myPTA Data Manager Documentation

team 5

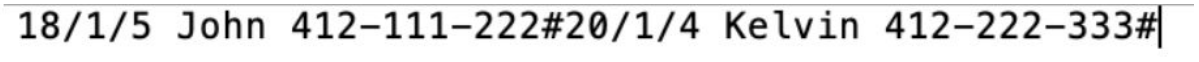
1. For the Sequential Files strategy:

A. Disk:
- For every new table a new directory will be created inside a disk.
- If record is written to table X, directory X will be created in the disk.
- Inside this directory X, text files will be stored which represent slotted pages like X1, X2 etc.
- Inside these slotted pages records will be placed.

B. Main memory:
- Inside the main memory buffer is used as a cache and there are metadata files present with it.
- Metadata files include:
    1. Hashmap for buffer content
    2. File which includes the current slotted page in every table
- For every operation like read, update, delete slotted page will be bought inside the buffer from main memory.
- Buffer has limited space so when it is full LRU(Least Recently Used) mechanism is implemented.
- Buffer can have 10 slotted pages.

C. Table Files:
- For every table a text file is created in the disk which represents the table.
- For example for table X there will be X.txt which consists of all the records inside table X.

D. Slotted Page Approach:
- Inside slotted page records will be stored sequentially.
- At the start we will store the length of the record.
- After that one bit is stored which represents whether the record is deleted or not.
- If it is not deleted, this bit will be 1. On deleting the record, this bit will be made 0.
- After that the whole record will be stored completely.
- Size of each page is 51 bytes.
- Exact approach is as shown in below image:
- So the 18 is the size of the tuple, 1 indicates record is not deleted, after that is there is tuple and # represents end of the record



E. Operations:
    1. Writing a record to table:
        - For this command first it will check whether the table is already created or not. If it is not created a new directory will be created as explained above in the disk.
        - If the table is already created it will go through the metadata file to check what is the current slotted page for the respective table.
        - If there is enough space inside current slotted page for that table it will be brought inside buffer and record will be written
        - If not, new slotted page will be created and brought inside the buffer and then record is written
        - After that the metadata file will be updated with the latest slotted page number for that table.

    2. Updating a record in the table:
        - If the record is already present in the slotted page and it is to be updated the corresponding slotted page will be brought inside the buffer from disk.
        - At the end and at the time of slotted page eviction it will be flushed to disk.

    3. Deleting a record in the table:
        - If the record is to be deleted, the required slotted page will be found in the disk.
        - That page will be brought inside the buffer.
        - The bit which is used at the time of creating a tuple to check if a record is deleted or not is made 0.
        - This bit being 0 means the record is deleted.
        - Information of deleted space will be stored inside another metadata file. This will be helpful to insert a new record if it fits the hole.
        - As reorganization of records inside slotted pages is a costly operation.
        - So we have not done that instead we are keeping holes as it is.
        - This will increase internal fragmentation but this is still much more efficient.

    4. Deleting a table:
        - Directory for this table from the disk will be deleted.
        - All the pages inside the buffer related to this table will be deleted.
        - Table file created for this table will also be deleted.
        - After deletion any access to this respective table will result in error.

5. Retrieving the record with primary key:
    - First check if any page inside the buffer has the required record
    - If it is just retrieve it
    - Else go through pages inside disk and bring required page to buffer
    - Then required tuple will be returned as a answer

6. Retrieving the record with area code:
    - When the page is brought inside buffer hashmap will be created for that page for area codes
    - Based on the hashmap searching will be faster
    - Records will be retrieved based on these hashmaps
    - For that particular table every slotted page will be swapped inside memory if not present inside the buffer

F. LRU implementation:
    - LRU is implemented with an array list
    - When the buffer is full we need to use LRU implementation to evict the page from the buffer.
    - Least recently used pages will always be in the first position of array list.
    - So for every eviction first entry from the array list will be evicted
    - If the page is already in the buffer then that entry will be deleted from list and put it in the last position of array list
    - So LRU page is in the start of array list and MRU page in the end of array list

G. Testing Strategy:
    - We are attaching the test scripts for the correct functioning of all 5 operations in the submission.
    - We had different scripts for all operations and we checked results of those queries.

