

Reliable and fault tolerant distributed caching using memcached

Bivas Das and Eric Freudenthal
Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79968

Email: bdas@miners.utep.edu and efreudenthal@utep.edu

March 20, 2010

Abstract

Enormous network of computers, supporting various services in the public domain i.e. the Internet or the World Wide Web, or in the private networks satisfying privileged audiences, require various intelligent techniques to enable high availability and reliability of the resources. Most of these services rely upon and serve as a query-response system, where they respond to or query appropriate host for a certain solution to a problem. For example, database server always listens on a designated port for structured query and responses them from the information it contains within. Since communication plays a major role, most of these intelligent techniques range from hardware level design to minimize overhead or latency in communication to software level optimization that reduces communication cost. Caching is one of the most used optimization techniques to reduce communication time in computing. Caching is also used in various forms in systems, with the same objective to minimize communication time.

In a service oriented architecture, consisting of many interdependent systems, caching plays a major role towards performance. Consider a web-server serving http response by querying a database for supporting information. The service running at the database end can cache frequently used for faster response. This way the server responses faster achieving smoother response and better usability in the user end.

In more complex system where the load is typically very high and acceptable response time to attain usability, a more generic caching mechanism may be used. As designed by Fitzpatrick [1], such a system, memcached can be used as a generic caching system that can cache virtually every kind of data as a stream and can respond very fast as it uses the system's idle memory to store the data, instead of the disk. Memcached is widely used in many online services such as Wikipedia, YouTube, Craigslist, Digg, Flickr, Twitter and many others. All these services have one major attribute in common, high user activity and high availability, meaning these systems handle thousands of requests per second and they try to attain that 24/7 with no or minimum downtime.

While trying out memcached, one major issue we noticed was, in spite of integrated in distributed network of systems the memcached server itself does not act as a distributed system itself. The system works as a stand-alone caching abstraction that can only reply with appropriate data if it is available in the cache in very low response time. This reduces the scalability of memcached by moving all the caching management part to the user end where memcached servers remain unaware of its neighbors. Furthermore, this particular approach increases risk of system availability in case of failure. While testing in a test environment using 10 memcached server and 1 client, the network performed as expected, but it failed drastically upon removal of a single node from the pool of systems. Thus we propose a high performance distributed caching system that will should try to heal itself when a system failure is detected.

As discussed, currently the memcached server works stand-alone and is completely unaware of its surrounding neighbors. The current model forces the clients to implement an api to map the data to the server. Therefore, the system in concern, will have the following properties:

- a. Self-healing and re-arrangement of the pool in case of a system failure
- b. Automatic re-distribution of the cached data upon changes in pool structure

This way the system will not only provide fast cached response but also will provide high availability even in times of system failure.

References

- [1] Brad Fitzpatrick. Distributed caching with memcached. linux journal, *Linux Journal*, 2004(124):5, August 2004. url:<http://www.linuxjournal.com/article/7451>.