Main page
Get MediaWiki
Get extensions
Tech blog
Contribute

Support
User help
FAQ
Technical manual
Support desk
Communication

Development
Bug tracker
Code repository
Code docs
Statistics
Wikimedia technology
Wikimedia audiences

MediaWiki.org
Community portal
Recent changes
Translate content
Random page
Current issues
Sandbox

Print/export
Create a book
Download as PDF
Printable version

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Cite this page

In other languages
Add links

Manual:Memcached

memcached is a memory-based object store, developed originally to lighten the load on LiveJournal's database servers. memcached has been supported in MediaWiki since pre v1.1.

Memcached is likely more trouble than a small site will need (for a single server, consider storing both code and data in APC - CACHE_ACCEL), but for a larger site with heavy load, like Wikipedia, it should help lighten the load on the database servers by caching data and objects in memory.

Contents [hide]

Requirements [edit]

- PHP must be compiled with --enable-sockets
- libevent
- optionally, epoll-rt patch for Linux kernel
- memcached

Memcached and libevent are under BSD-style licenses.

The server should run on Linux and other Unix-like systems. You can run multiple servers on one machine or on multiple machines on a network; storage can be distributed across multiple servers, and multiple web servers can use the same cache cluster.

Security [edit]

Memcached has no security or authentication. Please ensure that your server is appropriately firewalled, and that the port(s) used for memcached servers are not publicly accessible. Otherwise, anyone on the internet can put data into and read data from your cache.

An attacker familiar with MediaWiki internals could use this to give themselves developer access and delete all data from the wiki's database, as well as getting all users' password hashes and e-mail addresses.

Setup [edit]

If you want to start small, just run one memcached on your web server:

```
memcached -d -l 127.0.0.1 -p 11211 -m 64
```

(to run in daemon mode, accessible only via loopback interface, on port 11211, using up to 64MB of memory)

In your LocalSettings.php file, set:

```
$wgMainCacheType = CACHE_MEMCACHED; $wgParserCacheType =
CACHE_MEMCACHED; # optional $wgMessageCacheType = CACHE_MEMCACHED; #
optional $wgMemCachedServers = array( "127.0.0.1:11211" );
$wgSessionsInObjectCache = true; # optional $wgSessionCacheType =
CACHE_MEMCACHED; # optional
```

The wiki should then use memcached to cache various data. To use multiple servers (physically separate boxes or multiple caches on one machine on a large-memory x86 / Power box), just add more items to the array. To increase the weight of a server (say, because it has twice the memory of the others and you want to spread usage evenly), make its entry a subarray:

```
$wgMemCachedServers = array( "127.0.0.1:11211", # one gig on this box
array("192.168.0.1:11211", 2 ) # two gigs on the other box );
```

PHP client for memcached [edit]

As of this writing (MediaWiki 1.27), MediaWiki uses a fork of Ryan T. Dean's pure-PHP memcached client. It also supports the PECL PHP extension for memcached.

MediaWiki uses three object for object caching:

- $wgMemc, controlled by $wgMainCacheType
- $parserMemc, controlled by $wgParserCacheType
- $messageMemc, controlled by $wgMessageCacheType

If you set CACHE_NONE to one of the three control variable, (default value for $wgMainCacheType), MediaWiki still create a MemCacheClient, but requests to it are no-ops and we always fall through to the database. If the cache daemon can't be contacted, it should also disable itself fairly smoothly.

Troubleshooting [edit]

Loss of session data when saving [edit]

If you store session data in memcached, and users see this message intermittently when they try to save edits:

> Sorry! We could not process your edit due to a loss of session data.
>
> You might have been logged out. **Please verify that you're still logged in and try again.** If it still does not work, try logging out and logging back in, and check that your browser allows cookies from this site.

then one or more of your memcached servers might have a misconfigured `/etc/hosts` file. On each of your memcached servers, make sure the server's own hostname is mapped to localhost:

```
127.0.0.1 servername.here localhost localhost.localdomain ...
```

Otherwise, the server might not be able to connect to its own memcached process.

Using memcached in your code [edit]

If you're writing an extension that does expensive database queries, it might be useful to cache the data in memcached. There are a few main ways to get a handle to memcached:

-
  ```
  $cache = ObjectCache::getMainWANInstance()
  ```

  ...use this if you want a memory-based shared **cache** with explicit purge ability in order to store values derived from persistent sources

-
  ```
  $cache = ObjectCache::getLocalClusterInstance()
  ```

  ...use this if you want a memory-based ephemeral **store** that is not shared among datacenters

-
  ```
  $cache = ObjectCache::getLocalServerInstance()
  ```

  ...use this if you want a memory-based ephemeral **cache** that is not shared among web servers

-
  ```
  $cache = wfGetCache( CACHE_ANYTHING )
  ```

  ...use this if you want any available **cache**, which may or may not be per-datacenter, even an emulated one that uses a SQL database

Note that these may return handles that talk to Redis, APC, MySQL or other stores instead. The use of the word "memcached" is historically due to the API being defined around the simple commands that memcached supports and the fact that, to date, memcached is normally the best general-purpose cache store.

