Designing and Implementing Scalable Applications with Memcached and MySQL

A MySQL® White Paper

June, 2008

Table of Contents

Introduction

A "cache" duplicates, some or all the data stored on one or more databases for fast and scalable access by applications. Leveraging a caching system is valuable when the original data is expensive to fetch from the source database due to resource constraints. Once the source data is populated in the cache, future requests for the same data make use of the cached copy rather than fetching the data from the original source. Introducing a "caching-layer" typically makes data access times faster and improves the ability for the underlying database to scale in order to accommodate consistently higher loads or spikes in data requests. Often, the cache is "distributed", so that the contents of the cache can be spread out over multiple systems to make more efficient use of available memory, network and other computing resources.

Memached is an open-source, distributed memory caching system designed to tackle today's web-scale performance and scalability challenges. Many of the largest and most heavily trafficked web properties on the Internet like Facebook, Fotolog, YouTube, Mixi.jp, Yahoo, and Wikipedia deploy Memcached and MySQL to satisfy the demands of millions of users and billions of page views every month. By integrating a caching tier into their web-scale architectures, these organizations have improved their application performance while minimizing database load. In doing so, they manage to increase their scalability to accommodate more concurrent users and spikes in traffic while at the same time making the most efficient use of their existing computing infrastructure.

Memcached Overview

Memcached is an actively maintained open source project distributed under the BSD license with regular contributions from not only a community of developers, but also corporate organizations like Facebook and Sun Microsystems. Danga Interactive originally developed memcached to improve the performance and scalability characteristics of the blogging and social networking site LiveJournal. At the time, the site was delivering over 20 million dynamic page views per day to over 1 million users. With LiveJournal's implementation of memcached as a caching-tier, the existing load on the databases was significantly reduced. This meant faster page loads for users, more efficient resource utilization and faster access to the underlying databases in the event data could not be immediately fetched from memcached.

Since memcached was implemented at LiveJournal, it has subsequently been deployed by some of the largest web properties on the Internet. Memcached is most commonly leveraged by organizations to increase the performance of their dynamic, database-driven websites which handle large amounts of read requests by caching not only data and result sets, but also, objects, like pre-compiled HTML.

Memcached is designed to take advantage of free memory on any system running Linux, Open/Solaris, BSD or Windows with very low CPU overhead characteristics. It can be installed on dedicated servers or co-located with web, application or database servers. Out-of-the box memcached is designed to scale from a single server to dozens or even hundreds of servers. Facebook currently has the largest known deployment of memcached servers in production, numbering well over 800.[1] The ability to massively scale out is one of the key advantages of deploying memcached as a caching-tier.

How Memcached Works

A hash is a procedure for turning data into a small integer that serves as an index into an array. The net result is that it speeds up table lookup or data comparison tasks. Memcached leverages a two-stage hash that acts as like a giant hash table looking up key = value pairs. Memcached can be thought of as having two core components, a server and a client. In the course of a memcached lookup, the client hashes the key against a list of servers. When the server is identified, the client sends its request to the server who performs a hash key lookup for the actual data. Because the client performs one stage of the hashing, mamcached naturally lends itself towards the ability to easily add dozens of additional nodes. An added benefit is because there is no interconnect or multicast protocol being employed, the impact to the network is minimized.

Example Memcached Lookup

A basic memcached lookup can be illustrated in the example below with Clients X, Y, Z and Servers A, B, C.

'Set' the Key and Value

- Client X wants to set the key "foo" with value "seattle"
- Client X takes the list of available memcached servers (A,B,C) and hashes the key against them
- Server B is selected
- Client X directly connects to Server B, and sets key "foo" with the value "seattle"

'Get' the Key and Value

- Client Z wants to get key "foo"
- Client Z is able to use the same hashing process to determine that key "foo" is on Server B
- Client Z directly requests key "foo" from Server B and gets back "seattle"
- Subsequent requests from Clients X, Y or Z for key "foo" will access the data directly from Server B

