International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 2, February - 2014

A Brief Introduction to Memcached with its Limitation

1. Hely Shah, 2. Mohammed Husain Bohara
1,2. Department of Computer Science
and Engineering, Parul Institute of Engineering and Technology, Limda,Vadodara, Gujarat, India.

***Abstract*— Scalability is one of the major issues to be addressed for Software as a Service. It can be achieved by using proper object caching mechanism. Memcached is one of the available distributed cache which is used for internet applications. memcached uses scale out approach, that is adding more servers to system for dealing with increasing load. This paper explains basics of memcached. It covers its architecture, data structures, operations and process flow. Some limitations are also addressed in this paper so that it can be studied and used to develop more scalable architectures of memcached which can give better performance.**

I. Introduction

Memcached[6] was designed by Danga interactives to reduce the number of database hits by caching the most commonly used objects. Originally developed at Danga Interactive for LiveJournal, the Memcached system is designed to reduce database load and speed page construction by providing a scalable key/value caching layer available to all webservers[4]. For increasing scalability memcached is used by many websites with heavy traffic like facebook, youtube, twitter, raddit etc. In next section memcached architecture is described with its basic usage. The data structures used in memcached are described in detail. How these data structures are proper for memcached is also described. In Basic Operations section get, set and delete are described and some other operations are also listed in the same. In the next section process flow of memcached is explained with example. While memcached already offers excellent performance, memcached architecture can be studied on some of its limitations and can give better results than of now. These limitations are related to data structures and some characteristics of memcached. These limitations are caused due to which characteristics of memcached is also discussed in the same section with logical reasons.

II. MEMCACHED ARCHITECTURE

Memcached is designed to scale from a single server to many servers using horizontal scaling approach also known as scale. It uses ‘c’ programming language. It has mainly two components. Memcached client and memcached server.

A. *Memcached Server Instance*

In distributed caching, Memcached server is a standalone process that basically handles three tasks [1].

- It manages memory allocation and restoration
- It keeps track of objects stored in memory
- It serves client requests regarding object retrieval and storage

B. *Memcached Client Instance*

Memcached client are intended for providing a common way of accessing the memcached server. Their main objective is to serialize objects, which will be stored in the cache. As the caching system is meant to be distributed, a client can connect to multiple servers by specifying their IP addresses and ports. Multiple clients accessing the same objects must have the same list of servers because the location of an object is determined by hashing the list of servers together with the object’s key.[1] Simple and lightweight client libraries have been implemented in different programming languages like C, Java, Python, Perl etc. There are number of client and server instances are possible at one time for large data set.

III. DATA STRUCTURES

Memcached has four main data structures.

A. *Hash table*

The hash table data structure is an array of buckets as show in figure 1. Each bucket is a single linked list which stores data item. To determine the hash value quickly bitwise end is executed between hash_value and hash_mask. Hash mask is 2^k-1, where k is size of the array. Hash function is used to accelerate table lookup or data comparison tasks such as finding items in a database.

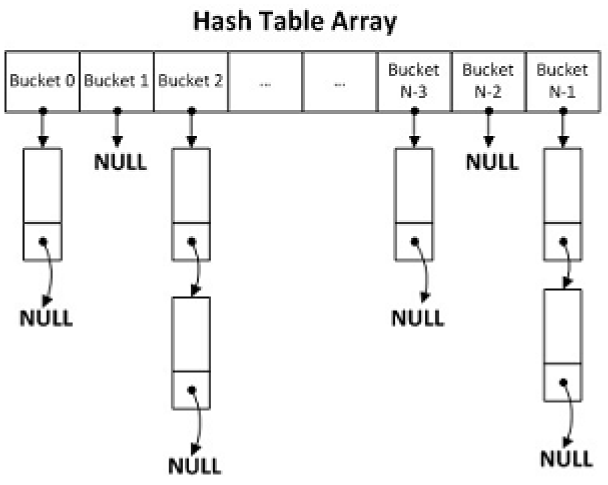

Fig.1 Hash table in memcached [2]

B. *Least recently used (LRU) list*

Memcached has a structure that is known as an LRU(Least Recently Used) cache, so that stored data that is the oldest and least accessed will be replaced with newer items when memcached's memory capacity is reached[3]. LRU list is stored in form of doubly linked list. When an item is requested from the memcached it is removed from its LRU list and putting it on the head of LRU list. Thus by above procedure the least recently used item will come to tail of LRU.