Extensions that have specific needs (like persistence) should define new configuration settings like `$wgMyExtCache` or `$wgMyExtWANCache`. Code using the caches can pass them to `wfGetCache()` and `ObjectCache::getWANInstance()`, respectively.

The following code snippet demonstrates how to cache the results of a database query into memcached for 15 minutes and query memcached first for the results instead of the database.

```
class MyExtensionFooBars { public function getPopularTen() { $cache =
ObjectCache::getMainWANInstance(); return $cache->getWithSetCallback( //
The variadic arguments to wfMemcKey() are used to construct the key for
the cache // in memcached. It must be unique to the query you are
saving. The first value is normally // the extension or component name,
and following values tell you what query you are saving. $cache-
>makeKey( 'myextension', 'foobars', 'popular', '10' ), // Cache for 15
minutes $cache::TTL_MINUTE * 15, // Function to generate the value on
cache miss function ( $oldValue, &$ttl, &$setOpts ) { $dbr = wfGetDB(
DB_SLAVE ); // Adjust TTL based on DB replication lag $setOpts =
Database::getCacheSetOptions( $dbr ); $res = $dbr->select( // your
database query goes here // see Database::select for docs ); $data =
array(); foreach ( $res as $row ) { $data[] = array( // Do something
with the data we // just got from the database // For example, if we
looked up // page_id from the page table, // we could do this: 'id' =>
$row->page_id ); } return $data; } ); } }
```

The abstract BagOStuff and WANObjectCache classes define and document all of the available functions:

- BagOStuff
- WANObjectCache

Old Development Notes [edit]

Broadly speaking, we'd like to be able to dump lots of data in the cache, use it whenever we can, and automatically expire it when changes are made.

Expiration model [edit]

- **explicit expiration times:** memcached lets us set an expiration time on an object when we store it. After the time is up, another request for the object will find that it has expired and return nothing to us.
  - pro: last-ditch fallback to let data that *could* be updated badly eventually fall out of the cache
  - con: we have to know ahead of time when it will cease to be invalid. hard to do when we're dealing with user edits!
- **delete cached objects when we know we're doing something that will cause them to be invalid but are not in a position to update them while we're at it**
  - pro: fairly simple; the item will be reloaded from the database and recached when it's next needed
  - con: if this will affect a large number of related items (for instance, creating or deleting a page invalidates the links/brokenlinks tables and rendered HTML cache of pages that link to that page) we may have to hunt them all down and do a lot of updating
- **include timestamps on cached objects and do our own expiries based on dependencies**
  - pro: can expire many objects at once by updating a single node they depend on
  - con: more things to load; multiple dependencies could be trickier to work with

Questions & Answers [edit]

Q: *The current plan is to deploy six load balanced Apaches, the likeliness that one of them renders the same page twice should be 1/6 of the current value, right?*
A: Memcached is a shared cache between all Apaches, communication is done with TCP.

Q: *Squid will cache the majority of content, reducing repetitions drastically. What's the point in memcached then?*
A: The squid only replaces the anonymous cache. Memcached has far wider applicability, both currently implemented and potentially. -- Tim Starling 02:12, 16 Jan 2004 (UTC)

Q: *Does Memcached have anything to do with your browser or your browser's cache?*
A: NO!

Q: *Can I have multiple clients written in different programming languages access the same Memcached server?*
A: Of course.

Q: *Can I search on part of a key or a regular expression on a Memcached server?*
A: No, you can only search for an exact key if you need more information on what you could possibly do you can check out the Memcached protocol: http://cvs.danga.com/browse.cgi/wcmtools/memcached/doc/protocol.txt (or

https://github.com/memcached/memcached/blob/master/doc/protocol.txt)

Q: *Can I have multiple wikis point to the same Memcached server?*
A: Yes, as long as each have different wiki-ids ($wgDBname). Certain cache keys are intentionally shared in such a scenario, such as rate limiting stuff.

See also [edit]

- docs/memcached.txt in the source documentation
- $wgMainCacheType
- $wgMemCachedServers
- Extension:Memcached
- Manual:Performance tuning
- Memcached on Bluehost
- meta:Tugela Cache
- "Memcached on 1and1 with MediaWiki"

Categories: Pages with ignored display titles | Cache | Performance tuning