Memcached Working Procedure and Protocols used in Memcache

RAJU B

August 2013

1 Introduction

Memcached was first developed by Brad Fitzpatrick for his website LiveJournal, on May 22, 2003
Wikipedia definition,Memcached is a general-purpose distributed memory caching system that was originally developed by Danga Interactive for LiveJournal, but is now used by many other sites. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read. Memcached runs on Unix, Linux, Windows and Mac OS X and is distributed under a permissive free software license.

Memcached's APIs provide a giant hash table distributed across multiple machines. When the table is full, subsequent inserts cause older data to be purged in least recently used (LRU) order. Applications using Memcached typically layer requests and additions into RAM before falling back on a slower backing store, such as a database.
Memcached is uses in the following sites

1. facebook,youtube,twitter,orange,LiveJournal, Slashdot, Wikipedia and other high traffic sites. etc ...
2. Google App Engine, AppScale, Amazon Web Services also offer a Memcached service through an API

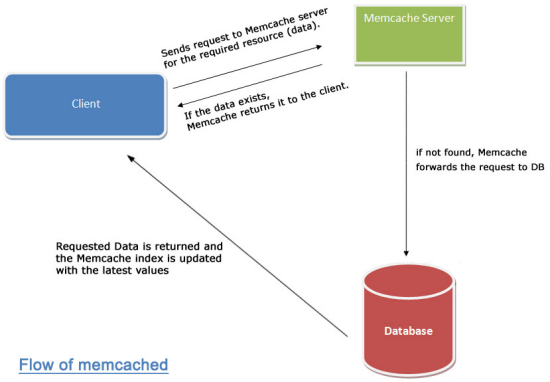
2 How Does Memcached Work

The following figure,it will be familiar to anyone who has ever written a script that interacts with a database. Here is the step-by-step explanation of how it works when you want to pull certain data from the database:

1. Check whether the desired data exists in the cache. If it is in the cache records, then just retrieve the data and hence there is no need to query the database.

2. If the data that you are looking for is not in the cache, then query the database. Return the required data to the script and store the information in the cache.

3. Keep the cache fresh. Whenever the data is changed (i.e. altered or delete for some reason), update this information into the cache. That way, when the cache is queried for the old data then it should either redirect to the database or give the updated information.



WHERE IS MY DATA?

1. The client (not server) uses a hashing algorithm to determine the storage server

2. Data is sent to only one server

3. Servers do not share data

4. Data is not replicated

5. Two hashing algorithms possible:

   (a) Traditional:Traditional hashing uses the forumla:

   ```
   1.hash % num_servers
   2. Resulting number determines the server used
   ```

   (b) "Consistent" Hashing:

      i. each server is allocated LOTS of slots and a key is hashed and to a number.

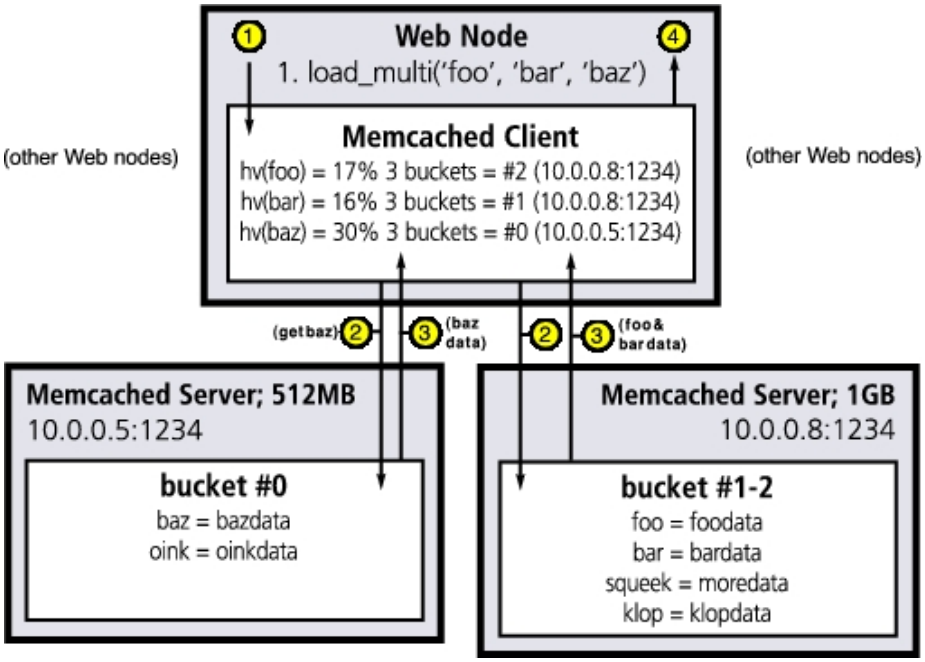
      ii. The closest slot to that number is the server.