2. For the LSM-tree strategy:

a. Description about the record placement, file organization, and memory management strategies.

i. record placement

There are 3 places in our system that contain records: memtable in the memory, cache, and the disk. The memtable is a treemap, all the new input data(from action W, E, D) will first be added into the memtable in the form of ID: [ClientName, Phone]. The cache is a list of ListNodes, each ListNode has a key which is the ID of this record, and a value which is the relevant [ClientName, Phone], and it also has a property tableName indicate which table this record comes from. In the disk, each sstable is stored in a single .txt file, the data structure we use to store data in each sstable is also the treemap, which ensure the orderliness of the records in each table.

ii. file organization

All the sstables in the disk are represented as .txt files, the name of the files indicate the level, tableName and the No. of its sstable, for example,the sstable in file 1X3.txt is the 3rd sstable of table X in level 1. Each file contains only one sstable, and the sstable is represented as a treemap, the key of each node in treemap is the ID of each record, and the value of each node is [ClientName, Phone]. Each level has a limitation on the number of sstabls, for example, there are at most 4 SSTables in level-0, and at most 10 SSTables in level-1.

iii. memory management

In the memory there is a memtable which is implemented using treemap. Whenever we do action W, E, and D, we will write related record into the memtable. Once the memtable is full, it will trigger flush and move the records in memtable into disk level 0. Whenever we do action R, we will search for the wanted record in the order of memtable, cache, disk.

b. The design and components of the row key (i.e. the key for each record).

Since the Primary key of each record is ID, we use the ID of each record as its key. Each record is stored as a key-value pair, where the key is the ID of each record, and value is a list [ClientName, Phone] of each record. Specially, if we want to erase a record, say the record whose key is 5, then we will append a record 5:[deleted] to the table.

c. The design of your read cache, and the methods to keep it consistent.

The read cache is a doubly linked list consist of listNodes, each listNodes has 3 property, key, which is the ID of the record, value, which is a [ClientName, Phone] pair, and tableName, which indicate which table this

record comes from. In the linked list has two special index: dummy and tail. Dummy point to the head node of the current linkedlist and tail point to the last node in the list.

The LRU replacement is implemented by this:

In our linked list, the head node is the record that least recently used, and the last node is the record that we have just used.

Whenever we read a record in the cache, we move this record(node) to the end of our linked list and let pointer tail point to this record.

Whenever we read a record in the disk, we add this record(node) to the cache.

Whenever we want to add a new record to the cache, if the cache is not full, just add this node to the end of our list; if the cache is full, then deleted the node pointed by pointer head, and let head point to the next node of the previous head, by doing this, we can delete the record that least recently used, and then add the new record to the end of our list.

The cache consistent is kept by this:

Whenever we write a record to the memtable, we will check whether this record exists in the cache. If in the cache there is a record that has the same tableName and key(ID) as the new one, then we will set the value of this node to the latest value. Thus we can keep the cache consistent.

d. For operation W: Describe all processes that could be triggered by an operation W. Detailed description of the compaction strategies (e.g., how many and which SSTable(s) to pick for compaction in each level?).

Whenever we do an operation W, we will first add this record to memtable, since the memtable is a treemap, the record is added as a key-value pair, where key is the ID, value is [ClientName, Phone] list. Then if the records in memtable is able to fulfill a sstable, we will trigger flush and move all the records in memtable to an immutable memtable, and then create a new level 0 sstable to store these table. At the same, we will check whether the level 0 is full. Since level 0 can have at most 4 sstables, if there are already 4 sstable in level 0, while create the new sstable, we will trigger compaction. For compaction, once the lower level of LSM tree is full, we will use the all the SSTables in the lower level, and combine these SSTables with the existed SSTables in the higher level (if there is no SSTable in the higher level then skip this step). Once we finish the combination, we push the result into the higher level, and then check whether the higher level is full. If it is full again, then we do the above steps again.

e. For operation E: Describe all processes that could be triggered by an operation E, which are different than those by an operation W.

When operating operation E, we will add a special record to memtable, the record is also a key-value pair, its is the ID, value is [deleted] list. After add this record, we also need to check whether we need to trigger flush, if the records in memtable is able to fulfill a sstable, we will trigger flush and move all the records in memtable to an immutable memtable, and then create a new level 0 sstable to store these table. At the same, we will check whether the level 0 is full. Since level 0 can have at most 4 sstables, if there are already 4 sstable in level 0, while create the new sstable, we will trigger compaction.

The difference between E and W is that, the record they append to the memtable is different, what W append is: key: [ClientName, Phone], while what E append is: key: [delete]

f. For operation D: Describe all processes that could be triggered by an operation D.

When operating operation D, In memory, we will clear the memtable of this target table and then add delete: [] to this empty memtable.

For the SSTable, once we see a D operation, we will add a new delete node into the SSTable. If we need compact this operation with a existed record in the higher level, we will update that record into ‘deletion’, and when the record reach to the last level, the record will be deleted completely.

g. For operation R: How to guarantee the correctness of the read under different conditions (e.g., multiple updates, deleted records, deleted tables).

When operating operation R, we will search the record in the order of cache,memtable, and disk.

i. search the cache, since the cache is a linked list, we will do our search one by one from the head node of the list, check whether the current node has the same tableName and key as we wanted, if hit, return this record and move this node to the end of our linkedlist, if doesn’t hit, check the next node. If we reach the tail node of our linkedlist, it means that there is no such record in our cache. So then we will search our memtable.
ii. Checking whether the memtable of the target table contains this key, if it does contains, just return this record and the R operation is finished. Otherwise, search the disk.
iii. We need do the disk search level by level, the first search begins at level 0, to better illustrate, I can show a example here.

