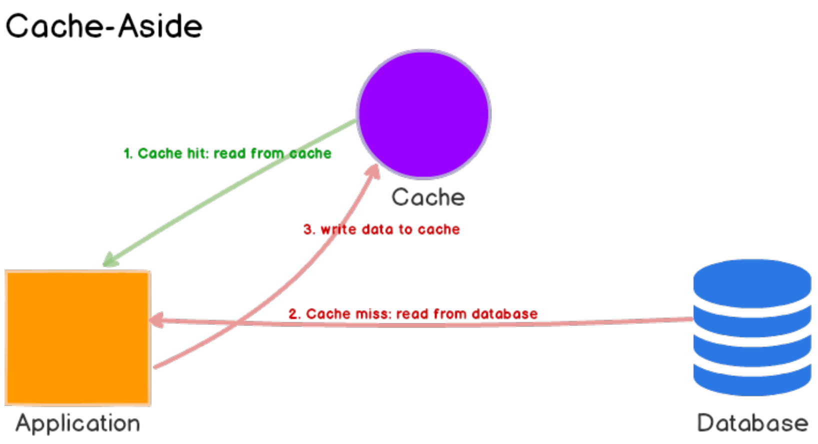
In Memory File Cache Design

Table of Contents

Cache Design

Cache is designed in such a way that it caters both READ HEAVY and WRITE HEAVY use cases.

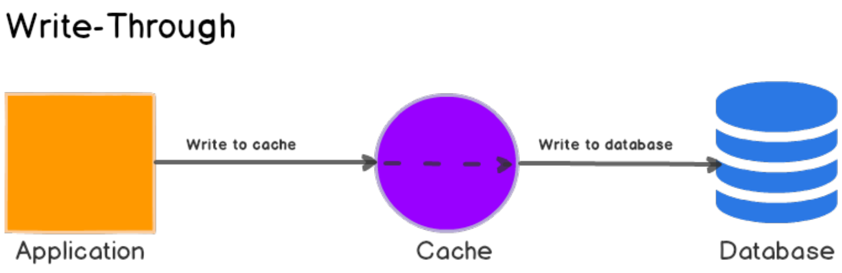
Cache – Aside Read


Cache-Aside
1. Cache hit: read from cache
Cache
3. write data to cache
2. Cache miss: read from database
Application
Database

1. The application first checks the cache.
2. If the data is found in cache, we’ve cache hit. The data is read and returned to the application.
3. If the data is not found in cache, we’ve cache miss. Application reads it from memory and update the cache.
4. This will be resilient in case of Cache failure.

This holds good for READ-HEAVY use case.

Write-Through



1. Application first writes to cache and then to memory.
2. This approach have relatively high “write latency” when compared to “Write-back” or “Write-around” approach.

3. But this approach will nullify any dirty cache scenario in case of READ-HEAVY scenarios.
4. This approach eliminates the need for cache timer/ cache aging logic as we will not hit stale cache entry.
5. This approach will make application more reliable.

Combination of "Cache Aside" read and "Write through" will result in resilient and reliable model in both READ-HEAVY and WRITE-HEAVY scenarios.

Eviction Policy

Models considered

1. Least Recently Used (LRU)
2. Least Frequently Used (LFU)
3. FIFO
4. Most recently Used (MRU)

Least recently used (LRU) is more suitable for our use case.

1. Cache size is very small compared to actual data (from problem statement)
2. FIFO and MRU will not help in case of READ-HEAVY scenario.
3. LFU will need additional cache timer/cache aging mechanism which is an overhead for small sized cache.
4. We use "write through", so no need to worry about cache staleness.

Considering all these aspects, LRU Eviction policy seem to be a better fit for the problem.

Programming Design

Coding is divided into 2 parts. Application specific code in "file_cache.cpp" and "cache" utilities are in "cache.h".The idea is to use this "cache.h" for various cache applications. Hence decoupled application and utility.

DATA STRUCTURES USED (CACHE)

1. List – to store cached values. Its a key,value pair
2. Unordered_map – to store indices of List (Time complexity O(1))

THREADS (APPLICATION)

1. Boost thread library is used to create threads.
2. Boost mutex library (“lock_guard”)is used to handle thread synchronization.
3. Both read and write threads are modifying “cache”.So using “unique_lock” or “reader/writer” lock will not have any advantage.
4. Separate Mutex variables used for cache access and file access.
5. Read and write call backs both use “Cache lock” to avoid race condition with cache list.
6. Write call back uses “file lock” to avoid corrupting item_file.

UNIT TESTING

Application unit testing cases are listed in “`application_unit_Test.txt`”. All the cases passed.

PERFORMANCE TESTING

Optional Performance benchmark option has been designed. If we compile the application with “-DPERFORMANCE_TEST” flag, we will get application performance in terms of CPU time usage.

Application performance has been tested with CPU time, with more no. of threads.

We could see, as the cache size grows, CPU time decreases. This shows the cache performance.

O/P:

```
ram@ram-VirtualBox:~/work/cache/my_cache$ ./cache 1 reader_file.txt writer_file.txt
items_file.txt
  ->  Time taken by application:    0.300 seconds CPU time
ram@ram-VirtualBox:~/work/cache/my_cache$ cp items_file_orig.txt items_file.txt
```

```
ram@ram-VirtualBox:~/work/cache/my_cache$ ./cache 2 reader_file.txt writer_file.txt
items_file.txt
  ->  Time taken by application:    0.293 seconds CPU time
ram@ram-VirtualBox:~/work/cache/my_cache$ cp items_file_orig.txt items_file.txt
ram@ram-VirtualBox:~/work/cache/my_cache$ ./cache 3 reader_file.txt writer_file.txt
items_file.txt
  ->  Time taken by application:    0.274 seconds CPU time
ram@ram-VirtualBox:~/work/cache/my_cache$ cp items_file_orig.txt items_file.txt
ram@ram-VirtualBox:~/work/cache/my_cache$ ./cache 4 reader_file.txt writer_file.txt
items_file.txt
  ->  Time taken by application:    0.266 seconds CPU time
ram@ram-VirtualBox:~/work/cache/my_cache$ cp items_file_orig.txt items_file.txt
ram@ram-VirtualBox:~/work/cache/my_cache$ ./cache 5 reader_file.txt writer_file.txt
items_file.txt
  ->  Time taken by application:    0.256 seconds CPU time
ram@ram-VirtualBox:~/work/cache/my_cache$ cp items_file_orig.txt items_file.txt
ram@ram-VirtualBox:~/work/cache/my_cache$ ./cache 6 reader_file.txt writer_file.txt
items_file.txt
  ->  Time taken by application:    0.249 seconds CPU time
ram@ram-VirtualBox:~/work/cache/my_cache$ cp items_file_orig.txt items_file.txt
ram@ram-VirtualBox:~/work/cache/my_cache$
```

DEBUG INFRA

Optional Debug infra has been designed.If we compile the application with "-DDEBUG" flag, we will get debug log messages.