[1] http://highscalability.com/strategy-break-memcache-dog-pile

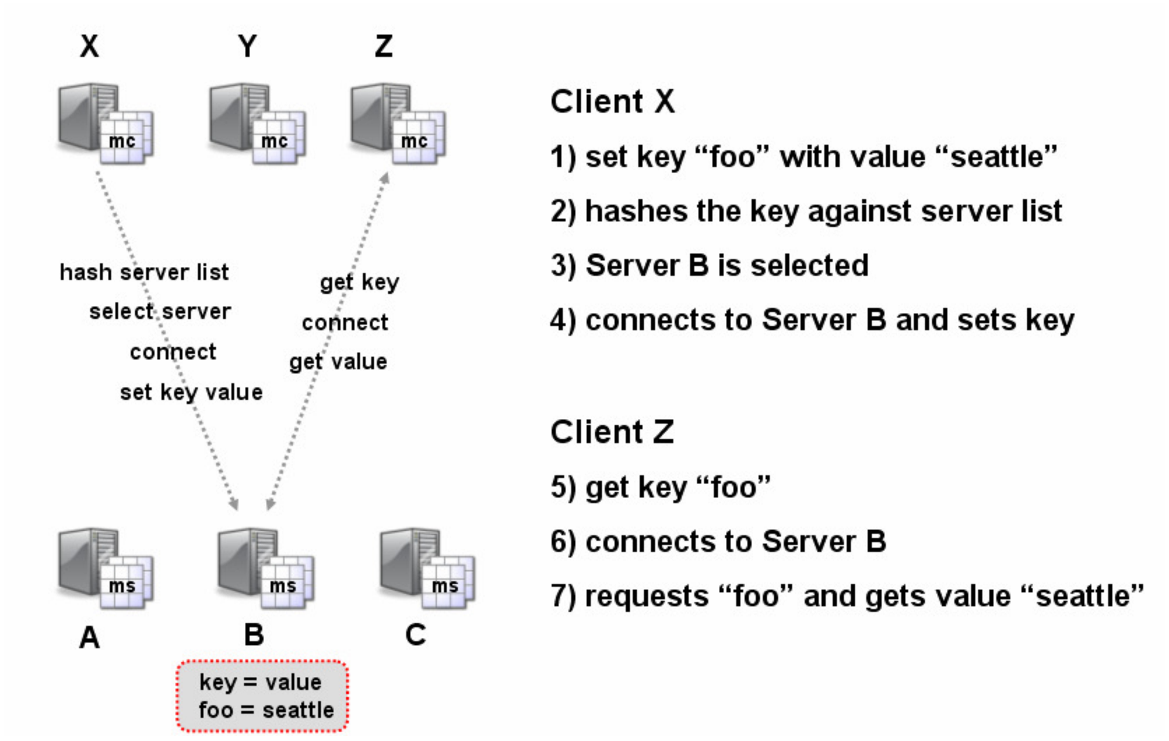

Figure 1: Basic Memcached Lookup

Memcached Server

As previously mentioned, there are two core components to memcached, the server and the client. In this section we cover some of the core capabilities of the memcached server, including how the memcached server deals with memory allocation, the caching of data. Overall, the memached server is implemented as a non-blocking event-based server with an emphasis on scalability and low resource consumption.

Memory Allocation

By default, the memcached server allocates memory by leveraging what is internally referred to as a "slab allocator". The reason why this internal slab allocator is used over malloc/free (a standard C/C++ library for performing dynamic memory allocation) is to avoid fragmentation and the operating system having to spend cycles searching for contiguous blocks of memory. These tasks overall, tend to consume more resources than the memcached process itself. With the slab allocator, memory is allocated in chunks and in turn, is constantly being reused. Because memory is allocated into different sized slabs, it does open up the potential to waste memory if the data being cached does not fit perfectly into the slab.

There are also some practical limits to be aware of concerning key and data size limits. For example, keys are restricted to 250 characters and cached data cannot exceed the largest slab size currently supported, 1 megabyte.

The hashing algorithm that memcached makes use of to determine which server a key is cached on does not take into account the size of all the memory available across all the servers participating in a memcached cluster. For those concerned about this issue, a potential workaround is to run multiple memcached instances on servers with more memory with each instance using the same size cache as all the other servers.

Caching Structure

When memcached's distributed hash table becomes full, subsequent inserts force older cached data to be cycled out in a least recently used (LRU) order with associated expiration timeouts. How long data is "valid" within the cache is set via configuration options. This "validity" time may be a short, long or permanent. As mentioned, when the memcached server exhausts its memory allocation, the expired slabs are cycled out with the next oldest, unused slabs queued up next for expiration.