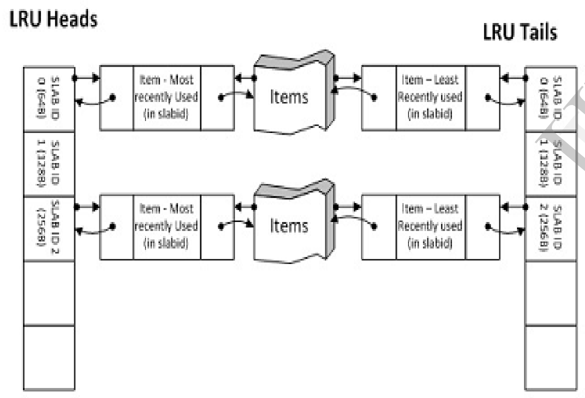
LRU Heads
LRU Tails
SLAB ID 0 (64B)
SLAB ID 1 (128B)
SLAB ID 2 (256B)
Item - Most recently Used (in slabid)
Items
Item - Least Recently used (in slabid)

Fig 2: LRU list for cache item eviction [2]

So when there is a need to remove some item from memcached the item at tail is removed as least recently used. Figure 2 explains cache item eviction using LRU in memcached.

C. *Cache item data structure*

The cache item data structure holds the key-value pair data[2]. Information like The key, value, length of the value, pointers used in hash table and LRU, counter that determines how many threads are accessing cache item etc are stored in this data structure.

D. *Slab allocator*

Instead of conventional memory allocation like malloc and free, memcached uses slab allocator for memory management. Memory management in memcached is described in [5]. As shown in figure 3, let us imagine 2048 bytes of memory available for the distributed cache. We can divide this memory in 4 slab classes of 512 bytes each. The first slab class is intended for slabs with 64 bytes, second for 128 bytes and so on. The object will be stored into the first slab where it fits in and will occupy the whole slab. In this case an object of 50 bytes will actually use up 64 bytes, as 14 bytes will remain unused. As soon as the object is deleted, its slab is added to the freelist. Space is wasted in this concept but it saves time and avoid thrashing.

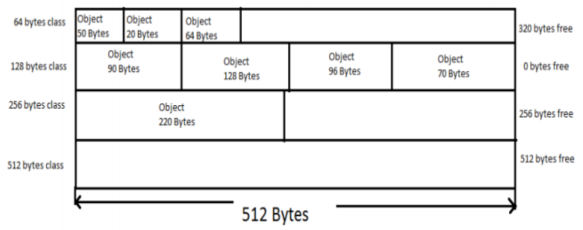

Fig 3. Slab allocator in memcached [1]

IV. BASIC OPERATIONS

There many operations in memcached to deal with data. Memcached provides a simple set of operations (set, get, and delete) that makes it attractive as an elemental component in a large-scale distributed system [5].

Set : Value which is to be stored on cache is passed with an key with help of this function. Memcached client node wants to set the key "test_key" with the value "test_value". Memcached client node takes the list of available memcached server nodes and performs hash operation on them in order to find the memcached server node to store particular value for particular data.Proper Memcached server node is selected. Memcached client node directly connects to memcached server node and sets key "test_key" with the value "test_value".

Get : Memcached client node wants to get key "test_key". Memcached client node is able to use the same hashing process to determine that key "test_key" is on which server. Memcached client node directly requests key "test_key" from memcached server node and gets back "test_value".

Delete : Memcached client node wants to delet key "test_key". Memcached client node is able to use the same hashing process to determine that key "test_key" is on which server. Memcached client node directly requests key "test_key" from memcached server node and deletes vallue "test_value".

Replace, increment, decrement are some other important operations of memcached used for different purposes.

V. PROCESS FLOW

Memcached servers seat between web browser and database as shown in figure below.

