Memcached

From Wikipedia, the free encyclopedia

Memcached（内存缓存服务）是一个通用的分布式内存缓存系统。它经常用于，通过缓存数据和对象在RAM中来减少外部数据源的读取次数，加快动态数据库驱动网站的访问速度。

Memcached (Mem-Cache-D) is a general-purpose distributed memory caching system. 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 is free and open-source software, licensed under the Revised BSD license.[2] Memcached runs on Unix-like operating systems (at least Linux and OS X) and on Microsoft Windows. It depends on the libevent library.

Memcached's APIs provide a very large 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.[3][4] Applications using Memcached typically layer requests and additions into RAM before falling back on a slower backing store, such as a database.

The size of this hash table is often very large. It is limited to available memory across all the servers in the cluster of servers in a data centre. Where high volume, wide audience web publishing requires it, this may stretch to many gigabytes. Memcached can be equally valuable for situations where either the number of requests for content is high, or the cost of generating a particular piece of content is high.

Memcached was originally developed by Danga Interactive for LiveJournal, but is now used by many other systems, including MocoSpace,[5] YouTube,[6] Reddit,[7] Survata,[8] Zynga,[9] Facebook,[10][11][12] Orange,[13] Twitter,[14] Tumblr[15] and Wikipedia.[16] Engine Yard and Jelastic are using Memcached as part of their platform as a service technology stack[17][18] and Heroku offers several Memcached services[19] as part of their platform as a service. Google App Engine, AppScale, Microsoft Azure and Amazon Web Services also offer a Memcached service through an API.[20][21][22][23]

Memcached

Developer(s)	Danga Interactive
Initial release	May 22, 2003
Stable release	1.4.24 / April 25, 2015; 2 months ago[1]
Written in	C
Operating system	Cross-platform
Type	distributed memory caching system
License	Revised BSD license[2]
Website	www.memcached.org

Contents [hide]

History [edit]

Memcached was first developed by Brad Fitzpatrick for his website LiveJournal, on May 22, 2003.[24][25][26] It was originally written in Perl, then later rewritten in C by Anatoly Vorobey, then employed by LiveJournal.[27]

Software architecture [edit]

This section **does not cite any references or sources**. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. *(June 2013)*

The system uses a client–server architecture. The servers maintain a key–value associative array; the clients populate this array and query it by key. Keys are up to 250 bytes long and values can be at most 1 megabyte in size.

1. 客户端包首先计算出键的散列值，来决定使用哪台服务器

Clients use client-side libraries to contact the servers which, by default, expose their service at port 11211. Each client knows all servers; the servers do not communicate with each other. If a client wishes to set or read the value corresponding to a certain key, the client's library first computes a hash of the key to determine which server to use. Then it contacts that server. This gives a simple form of sharding and scalable shared-nothing architecture across the servers. The server computes a second hash of the key to determine where to store or read the corresponding value.

2. 服务端会计算出键的二次散列值，来决定从哪里存储或读取相关的值

The servers keep the values in RAM; if a server runs out of RAM, it discards the oldest values. Therefore, clients must treat Memcached as a transitory cache; they cannot assume that data stored in Memcached is still there when they need it. Other databases, such as MemcacheDB, Couchbase Server, provide persistent storage while maintaining Memcached protocol compatibility.

If all client libraries use the same hashing algorithm to determine servers, then clients can read each other's cached data.

A typical deployment has several servers and many clients. However, it is possible to use Memcached on a single computer, acting simultaneously as client and server.

Security [edit]

Most deployments of Memcached are within trusted networks where clients may freely connect to any server. However, sometimes Memcached is deployed in untrusted networks or where administrators want to exercise control over the clients that are connecting. For this purpose Memcached can be compiled with optional SASL authentication support. The SASL support requires the binary protocol.

A presentation at BlackHat USA 2010 revealed that a number of large public websites had left Memcached open to inspection, analysis, retrieval, and modification of data.[28]

Even within a trusted organisation, the flat trust model of memcached may have security implications. For efficient 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, even when these are higher-security and that client has no justifiable need for them. If the cache key can be either predicted, guessed or found by exhaustive searching, its cache entry may be retrieved.

Some attempt to isolate setting and reading data may be made in situations such as high volume web publishing. A farm of outward-facing content servers have *read* access to memcached containing published pages or page components, but no write access. Where new content is published (and is not yet in memcached), a request is instead sent to content generation servers that are not publically accessible to create the content unit and add it to memcached. The content server then retries to retrieve it and serve it outwards.

Example code [edit]

Note that all functions described on this page are pseudocode only. Memcached calls and programming languages may vary based on the API used.

Converting database or object creation queries to use Memcached is simple. Typically, when using straight database queries, example code would be as follows:

```
function get_foo(int userid) {
   data = db_select("SELECT * FROM users WHERE userid = ?", userid);
   return data;
}
```

After conversion to Memcached, the same call might look like the following

```
function get_foo(int userid) {
   /* first try the cache */
   data = memcached_fetch("userrow:" + userid);
   if (!data) {
      /* not found : request database */
      data = db_select("SELECT * FROM users WHERE userid = ?", userid);
      /* then store in cache until next get */
      memcached_add("userrow:" + userid, data);
   }
   return data;
}
```

The client would first check whether a Memcached value with the unique key "userrow:userid" exists, where userid is some number. If the result does not exist, it would select from the database as usual, and set the unique key using the Memcached API add function call.