Memcached makes use of lazy expiration. This means it does not make use of additional CPU cycles to expire items. When data is requested via a 'get' request, memcached references the expiration time to confirm if the data is valid before returning it to the client requesting the data. When new data is being added to the cache via a 'set', and memcached is unable to allocate an additional slab, expired data will be cycled out prior to any data that qualifies for the LRU criteria.

Caching Database Queries

One of the most popular use cases for leveraging memcached is to cache the results of database queries. Below is a simple outline of how the 'add' and 'set' memcached functions can be leveraged in this respect.

- The server first checks whether a memcached value with a unique key exists, for example "user:userid", where userid is a number.

  "SELECT * FROM users WHERE userid = ?"

- If the result is not cached, the request will issue a select on the database, and set the unique key using the memcached 'add' function call.

- If this call was the only one being altered the server would eventually fetch incorrect data, so in addition to using the 'add' function, an update is also required, using the 'set' function.

- This 'set' function call updates the currently cached data so that it is synchronized with the new data in the database. (Another method for achieving a similar behavior is to invalidate the cache using the 'delete' function so that additional fetches result in a cache miss forcing an update to the data.)

At this point, whenever the database is updated, the cache also needs to be updated in order to maintain the desired degree of consistency between the cache and the source database. This can be achieved by tagging the cached data with a very low expiration time. However, this does mean that there will be a delay between the update occurring on the database and the expiration time on the cached data being reached. Once the expiration time is reached, a subsequent request for the data will force an update to the cache.

You may discover during testing or benchmarking that memcached may not yield faster performance than simply running queries on the source database. The perspective to keep in mind is that memcached is designed to assist in scaling the database, so as connections and requests to the source database ramp up, memcached will assist in alleviating load by handling read requests.

Data Redundancy and Fail Over

By design, there are no data redundancy features built into memcached. Memcached is designed to be a scalable and high performance caching-layer, including data redundancy functionality would only add complexity and overhead to the system.

In the event one of the memcached servers does suffer a loss of data, under normal circumstances it should still be able to retrieve its data from the original source database. A prudent caching design involves ensuring that your application can continue to function without the availability of one or more memcached nodes. Some precautions to take in order not to suddenly overwhelm the database(s) in the event of memcached failures is to add additional memcached nodes to minimize the impact of any individual node failure. Another option is to leverage a "hot backup", in other words a server that can take over the IP address of the failed memcached server.

Also, by design, memcached does not have any built in fail over capabilities. However, there are some strategies one can employ to help minimize the impact of failed memcached nodes.

The first technique involves simply having an (over) abundance of nodes. Because memcached is designed to scale straight out-of-the-box, this is a key characteristic to exploit. Having plenty of memcached nodes minimizes the overall impact an outage of one or more nodes will have on the system as a whole.

One can also remove failed nodes from the server list against which the memcached clients hash against. A practical consideration here is that when clients add or remove servers from the server list, they will invalidate the entire cache. The likely effect being that the majority of the keys will in-turn hash to different servers. Essentially, this will force all the data to be re-keyed into memcached from the source database(s). As mentioned previously, leveraging a "hot backup" server which can take over the IP address of the failed memcached server can go a long way to minimize the impact of having to reload an invalidated cache.

Loading and Unloading Memcached

In general, "warming-up" memcached from a database dump is not the best course of action to take. Several issues arise with this method. First, changes to data that have occurred between the data dump and load will not be accounted for. Second, there must be some strategy for dealing with data that may have expired prior to the dump being loaded.

In situations where there are large amounts of fairly static or permanent data to be cached, using a data dump/load can be useful for warming up the cache quickly.

Memcached Threads

With the release of version 1.2, memcached can now utilize multiple CPUs and share the cached between the CPUs engaged in processing. At its core is a simple locking mechanism that is used when particular values or data needs to be updated. Memcached threads presents considerable value to those making use of 'multi-gets', which in-turn become more efficient and makes memcached overall easier to manage, as it avoids having to run multiple nodes on the same server to achieve the desired level of performance.

Memcached Clients

In a typical memcached cluster, the application will access one or more memcached servers via a memcached client library. Memcached currently has over a dozen client libraries available including, Perl, PHP, Java, C#, C/C++, Lua and a native MySQL API. A complete list of client APIs can be found at:

http://www.danga.com/memcached/apis.bml

Supported Client Libraries

It is important to note that there are different memcached client implementations. Not only does the manner in which they store data into memcached vary between client libraries, but also in how they implement the hashing algorithm. Because of this, the implications of mixing and matching client libraries and versions should be carefully considered. However, unlike the variances found between client versions, memcached servers always store data and apply hashing algorithms consistently.