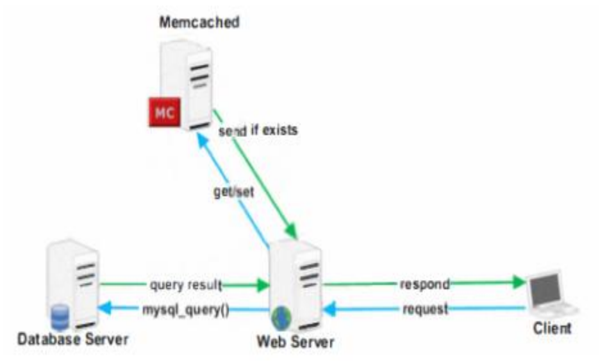

Fig 4. Memcached as layer between client and database [8]

Client checks on memcached for the value. If the value is available then it is sent back to client. If the vale is not available on memcached then it is fetched from database, stored on memcached and then returned to client. Thus memcached speeds up the read write operation for client.

Request arrives and is processed by libevent client library.

VI. LIMITATIONS IN MEMCACHED

Memcached has provided very efficient solution for scalability issue, but still there are some of its characteristics which can be reviewed so that more efficient caching solution can be developed.

A. *Non-persistent Cache*

Sudden failure or offline maintenance will cause the data loss from memcached server node.This causes unpredictable degradation of application performance because of loss of data. Since the new memcached server node is empty at the start, all data requests need to be serviced by the database servers in the RDBMS layers until that cache node warms up[7]. On failure of memcached server node storing all the data once again on it could be costly and results in performance degradation.

B. *Limitations Due To Scale Out Approach*

When more storage capacity is required for the cache scale out approach is used instead of scale up that is adding more number of servers then adding more memory to available servers. While using this scale out approach in memcached tier can cause unpredictable application performance degradation due to increasing pressure on the RDBMS layer. If a new node is added to an existing N node memcached tier, around 1/(N+1) th of the keys need to be remapped to different nodes[7]. Additionally, clients need to be updated with this remapping of keys to avoid loss of data or incorrect data delivery, else the application may query another memcached node that does not have the data or has a old version of it. This may force the application to send the query to the RDBMS layer, or return incorrect data to the user.

C. *Lack Of Unified View And Monitoring Problems*

Memcached nodes are independent and unaware of the presence or state of other memcached nodes. Due to this non-grouping approach it is tough to manage some operations, provide some monitoring solution for all the nodes. Also no unified view of all the nodes is not available as the nodes are not behaving as a cluster.

D. *Old Data Access [7]*

There is no provision of up-to-date key-server remapping info. Due to this limitation clients might read or write a key from a wrong memcached server and that will lead to inconsistent data. For example, if there is any network disruption, and one or more clients decide that a particular memcached server is not available anymore, they will automatically rehash some data into the rest of the nodes even if the original one is still available [7]. When the node containing original value eventually returns to service after the network outage is resolved, the data on that node will be stale and the clients without updated key-server remapping info will read stale data.

VII. CONCLUSION

Memcached is an open-source caching technology that is used by top most trafficked websites like facebook, twitter, Livejournal in the world as well as several other high demand online applications. It is primarily used as a performance enhancing mechanism to the database layer and over last ten years it has been widely adopted owing to its two main properties: low latency and ease of programming methodology. However, there are several data storage and operational challenges that still affect memcached users. In this paper all those limitations of memcached are described with an explanation that why they are part of memcached. Future work includes various ways to be found to deal with these problems.

REFERENCES

[1] Petrovic, Jure. "Using Memcached for Data Distribution in Industrial Environment." *ICONS*. 2008.
[2] Wiggins, Alex, and Jimmy Langston. "Enhancing the scalability of memcached." *Intel Software Network* (2012).
[3] Galbraith, Patrick. "*Developing Web Applications With Apache, Mysql, Memcached, and Perl*". John Wiley & Sons, 2009.
[4] Talaga, Paul G., and Steve J. Chapin. "Exploring non-typical memcache architectures for decreased latency and distributed network usage." (2011).
[5] Nishtala, Rajesh, et al. "Scaling memcache at facebook." *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2013.
[6] Fitzpatrick, Brad. "Memcached." *Computer Program, available via http://www. memcached. org* (2010).
[7] Brown, Martin C. " *Developing with Couchbase Server*". O'Reilly Media, Inc., 2013.
[8] Bakar, Khairina Abu, Mohd Hafiz Md Shaharill, and Mohiuddin Ahmed. "Performance evaluation of a clustered memcache." *Information and Communication Technology for the Muslim World (ICT4M), 2010 International Conference on*. IEEE, 2010.