User Id: __Zhenyu Wu zhw323

CSE 303: Quiz #3

Due Friday November 4th, 2022 at 11:59 PM

The quiz has THREE questions. Please submit your answer on CourseSite as a pdf whose name is exactly your user Id and the "pdf" extension (e.g., abc123.pdf) before the deadline.

Question 1: Consider the hash table we had in P2, implemented as an array of buckets. In each bucket, there is a lock and a list of key/value pairs. A pair is located in the bucket by using some hash function on its key to choose the appropriate bucket in the array. Now suppose you want to add a `move` operation to this hash table, defined as follows:

```
/// Given a key k1 and a key k2, if there exists a mapping from k1 to some value v, and there is no
mapping from k2 to any value, then eliminate k1's mapping, and create a new mapping from k2 to v.
///
/// NB: This method must appear to be atomic from the perspective of concurrent insert, remove,
upsert, do_with, do_with_readonly, and do_all_readonly operations
///
/// @return MOVED if the move happens, NOVAL if k1 did not have a mapping in the hash table, and
NOROOM if k2 already had a mapping.
```

Provide a pseudocode for this operation (no need for a full C/C++ code, only a pseudo code that highlights the main functionality). **It must handle important safety and progress pathologies.**

```
//Create global atomic bool
Std::atomic <bool> hasVal (false)
Std::atomic <bool> hasRoom (true)
//Function below
Void move(k1, k2):
//Firstly, use the prehash(a helper function which inplements hash functions) to find the position of pairs
Pos1 = Prehash(k1)
Pos2 = Prehash(k2)
//Lock k1 bucket to check pair 1
lock_guard<mutex> guard(kvstore[Pos1]->Lock);
V val1
//Iterate through array of k1 bucket
Iter1 = kvstore[Pos1]->pairs.begin();
while(iter1 != kvstore[Pos1]->pairs.end())
        if iter1->first == k1
                if iter1->second != null
                        hasVal = true
                        val1 = iter1->second
                        kvstore[Pos1]->pairs.erase(iter1)
                        break
        iter1 +=1

//Lock k2 bucket to check pair 2
lock_guard<mutex> guard(kvstore[Pos2]->Lock)
//iterate through array of k2 bucket
Iter2 = kvstore[Pos2]->pairs.begin();
```

```
while(iter2 != kvstore[Pos1]->pairs.end())
        if iter2->first == k2
                hasRoom = false
                break
        iter1 +=1

if hasRoom and hasVal
        hasRoom = false
        hasVal = false
        kvstore[Pos2]->pairs.push_back(make_pair(k2, val1));
        return MOVED
        both lock unlock
if !hasRoom
        hasRoom = false
        hasVal = false
        return NOROOM
        both lock unlock
if !hasVal
        hasRoom = false
        hasVal = false
        return NOVAL
        both lock unlock
```

User Id: __Zhenyu Wu zhw323

Question 2: What is one thing you learned about persistence in CSE 303 that you didn't know before?

Please use the remainder of this page to provide your answer to the question. Give a detailed answer, but keep your entire response to a single side of an 8.5x11 page.

One thing that I have learned is how we can achieve Atomic and Durability of ACID of a transaction. Atomic refers to 'do all or do nothing' and Durability refers to 'changes of committed transactions survive crashes'. There are 2 solutions redo-logging and undo-logging. Redo logging includes 4 steps. 1. Append all planned updates(new value) to the log file. 2. Commit(or abort): append a special commit or abort record to the log. 3. Write-back: all of the updates will be written to target location replacing old values. 4. Garbage Collect: reclaim the log. To recover from crash, we just can through the log and only write-back the committed updates. On the other hand, undo logging includes 1. Write old value in undo log, new value in final storage; 2.Reading: read directly from the storage; 3. Commit: Add a commit record to log; 4. Write-back: undo updates by writing the old values. To recover from crash: scan the log, discard all committed records and undo others.

Question 3: Caching and prefetching are two techniques that are frequently employed in enterprise storage systems. Define and contrast the two techniques, making sure to give examples to illustrate both the strengths and weaknesses of each.

Please use the remainder of this page to provide your answer to the question. Give a detailed answer, but keep your entire response to a single side of an 8.5x11 page.

The term "caching" refers to a technique that temporarily saves data so that it can be accessed more rapidly. Either saving the data in a memory region on the machine itself or storing the data on a remote server is two possible ways to accomplish this goal. The process of prefetching is one that anticipates the requirements of the user and downloads the necessary material prior to the user making the request for it(To bring data into a cache before it is needed). Either receiving the data from a remote server or reading the data from a memory location on the system can accomplish this goal. Both methods are viable options.

Caching is a technique that can be utilized to improve the overall performance of a computer system. Caching, for instance, can save time for the system by retrieving the data from a local memory location rather than searching the database when a user makes a request for data from a database. On the other side, prefetching can increase performance by anticipating the needs of the user and downloading the necessary data before the user demands it. This is done by downloading the data in advance. This can increase the system's responsiveness by reducing the amount of time the user has to wait for data, and it can also reduce the amount of time the system needs to wait for data that has already been downloaded, hence reducing the amount of time the user has to wait for data.

Caching has a number of benefits, but one of its drawbacks is that it might bring about a drop in the speed of the system if the data that is cached is no longer current. If, for instance, a corporation keeps older versions of its documents in its memory, caching those documents would result in decreased system performance since newer copies of the documents will be requested instead of the older versions of the documents.

User Id: __Zhenyu Wu zhw323

For prefetching, if a process reads the first two blocks of a file, operating system may prefetch the next ten blocks to improve I/O performance. The advantage of such prefetching includes: reduced latency, reduced device overhead, improved parallelism. When predictions are accurate, prefetching can reduce latency of future request since reads can be serviced from main memory which is faster than storage devices. Prefetching can also replace a large number of small requests with one large one to reduce storage device overhead. Lastly, Prefetching provides a way for operating system to take advantage of available hardware parallelism(RAIDs and Flash drives). The weakness of prefetching includes cache pressure, I/O contention and wasted effort. Since each prefetched block is stored in the block cache, and it may displace another block from the cache. If the evicted block is needed before the prefetched one is used, prefetching is likely to hurt overall performance. For I/O contentions, If there are other requests that have to wait behind prefetch requests while prefetching request is consuming the I/O resources, performance will be hurt. Lastly, if the prefetched blocks are never used, prefetching may hurt overall performance by wasting memory space, I/O device bandwidth, and CPU cycles.