Memcached

RONAK PAREKH[1] AND GREGOR VON LASZEWSKI[2]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.

project-000, March 26, 2017

Memcached is a free and open source, high-performance, distributed memory object caching system. It allows the system to make better use of its memory. It uses data caching technique to speed up dynamic database-driven websites. This reduces the number of times an external data source is accessed by the application. Memcached service is mainly offered through an API. It allows to take memory from parts of the system where you have more memory and allocate it to those parts of the system where you have less memory.

Keywords: Cloud, I524

https://github.com/ronak1182/sp17-i524/tree/master/paper2/S17-IR-2024/report.pdf

INTRODUCTION

Memcached is a high-performance, distributed memory object caching system, generic in nature, but originally intended for use in speeding up dynamic web applications by alleviating database load [1]. Memcached allows you to take memory from parts of your system where you have more than you need and make it accessible to areas where you have less than you need. It is a free and open-source software, licensed under the Revised BSD [2] license. Memcached runs on Unix-like operating systems (at least Linux and OS X) and on Microsoft Windows. It depends on the libevent library.

COMPARISON OF MEMCACHED AND REDIS

- Server: Memcached server is multi-threaded whereas Redis is single threaded. As a result, Redis [3] can't effectively harness multiple cores.
- Distributability: Memcached is very easy to distribute since it doesn't support any rich data-structures. Redis, on the other hand, is much harder to distribute without changing semantics/performance guarantees of some its commands.
- Ease of Installation: Comparing ease of Installation Redis is much easier. No dependencies required.
- Memory Usage: For simple key-value pairs memcached is more memory efficient than Redis. If you use Redis hashes, then Redis is more memory efficient.
- Persistence: If you are using Memcached then data is lost with a restart and rebuilding cache is a costly process. On the other hand, Redis can handle persistent data. By default, Redis syncs data to the disk at least every 2 seconds.
- Replication: Memcached does not supports replication. Whereas Redis supports master-slave replication. It allows slave Redis servers to be exact copies of master servers.Data from any Redis server can replicate to any number of slaves [4].
- Read/Write Speed: Memcached is very good to handle high traffic websites. It can read lots of information at a time and give you back at a great response time. Redis can also handle high traffic on read but also can handle heavy writes as well.
- Key Length: Memcached key length has a maximum of 250 bytes, whereas Redis key length has a maximum of 2GB. When advanced data structures or disk-backed persistence are important, Redis is used.

ARCHITECTURE OF MEMCACHED

Memcached implements a simple and lightweight key-value interface where all key-value tuples are stored in and served from DRAM. The following commands are used by clients to communicate with the Memcached servers:

- SET/ADD/REPLACE(key, value): add a (key, value) object to the cache.
- GET(key): retrieve the value associated with a key.
- DELETE(key): delete a key

Internally, a hash table is used to index the key-value entries. Memcached uses a hash table to index the key-value entries. These entries are also in a linked list sorted by their most recent access time. The least recently used (LRU) entry is evicted and replaced by a newly inserted entry when the cache is full [5].

Hash Table: To lookup keys quickly, the location of each key-value entry is stored in a hash table. Hash collisions are resolved by chaining: if more than one key maps into the same hash table bucket, they form a linked list. Chaining is efficient for inserting or deleting single keys. However, lookup may require scanning the entire chain.

Memory Allocation: Naive memory allocation could result in significant memory fragmentation. To address this problem, Memcached uses slab-based memory allocation. Memory is divided into 1 MB pages, and each page is further sub-divided into fixed-length chunks. Key-value objects are stored in an appropriately sized chunk. The size of a chunk, and thus the number of chunks per page, depends on the particular slab class. For example, by default the chunk size of slab class 1 is 72 bytes and each page of this class has 14563 chunks; while the chunk size of slab class 43 is 1 MB and thus there is only 1 chunk spanning the whole page. To insert a new key, Memcached looks up the slab class whose chunk size best fits this key-value object [5]. If a vacant chunk is available, it is assigned to this item; if the search fails, Memcached will execute cache eviction.

Cache policy In Memcached, each slab class maintains its own objects in an LRU queue (see Figure 1). Each access to an object causes that object to move to the head of the queue. Thus, when Memcached needs to evict an object from the cache, it can find the least recently used object at the tail. The queue is implemented as a doubly-linked list, so each object has two pointers.

Threading: Memcached was originally single-threaded. It uses libevent for asynchronous network I/O callbacks. Later versions support multi-threading but use global locks to protect the core data structures. As a result, operations such as index lookup/update and cache eviction/update are all serialized [5].

WORKING OF MEMCACHED

In the Memcached system, each item comprises a key, an expiration time, optional flags, and raw data. When an item is requested, Memcached checks the expiration time to see if the item is still valid before returning it to the client. The cache can be seamlessly integrated with the application by ensuring that the cache is updated at the same time as the database. By default, Memcached acts as a Least Recently Used cache plus expiration timeouts. If the server runs out of memory, it looks for expired items to replace. If additional memory is needed after replacing all the expired items, Memcached replaces items that have not been requested for a certain length of time (the expiration timeout period or longer), keeping more recently requested information in memory.

