Distributed Cache Optimization

Dhruv Kumar - 150233, Sonam Tenzin - 150724

CS632: Topics in Distributed Systems

Abstract

Implement distributed system which communicates in a way that assists memcached servers to do speculative caching of popular keys & removal of not so popular key. Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering

1. Introduction

We have created a distributed caching system which uses the cache to store recently and most popular values. This will hugely decrease the response time since memory access is comparatively faster than drive access. This model utilizes a **server and client** model along with a memcache server. User can login into his account, and then can carry out a wide variety of queries which can be of the type "update": which essentially allows the them to add their comments, "search": which allows them to search for any n number of comments made by a particular user. The memcache server which is attached to the main server is accessed using sockets and utilizes the conventional "get" and "set" functions to access the keys and their corresponding values in the server. This server essentially uses the RAM to cache the frequently used values which decreases the accesses made to the drive. Along with this, there is also a database connected to our main server which is comparatively less frequently accessed but serves to store the entire lot of the users and the comments made by them. The database logs all the comments made by the different users in an increasing fashion which further allows us to create an ordering among the comments and help in queries like "search". There can be a wide spread applications of this memcached based model but we have decided to implement it as social-networking application. These comments can be viewed by anyone who knows the username of the

person. The comments posted can also be selectively deleted by the user by specifying the post number and furthermore if the need arises the user can also delete his/her account from the platform.

2. Implementation Details

The overview of the application contains the following elements:

- **Main server**: Responsible for overlooking the connection to clients, the database and the memcached server
- **Memcached**: Main server acts as a client and sends "get" and "set" queries for addition or removal of cache entries.
- **Database**: The records of all the comments made by the user are stored in the database. These are numbered in increasing order of the time in which user made these comments.
- **Client**: The client is comprises of text based terminal which can be used to submit the queries implemented on the server.

2.1. Main Server

The main server is the backbone of the entire application. It serves as main hub where all the queries made by the user are executed. It uses python and its libraries to provide basic functionality. The server works by first of all forming socket connection with the database and the memcached server and then starting the listening service to accept any of the incoming client connections. The clients will connect to this server by typing the specified port number in order to make use of the services. On receiving a request from the client the server will fork a child process to handle it and then the parent process will go back to serve other requests from clients thus allowing it to serve more than one client simultaneously. The child process will start working on the query supplied by the user which will require access to the database or the memcached server depending on the type of query. After returning the result, the child process will close its sockets and exit.

Different queries implemented on the server:

- **UserExists**: This is an internal query which is not explicitly specified by the user. The main objective of this query is to check whether the user is a new or an old user. In case that it is a new user, it will

automatically create a new user(for this the user has to specify that he is a new user)

- **Search**: This allows him to know the last n (specified as an argument) comments made by a specific user. But for this to function, it is necessary apriori to know the username of the person to be searched for.

- **Update**: This enable the user to write a new comment. The comment posted by this query will not have any expiry time which won't be delete from database unless the user explicitly run the delete command.

- **Deleteme**: This query allows the user to delete its entire footprint from the database along with any data relevant to him in the memcached. Some restraint is required while using this command as there is no going back after the deed is done.

- **DeleteI**: This enables the user to delete any post which he/she may feel are not relevant to him/her anymore. It allows the user to handpick comments one by one. The user can also specify a range in which the comments which lie in a particular range can be deleted. Again there is no coming back once the command has been issued as the comments will be deleted both from the cache and the database.

- **getLatest**: This query relies heavily upon the memcache. It essentially allows the user to show all the posts which have been posted to the server in the last 20 seconds. This essentially allows us to implement this function without any extra costs and worrying cache becoming full.

Since there are updates and reads are being done by different processes, it is necessary for a locking mechanism to exist. Otherwise, the entire system would be have an unexpected behaviour.

2.2. Concurrency Control

Locking Mechanism is implemented making use of both cache and database. If a process tries to execute an update or delete command, then it checks whether the lock corresponding to that particular user exists in the cache or not. If it exists in the cache, then it waits unless the lock is evicted or removed. When it founds that the lock for that user is no longer in the cache, it attempts to update the value of the entry for that user in

'locks' database only if the value if already 0 (denoting there is no lock). If the affected_rows is 0 this means that another process has already attempted a lock in this race condition, the current process backs off. Otherwise if it is able to successfully update that entry corresponding to the user, whose value is 0, to value 1, then it adds lock into memcached. It now executes the update/delete for post of that user in mutual exclusion. After he has completed is task, he release lock by first removing the entry from cache, then updating the entry of database to 0, denoting that the lock is released.

2.3. Distributed Cache

The frontEndServer would face all the client requests and would redirect it to one of it's available nodes who would have the information for successful execution of the query. All the information about posts, deleted posts and total posts are kept on that choosen node. The node is decided in the beginning at the creation of user, decided by the value of uniform hash taken over the username

2.4. Memcache

Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) which can be used to store the results of database calls, API calls, or page rendering. And in a similar fashion we are also using it by caching the query results. It is implemented upon the python library pymemcache which essentially allows us to connect our main server as a client to the memcache server. It utilizes the main memory or the RAM of the systems to provide the caching functionality. The key and value pairs can be specified using the "set" and "add" function. Both the functions differ in the way that, "add" only allows creation if the keys is not already present in the memory cache while the "set" allows overrides the key if it is already present. Along with this there are other functions like "delete", "append", "get", etc. These functions are utilized by setting the userhash, which is essentially a string made up of username and the current comment number, and the message, which is the comment which is written by the user as the (key, value) pair for the memcache entry.

The format of storage is as follows:

KEY	VALUE
username#0	The number of last undeleted post made by the user
username#i $\forall i > 0$	A post made by that user
username#i $\forall i < 0$	Denotes the list of post no.s from $100*\|\|i\|\| - 1$ to $100*\|\|i\|\|$ deleted by the user

While setting the values of the keys, we can also specify the tentative time to live for each of the keys. This has been extensively used in the "getLatest" function implemented on the main server. The function uses a special type of key which is made up of the string of the word "latest" and the time it was updated. The value corressponding to this key is the "userhash". Whenever the update function is called, this special key is updated and another feature of the key is that it is has its TTL set as 20, which essentially means that it will get evicted from the cache after 20 second. The "getLatest" function works by sending all the latest keys which have been set in the last 20 seconds of the funtion being called.

2.5. Database

The database is implemented on MYSQL and uses the python "_mysql" library to connect to the database. There are two tables present in the database which are namely "status" and "locks". The status table essentially stores "userhash" and the comment associated