Assume the name of sstable is construct using level + tableName + No. of table(table 1X3 means that it is the third sstable of table X in level 1)

The larger the No. of sstable is, the later it is created, so the sstable with larger No. has newer record then sstable with smaller No..

Therefore, everytime we search sstables in a level, search the sstable with largest No. first, then search table with the smaller No.

Assume now we have table 0X1, 0X2, 0Y1, 1X1 in disk.
And we are going to run R X 5.
step1: search 0X2, get this sstable and check whether it contains a record whose key is 5, if hit, return this record, operation R finished. Otherwise, go to step 2.
step2: search 0X1, get this sstable and check whether it contains a record whose key is 5, if hit, return this record, operation R finished. Otherwise, go to step 3.
step3: Since we have finished searching all X's sstable in level 0, we will begin to search level 1, get this sstable in 1X1 and check whether it contains a record whose key is 5, if hit, return this record, operation R finished. Otherwise, since it is the last X's table in level 1, and there is no next level such as level 2, we can say that the wanted record doesn't exist in our database, return null.

We only return the qualified record that we first hit, since the first record we hit is the latest record. while getting the data of this record, if the value of this record is [ClientName, Phone], the size of the value list is 2, it means this record is valid, if the value of this record is [deleted], the size of the list is 1, it means this value has already been deleted.

For faster retrieval, we have created a HashSet named deletedTable, whenever we do operation D, we will add the tableName to this set. Therefore, when we are doing R, we will first check whether the target table is in deletedTable set, if it is, it means this table has been deleted and we don't need to future search. If it is not in the set, it means this table is valid and we need to do the search procedure described above.

h. For operation M: Have you implemented any methods other than the full table scan to do it?

We haven't implemented a method that can complete M without a full table scan. But we do have some ideas that may works. For example, we can store an index of area code in each sstable, and this index will indicate the location of records that have specific area codes. In this way, whenever we do the M search in sstable, we will first search the index, and find out the location of records that have wanted area code, so that we can get the desired record directly without scan the whole sstable.

i. Any other implemented optimization methods, e.g., additional index files, auxiliary memory objects, parallel compaction, etc.

For the SSTable, when we do the compaction, we will use 2 memory blocks: emptying block and filling block. We will put the data we need to compact to the emptying block to sort, then push the result into the filling

block. In the meantime we can delete the data in c0 node as we already read them into emptying block, so we can save memory. And when the filling block is full, we transfer the data into the c1 node. In this way we can do sorting merge in a really fast speed.

j. Test cases indicating the correctness of the four operations under different conditions.
The input script is : DBMS_project/LSM/src/com/lsm/script
All test cases are in the input script

3. For the comparison between the two file organization strategies:

a. The comparison and analysis of the read/write throughput (records per second) between the two strategies under different combinations of buffer sizes and SSTable sizes.
   i. For the LSM-tree strategy
   Test cases:
      1. buffer size: 2
         sstable size：2
         read throughput: 203.39
         write throughput: 923.07
      2. buffer size: 2
         sstable size：4
         read throughput: 210.53
         write throughput: 1090.9
      3. buffer size: 4
         sstable size：2
         read throughput: 230.76
         write throughput: 923.08
      4. buffer size: 4
         sstable size：4
         read throughput: 206.89
         write throughput: 1090.91

   Conclusion:
   For LSM strategy, the throughput of write is much larger then read, which means the write operation can be down much faster then read.

The write can be done very quickly because the write is simply an addition. Every time we write a record to our database, just append it to the memtable. However, read can be complicated and time-consuming, as related records may be stored sporadically in several tables.So, if the data we want was stored long ago, for example, in level 100, then we need to search all the tables in all 99 levels above it to find it.

4. For any implemented optimization methods of the LSM-tree strategy:

a. The comparison and analysis of the read/write throughput(records per second) between the LSM-tree strategy without the optimization and with the optimization.

For the compaction algorithm, one way is to first load all needed data, then we do the sort, and finally write it into the SSTable. But an optimized way to do it is that we can use some blocks to finish this job in a parallel fashion. We will only write some blocks in the lower SSTable into the blocks and do the sorting job before we get more data. And once the block is full, we flush it into the disk and do the next sorting. By using this method, we can do the sorting and reading in the meantime, so this method can increase the throughput of write.

5. Others

a. Detailed description of the contribution of each team member based on the implemented components (e.g., for the LSM-tree strategy, who implemented the memtable, the SSTable, the compaction algorithms?).
   i. Sequential Files strategy(Sudeep and Antony)
   ii. LSM-tree strategy(Ziyi Huang and Lingrui Ouyang)
      1. Memtable: Ziyi Huang
      2. Cache: Ziyi Huang
      3. SStable: Ziyi Huang and Lingrui Ouyang
      4. Compaction: Lingrui Ouyang