Working of Memcached Client [6]: Firstly, a memcached client object is created and it starts calling its method to get and set cache entries. When an object is added to the cache, the Memcached client will take that object, serialize it, and send a byte array to the Memcached server for storage. At that point, the cached object might be garbage collected from the JVM where the application is running. When the cached object is needed, the Memcached client's get() method is called. The client will take the get request, serialize it, and send it to the Memcached server. The Memcached server will use the request to look up the object from the cache. Once it has the object, it will return the byte array back to the Memcache client. The client object will then take the byte array and deserialize it to create the object and return it to the application. Even if the application runs on more than one server, all of them can point to the same Memcached server and use it for getting and setting cache entries. The Memcached client is configured in such a way that it knows all the available Memcached servers.

SECURITY

Memcached is mostly used for deployments within a trusted network. However, sometimes Memcached is deployed in untrusted networks and enviroments where administrators exercise control over the clients that are connected. SASL authentication support is required for Memcached to compile and it requires the binary protocol. For simplicity, all Memcached operations are treated equally. Clients with a valid need for access to low-security entries within the cache gain access to all entries within the cache. If the cache key can be either predicted, guessed or found by exhaustive searching, its cache entry may be retrieved [7].

NEW FEATURES

Chained items were introduced in 1.4.29. With -o modern items sized 512k or higher are created by chaining 512k chunks together. This made increasing the max item size (-I) more efficient in many scenarios as the slab classes no longer have to be stretched to cover the full space. There was still an efficiency hole for items 512k->5mb or so where the overhead is too big for the size of the items. This change fixes it by using chunks from other slab classes in order to "cap" off chained items. With this change larger items should be more efficient than the original slab allocator in all cases.Chunked items are only used with -o modern or explicitly changing -o slab-chunk-max It is not recommended to set slab-chunk-max to be smaller than 256k at this time.

USES OF MEMCACHED

Memcached is used for scaling large websites. Facebook is one of the largest users of memcached [8]. It is used to alleviate database load. Facebook uses more than 800 servers supplying over 28 terabytes of memory to our users. As Facebook's popularity had skyrocketed, they've run into a number of scaling issues. This ever increasing demand required Facebook to make modifications to both our operating system and memcached to achieve the performance that provided the best possible experience for their users. There were thousand of computers, each running hundreds of Apache processes which ultimately led to thousands of TCP connections open to the memcached processes. Memcached uses a per-connection buffer to read and write data out over the network. When hundreds and thousands of connections were in consideration, memory added up to be in gigabytes. Memory that could be better used to store user data had to be considered. Thus, memcached came into consideration while scaling large websites.

CONCLUSION

Memcached is used when scaling large websites is to be done. By alleviating the database load, it helps making the maximum use of its caching technique to find hits before the database is queried. This results in lesser amount of cycles to and from the external data source or database when it comes to speeding up query results. Memcached has its main advantage in distributability when it comes to setting up a distributed cache and gives superior performance when multithreaded processes are in consideration.

REFERENCES

[1] Dormando, “What is memcached?” Web Page, Mar. 2015, accessed 2017-03-21. [Online]. Available: https://memcached.org/

[2] Wikipedia, “Berkeley software distribution,” Web Page, Mar. 2017, accessed 2017-03-22. [Online]. Available: https://en.wikipedia.org/wiki/Berkeley_Software_Distribution

[3] Wikipedia, “Redis,” Web Page, Mar. 2017, accessed 2017-03-22. [Online]. Available: https://en.wikipedia.org/wiki/Redis

[4] Squarespace Inc., “Memcached vs redis,” Web Page, Mar. 2014, accessed 2017-03-25. [Online]. Available: http://www.openldap.org/lists/openldap-software/200010/msg00097.html

[5] S. M., “Memcached vs redis,” Blog, Mar. 2014, accessed: 23-Mar-2017. [Online]. Available: http://blog.andolasoft.com/2014/02/memcached-vs-redis-which-one-to-pick-for-large-web-app.html

[6] Sunil Patil, “Use memcached for java enterprise performance,” Web Page, Mar. 2012, accessed 2017-03-24. [Online]. Available: http://www.javaworld.com/article/2078565/open-source-tools/open-source-tools-use-memcached-for-java-enterprise-performance-part-1-architecture-and-setup.html

[7] Wikipedia, “Memcached,” Web Page, Feb. 2017, online; accessed 23-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Memcached

[8] Facebook, “Scaling memcached at facebook,” Web Page, Mar. 2015, accessed 2017-03-22. [Online]. Available: https://www.facebook.com/note.php?note_id=39391378919&ref=mf