and find the number of blocks present in those tables each. Number of blocks is given by ceil (num_rows/ (BLOCK_SIZE/record_size)) and num_rows , record_size is given by the function int get_table_info(const char* db_file, string table_name, int* numb_rows, int* record_size)

If the number of tables is one then it is a **simple selection query**. We will iterate over all the blocks present in a table, i.e we simulate memory requests for each block in the table.

Processing of query:

for each block *i* of *instructor* **do**

    Load i into buffer pool

end

We will take a block and search it in the buffer pool using the function buffer_block_t * search_in_pool (string table_name, int page_number), if the block is found then decrement the number of blocks left to be searched by one. If it is not found and also the number of blocks in buffer pool is less than the maximum number of blocks then allocate space for new block by using buffer_block_t *allocate_block (int block_id) and load the details of the block which we have searched into the new block using function load_block (buffer_pool_t *pool, buffer_block_t *block, string table_name, int page_number)

1. Most recently used replacement strategy:

   It calls buffer_block_t *mru (buffer_pool_t *pool, string table_name, int page_number), which chooses the block which is most recently used (the block having highest timestamp) by iterating over all the blocks.
   That block will be evicted by function void evict_block (buffer_block_t *block).
   We will load the block of the table into the buffer pool by using the function void load_block (buffer_pool_t *pool, buffer_block_t *block, string table_name, int page_number) and return the allocated block.
2. Least recently used replacement strategy:

   It calls buffer_block_t *lru (buffer_pool_t *pool, string table_name, int page_number), which chooses the block which is least recently used (the block having lowest timestamp) by iterating over all the blocks.
   That block will be evicted by function void evict_block (buffer_block_t *block).
   We will load the block of the table into the buffer pool by using the function void load_block (buffer_pool_t *pool, buffer_block_t *block, string table_name, int page_number) and return the allocated block.
3. Clock replacement strategy:

   It calls buffer_block_t *clock_replacement (buffer_pool_t *pool, string table_name, int page_number).Initially, the clock hand will be pointed to the head of the buffer pool and iterates over the pool by moving the clock hand. If the current block is pinned then the clock hand is moved, if the current block is not pinned and clock_referenceBit is one it will be set to zero and move to the next block. Otherwise the block will be evicted by using the function void evict_block (buffer_block_t *block) and We will load the block of the table into the buffer pool by using the function void load_block (buffer_pool_t *pool, buffer_block_t *block, string table_name, int page_number) and return the allocated block.

If the number of tables is two then it is a **simple Join query**.

Processing of query:

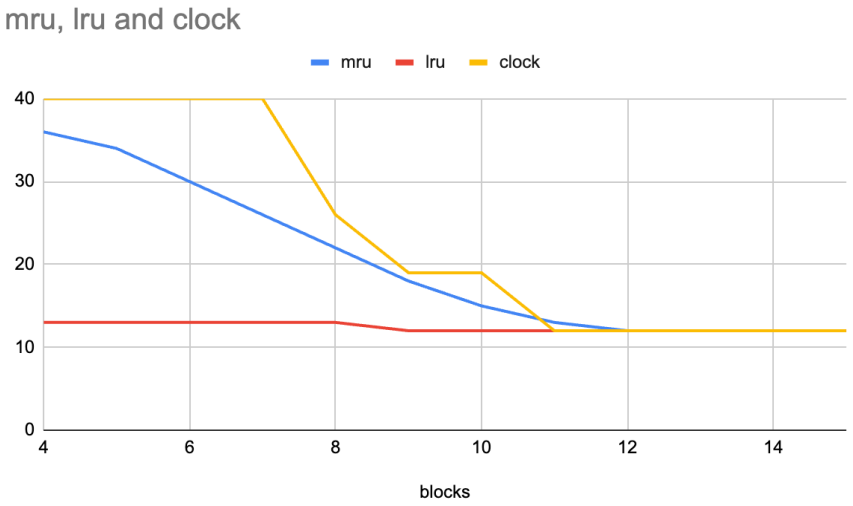
for each block i of instructor do
  Load i into buffer pool and pin, if already present just pin
  for each block d of department do
    Load d into buffer pool and pin, if already present and pin
  end
end