From a security standpoint, it does bear mentioning that memcached does not posses any authentication or security features. Best practices in this area would dictate that memcached should only be run on systems within a firewall. By default, memcached makes use of port 11211.

PHP PECL

For information concerning the PHP extensions which allows for Object-Oriented and procedural interfaces to work with memcached, please see:

http://pecl.php.net/package/memcache

Java Client Libraries

Two Java APIs for memcached are currently in development:

Memcached Client for Java

http://www.whalin.com/memcached/

Spymemcached – Java Client for Memcached

http://code.google.com/p/spymemcached/

Python Client Library

A 100% Python interface to memcached. For more information, please see:

ftp://ftp.tummy.com/pub/python-memcached/

Perl Client Library

A Perl client library for memcached. For more information, please see:

http://search.cpan.org/dist/Cache-Memcached/

C Client Library

There are currently three C libraries for memcached. For more information, please see:

apr_memcache

http://www.outoforder.cc/projects/libs/apr_memcache/

libmemcached

http://tangent.org/552/libmemcached.html

libmemcache

http://people.freebsd.org/~seanc/libmemcache/

MySQL API

The memcache_engine allows memcached to work as a storage engine to MySQL. This allows SELECT/UPDATE/INSERTE/DELETE to be performed from it were a table in MySQL.

http://tangent.org/index.pl?node_id=506

Also, there is a set of MySQL UDFs (user defined functions) to work with memcached using libmemcached.

http://tangent.org/586/Memcached_Functions_for_MySQL.html

Example Architectures

There are various ways to design scalable architectures using memcached and MySQL. Below we illustrate several of these architectures.

Memcached & MySQL

In this architecture we combine several memcached servers and a stand-alone MySQL server. This architecture allows for scaling a read intensive application. The memcached clients are illustrated separately; however, they typically will be co-located with the application server.