However, if only this API call were modified, the server would end up fetching incorrect data following any database update actions: the Memcached "view" of the data would become out of date. Therefore, in addition to creating an "add" call, an update call would also be needed using the Memcached set function.

```
function update_foo(int userid, string dbUpdateString) {
   /* first update database */
   result = db_execute(dbUpdateString);
   if (result) {
      /* database update successful : fetch data to be stored in cache */
      data = db_select("SELECT * FROM users WHERE userid = ?", userid);
      /* the previous line could also look like data =
createDataFromDBString(dbUpdateString); */
      /* then store in cache until next get */
      memcached_set("userrow:" + userid, data);
   }
}
```

This call would update the currently cached data to match the new data in the database, assuming the database query succeeds. An alternative approach would be to invalidate the cache with the Memcached delete function, so that subsequent fetches result in a cache miss. Similar action would need to be taken when database records were deleted, to maintain either a correct or incomplete cache.

See also [edit]

- Aerospike
- phpFastCache - Supported MemCached, MemCache, WinCache, APC and Files.
- Couchbase Server
- Redis
- Mnesia
- MemcacheDB
- MySQL - directly supports the Memcached API as of version 5.6.[29]
- Oracle Coherence - directly supports the Memcached API as of version 12.1.3.[30]
- GigaSpaces XAP - support Memcached with high availability, transaction support[31]
- Hazelcast
- Cassandra

Computer Science portal

Free software portal

References [edit]

1. ^ "Release notes for Release 1.4.22". Retrieved 2015-04-06.
2. ^ *a b* "Memcached license". GitHub. Retrieved 2014-06-27.
3. ^ "Memcached NewOverview".
4. ^ "Memcached NewUserInternals".
5. ^ MocoSpace Architecture - 3 Billion Mobile Page Views a Month. High Scalability (2010-05-03). Retrieved on 2013-09-18.
6. ^ Cuong Do Cuong (Engineering manager at YouTube/Google) (June 23, 2007). *Seattle Conference on Scalability: YouTube Scalability* (Online Video - 26th minute). Seattle: Google Tech Talks.
7. ^ Steve Huffman on Lessons Learned at Reddit
8. ^ [1]
9. ^ How Zynga Survived FarmVille
10. ^ Facebook Developers Resources
11. ^ Scaling Memcached at Facebook
12. ^ NSDI '13: Scaling Memcache at Facebook
13. ^ Orange Developers
14. ^ It's Not Rocket Science, But It's Our Work
15. ^ Engineer â€“ Core Applications Group job at Tumblr in New York, NY, powered by JobScore. Jobscore.com. Retrieved on 2013-09-18.
16. ^ MediaWiki Memcached
17. ^ Engine Yard Technology Stack
18. ^ Jelastic Memcached System
19. ^ Heroku Memcached add-ons
20. ^ Using Memcache - Google App Engine - Google Code
21. ^ http://appscale.cs.ucsb.edu Memcached in AppScale
22. ^ About In-Role Cache for Windows Azure Cache. Msdn.microsoft.com. Retrieved on 2013-09-18.
23. ^ Amazon ElastiCache. Aws.amazon.com. Retrieved on 2013-09-18.
24. ^ changelog: livejournal. Community.livejournal.com (2003-05-22). Retrieved on 2013-09-18.
25. ^ brad's life - weather, running, distributed cache daemon. Brad.livejournal.com (2003-05-22). Retrieved on 2013-09-18.
26. ^ lj_dev: memcached. Community.livejournal.com (2003-05-27). Retrieved on 2013-09-18.
27. ^ lj_dev: memcached. Lj-dev.livejournal.com (2003-05-27). Retrieved on 2013-09-18.

28. ^ BlackHat Write-up: go-derper and mining memcaches
29. ^ "Speedy MySQL 5.6 takes aim at NoSQL, MariaDB."
30. ^ [2]
31. ^ [3]

External links [edit]

- Official website
- Memcached wiki and faq
- PHP Memcached Manager with Tag Support
- membase
- Memcached and Ruby
- go-memcached - Memcached implementation in Go
- QuickCached - Memcached server implementation in Java
- nsmemcache - memcache client for AOL Server
- Memcached implementation on Windows 8/8.1

Commercially supported distributions [edit]

- Couchbase Server (formerly Membase) offers a Memcached "bucket type" (free for use, subscription support available)
- GigaSpaces Java based Memcached (free community edition, fault tolerance)
- Hazelcast Memcached clustered, elastic, fault-tolerant, Java based Memcached (free for use, subscription support available)

Categories: Free memory management software | Cross-platform software | Structured storage | 2003 software | Database caching