The algorithm we used here is:

We will take a block from table1 and check if the block is present in the buffer pool. If not we will load the block into the buffer pool by using replacement strategies as mentioned above and pin the block. Now we will iterate over all the blocks of table2 by checking if the block is present in the buffer pool or else load the block into the buffer pool using the abovementioned replacement strategies. Now, unpin the block when we have pinned earlier by using the function void unpin_block (buffer_pool_t *pool, buffer_block_t *block). Similarly, take the next block and proceed, iterate over all blocks in a table1.

Observations:

Disk i/o s vs Number of blocks in buffer pool for table1 **JOIN** table2 query
Considering number of blocks in table1 as 5, in table2 as 7



As the number of blocks in the buffer pool increases, the number of disk I/O decreases, resulting in better performance. This is because a larger buffer pool can store more data, reducing the number of cache misses.

The MRU and LRU algorithms perform similarly, but LRU tends to perform better when the access patterns are more sequential or have temporal locality. This is because LRU takes into account the recency of access, while MRU only considers the most recently used block.

The Clock algorithm performs differently from both MRU and LRU. It uses a circular buffer to evict blocks and is useful for handling a mix of sequential and random accesses. The clock algorithm can also reduce the overhead of maintaining the LRU information.
With the increase in number of blocks for , number of disk i/o s for MRU decreases significantly compared to LRU , decrease in number of disk i/o s for LRU is less compared to MRU and clock

Used tables:

```
sukeerthirajeevi@sukeerthis-MacBook-Air termProject % sqlite3 mydatabase.db
SQLite version 3.39.5 2022-10-14 20:58:05
Enter ".help" for usage hints.
sqlite> .tables
Persons  users
sqlite> SELECT * FROM users
   ...> ;
1|Paul|32|California|20000.0
2|paul|34| cal|43212.0
3|paul|34| cal|43212.0
4|paul|34| cal|43212.0
5|paul|34| cal|43212.0
6|paul|34| cal|43212.0
7|paul|34| cal|43212.0
8|paul|34| cal|43212.0
9|paul|34| cal|43212.0
10|paul|34| cal|43212.0
11|paul|34| cal|43212.0
sqlite> SELECT * FROM persons;
1|sukee|rajeevi|something|kadapa
2|sukee|rajeevi|something|kadapa
3|sukee|rajeevi|something|kadapa
4|sukee|rajeevi|something|kadapa
5|sukee|rajeevi|something|kadapa
6|sukee|rajeevi|something|kadapa
7|sukee|rajeevi|something|kadapa
8|sukee|rajeevi|something|kadapa
9|sukee|rajeevi|something|kadapa
10|sukee|rajeevi|something|kadapa
11|sukee|rajeevi|something|kadapa
12|sukee|rajeevi|something|kadapa
13|sukee|rajeevi|something|kadapa
14|sukee|rajeevi|something|kadapa
sqlite>
```

Number of blocks in buffer pool : 10

```
sukeerthirajeevi@sukeerthis-MacBook-Air termProject % ./apr
num of valid blocks=0
Enter SQL query: SELECT * FROM users JOIN persons ON users.id = persons.id
query=SELECT * FROM users JOIN persons ON users.id = persons.id
size of tables=2
users
Total number of records: 11
Final size of record= 24
number_of_blocks in table users = 5
persons
Total number of records: 14
Final size of record= 29
number_of_blocks in table persons = 7
number of clock disk reads=19
number of lru disk reads=12
number of mru disk reads=15
```

Link to code:

https://drive.google.com/file/d/104ENfxWpRlfAjTmp6SM5Omd8v7y9anGV/view?usp=sharing