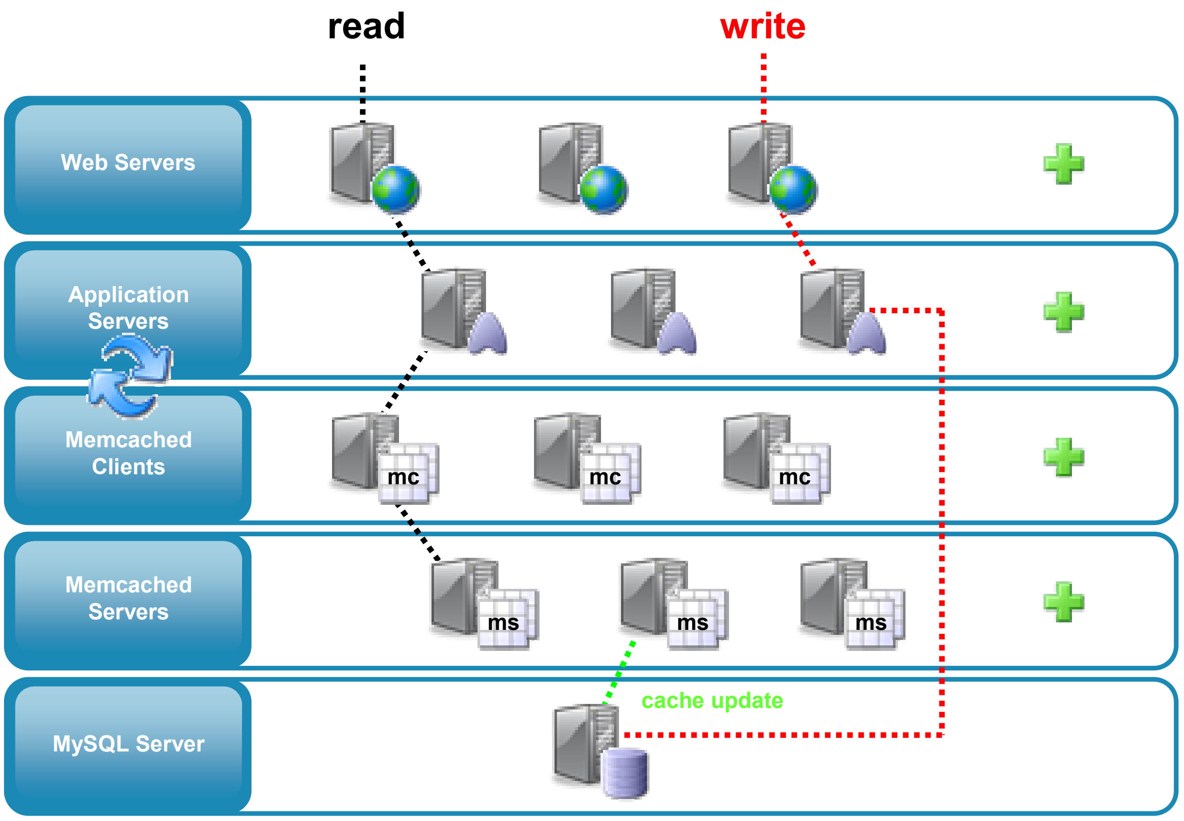

Figure 2: Multiple Memcached Servers and a Stand-Alone MySQL Server

Scale-Out: Memcached & MySQL Replication

In this architecture we combine several memcached servers with a master MySQL server and multiple slave servers. This architecture allows for scaling a read intensive application. The memcached clients illustrated are co-located with the application servers. Read requests can be satisfied from memcached servers or from MySQL slave servers. Writes are directed to the MySQL master server.