   Adding/removing servers from the list results in less key reassignment.

2.1 Working of Memcached with Example

The trick to Memcached is that for a given key, it needs to pick the same Memcached node consistently to handle that key, all while spreading out storage (keys) evenly across all nodes. It wouldn't work to store the key foo on machine 1 and then later have another process try to load foo from machine 2. Fortunately, this isn't a hard problem to solve. We simply can think of all the Memcached nodes on the network as buckets in a hash table.

The following figure shows the working procedure of memcached with example,The Memcached client library is responsible for sending requests to the correct servers.



1. Step 1: the application requests keys foo, bar and baz using the client library, which calculates key hash values, determining which Memcached server should receive requests.
2. Step 2: the Memcached client sends parallel requests to all relevant Memcached servers.
3. Step 3: the Memcached servers send responses to the client library.
4. Step 4: the Memcached client library aggregates responses for the application.

If you know how a hash table works, skim along. If you're new to hashes, here's a quick overview. A hash table is implemented as an array of buckets. Each bucket (array element) contains a list of nodes, with each node containing [key, value]. This list later is searched to find the node containing the right key. Most hashes start small and dynamically resize over time as the lists of the buckets get too long.

3 TCP Protocol

Communicating with a memcached server can be achieved through either the TCP or UDP protocols. When using the TCP protocol, you can use a simple text based interface for the exchange of information.

When communicating with memcached, you can connect to the server using the port configured for the server. You can open a connection with the server without requiring authorization or login. As soon as you have connected, you can start to send commands to the server. When you have finished, you can terminate the connection without sending any specific disconnection command. Clients are encouraged to keep their connections open to decrease latency and improve performance.

Data is sent to the memcached server in two forms:

1. Text lines, which are used to send commands to the server, and receive responses from the server.
2. Unstructured data, which is used to receive or send the value information for a given key. Data is returned to the client in exactly the format it was provided.

```
Both text lines (commands and responses) and unstructured data are always terminated with th
```

Commands to the server are structured according to their operation:

1. Storage commands: set, add, replace, append, prepend, cas
2. Retrieval commands: get, gets
3. Delete commands: delete
4. Increment/Decrement: incr, decr
5. Statistics commands: stats

4 UDP Protocol

For very large installations where the number of clients is high enough that the number of TCP connections causes scaling difficulties, there is also a UDP-based

interface. The UDP interface does not provide guaranteed delivery, so should only be used for operations that aren't required to succeed; typically it is used for "get" requests where a missing or incomplete response can simply be treated as a cache miss.

Each UDP datagram contains a simple frame header, followed by data in the same format as the TCP protocol described above. In the current implementation, requests must be contained in a single UDP datagram, but responses may span several datagrams. (The only common requests that would span multiple datagrams are huge multi-key "get" requests and "set" requests, both of which are more suitable to TCP transport for reliability reasons anyway.)

The frame header is 8 bytes long, as follows (all values are 16-bit integers in network byte order, high byte first):

0-1 Request ID
2-3 Sequence number
4-5 Total number of datagrams in this message
6-7 Reserved for future use; must be 0

The request ID is supplied by the client. Typically it will be a monotonically increasing value starting from a random seed, but the client is free to use whatever request IDs it likes. The server's response will contain the same ID as the incoming request. The client uses the request ID to differentiate between responses to outstanding requests if there are several pending from the same server; any datagrams with an unknown request ID are probably delayed responses to an earlier request and should be discarded.

`The sequence number ranges from 0 to n-1, where n is the total number of datagrams in the message. The client should concatenate the payloads of the datagrams for a given response in sequence number order; the resulting byte stream will contain a complete response in the same format as the TCP protocol (including terminating \r\n sequences).`

you can **secure your cache data** in the following ways if you need to:

Choosing unique keys: Because data stored in Memcached can be taken as a large associative array, you should use unique keys. The only way you can access data is by keys and there is no other way to query Memcached. This limitation can be used to your advantage by choosing unique keys that are hard to guess. You can name the keys using some unpredictable convention.

Securing Memcached server: A server that has Memcached installed and is queried remotely should be behind a firewall. You can define an access-level control in the server, which allows access to only authorized machines.

Encode your data: You can encode your data/key or even both before stor-

ing them in the cache. That involves an extra CPU cycle and is sometimes expensive in the case of larger data, but you still can give it a try and check the performance.

5 References

1. http://www.linuxjournal.com/article/7451
2. http://memcached.org/
3. http://memcachedb.googlecode.com/svn/trunk/doc/protocol.txt
4. `http://www.developer.com/open/article.php/3901666/Getting-Started-with-Memcached-Distri`
5. https://code.google.com/p/memcached/wiki/NewOverview