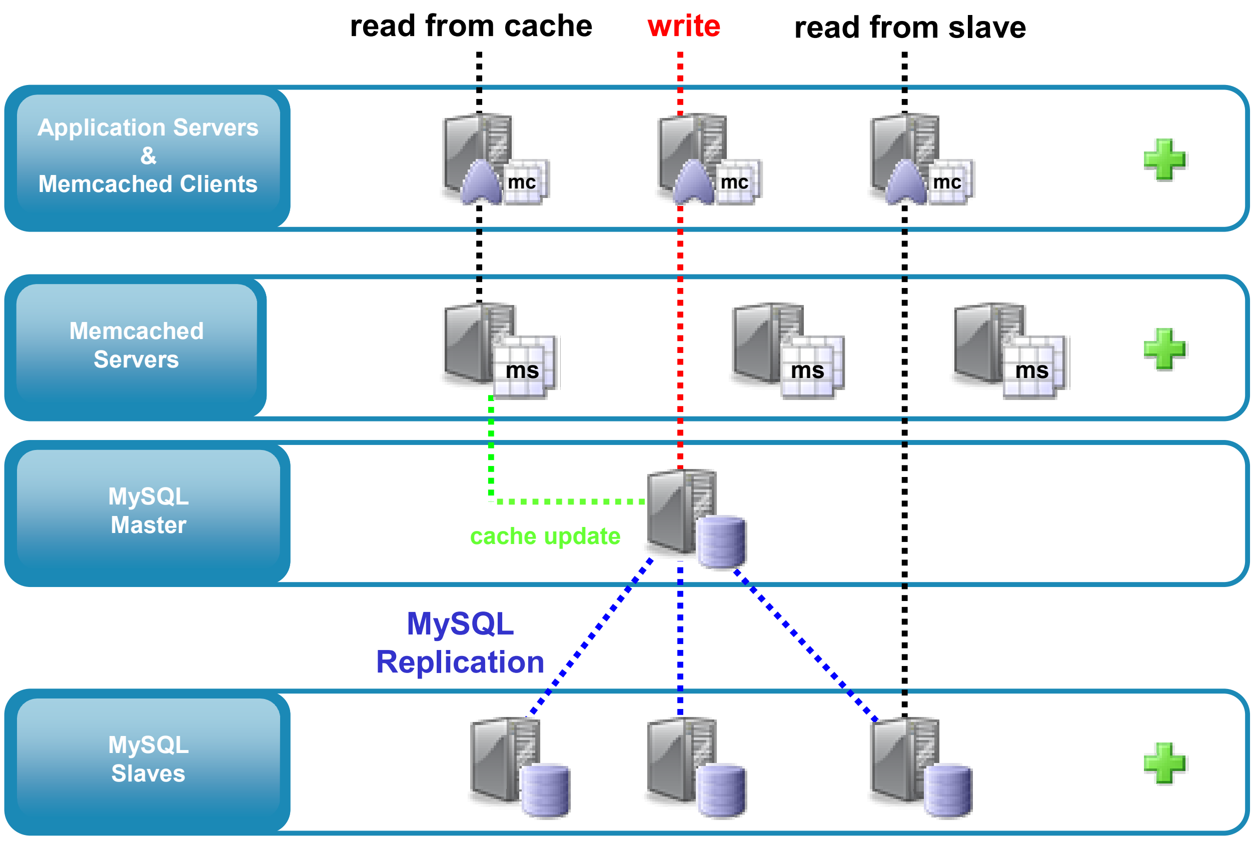

Figure 3: Multiple Memcached Servers with a Master and multiple Slave MySQL Servers

Memcached, Sharding & MySQL Replication

With sharding (application partitioning) we partition data across multiple physical servers to gain read and write scalability. In this example we have created two shards partitioned by customer number. Read requests can be satisfied from memcached servers or from MySQL slave servers. Writes are directed to the appropriate MySQL master server.

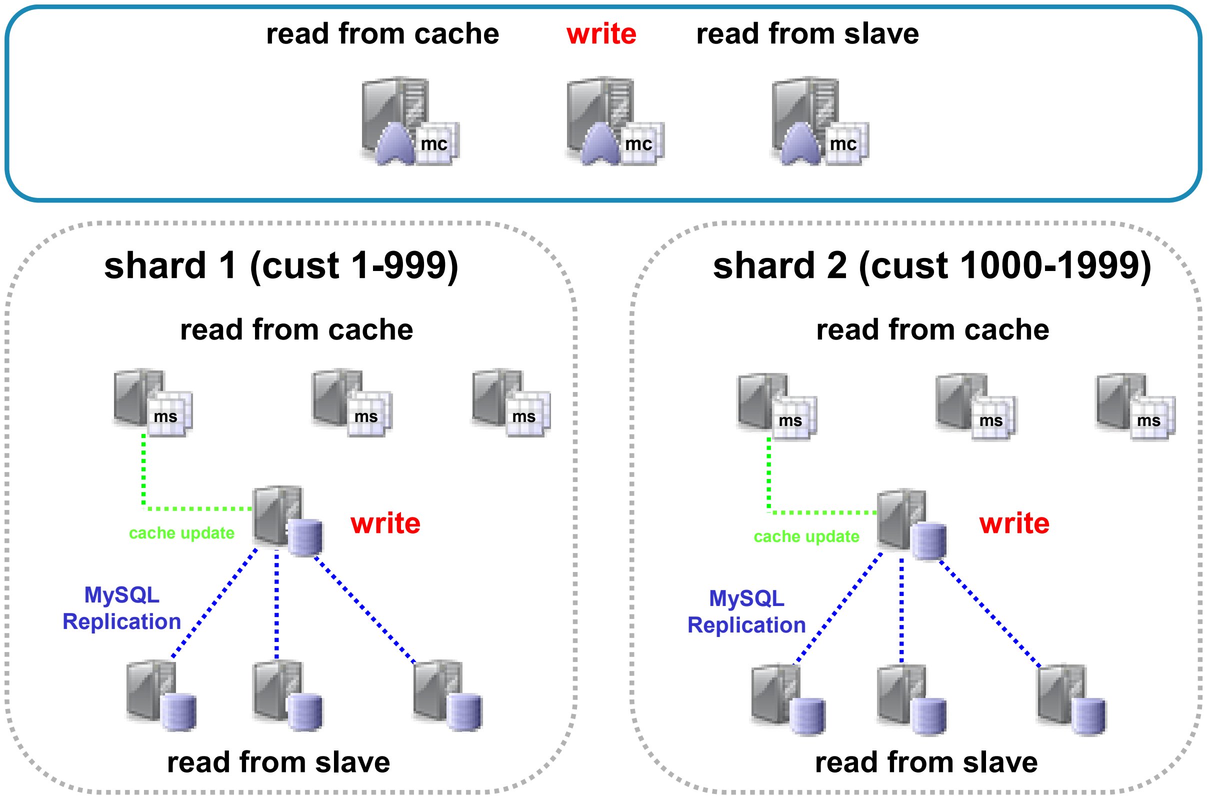

Figure 4: Sharding, multiple Memcached Servers with a Master and multiple Slave MySQL Servers

Solutions: Memcached for MySQL

The MySQL Enterprise subscription now includes 24x7 production support for the memcached distributed memory caching system. Support for memcached is available at the Gold and Platinum subscription levels.

Memcached for MySQL enables organizations to:

- Implement a scalable, high performance data caching solution for their online applications
- Reduce database Total Cost of Ownership (TCO) by eliminating licensing costs for proprietary data caching software
- Reduce system TCO by making better use of resources like idle or spare RAM on existing systems
- Incrementally add/remove data caching capacity, on-demand to quickly meet changing requirements.

For more information about MySQL Enterprise, please visit:

http://www.mysql.com/products/enterprise/features.html

High Performance Data Caching

Memcached enables organizations to increase the scalability of new and existing MySQL applications. Memcached combined with MySQL Replication, is an excellent solution for improving application performance and leveraging scale out architectures at the same time.

On-Demand Scalability

Memcached can be incrementally scaled out in an on-demand fashion. Because Memcached can scale to support dozens of nodes with minimal overhead, anywhere spare memory resources exist is an opportunity to scale your application even further. Memcached is designed as a non-blocking event-based server with no special networking or interconnect requirements.

Lower TCO with Open Source

Delivering a scalable, high performance data caching solution based on open source components is a cost-effective alternative to expensive proprietary software. Memcached is a great way to protect yourself from proprietary vendor lock, which can result in unpredictable costs as the size of your application grows and the number of users increases exponentially. MySQL and Memcached can be implemented with confidence as they both benefit from a large community of developers and users ensuring they stand up even under the most extreme production loads. Further, memcached gives you the benefit of caching more then just result sets. For example, preprocessed HTML can be cached on the same Memcached nodes, further improving the performance of your web application.

24x7 Support from MySQL

Organizations can get the professional help they need from the Memcached and MySQL experts at Sun Microsystems

Professional Consulting Services

With extensive memcached and MySQL knowledge plus the hands-on experience to implement scalable, high performance solutions for MySQL customers, our consultants use proven methodologies and expertise in Database Design, Architecture, Performance Tuning, Replication, Fail-Over and Fault-Tolerance.

For more information about consulting services, please visit:

http://www.mysql.com/consulting/

Conclusion

In this whitepaper we have provided a basic introduction to memcached and how it can improve application performance while minimizing database load. Memcached increases the scalability of dynamic, data-driven applications by increasing the possibility of supporting more concurrent users and spikes in traffic, while at the same time making the most efficient use of existing computing infrastructure. We explored how the memcached server allocates memory, implements hashing and interacts with memcached clients. We have also illustrated several scalable architectures which combine memcached, MySQL and Replication. Finally, we presented Memcached for MySQL which is a standard part of a MySQL Enterprise Gold or Platinum subscription. Not only does a MySQL Enterprise subscription get you access to 24x7 production support from MySQL and memcached experts from Sun Microsystems, it also gives you access to the MySQL Enterprise Server and MySQL Enterprise Monitor software.

Additional Resources

White Papers

http://www.mysql.com/why-mysql/white-papers/

Case Studies

http://www.mysql.com/why-mysql/case-studies/

Press Releases, News and Events

http://www.mysql.com/news-and-events/

Live Webinars

http://www.mysql.com/news-and-events/web-seminars/

Webinars on Demand

http://www.mysql.com/news-and-events/on-demand-webinars/

Download Memcached

http://www.danga.com/memcached/download.bml

Installation Tutorial

http://blog.ajohnstone.com/archives/installing-memcached/

Memcached for MySQL Reference Manual

http://dev.mysql.com/doc/refman/5.1/en/ha-memcached.html

Memcached FAQ

http://www.socialtext.net/memcached/index.cgi?faq

Memcached for MySQL Forums

http://forums.mysql.com/list.php?150

About Sun’s MySQL Portfolio

The MySQL product portfolio is the most popular open source database software in the world. Many of the world's largest and fastest-growing organizations use MySQL to save time and money powering their high-volume Web sites, critical business systems and packaged software -- including industry leaders such as Yahoo!, Alcatel-Lucent, Google, Nokia, YouTube and Zappos.com. At http://www.mysql.com, Sun provides corporate users with commercial subscriptions and services, and actively supports the large MySQL open source developer community.

To discover how Sun’s offerings can help you harness the power of next-generation Web capabilities, please visit http://www.sun.com/web .