Summer 5-25-2013

Design of a Multi-Host Shared Memory Services System

Chung-Yang Chen
Department of Information Management, National Central University, No. 300, Jhongda Rd., Jhongli City, Taoyuan Country 32001, Taiwan, 974403007@cc.ncu.edu.tw

Wen-Lung Tsai
Department of Information Management, National Central University, No. 300, Jhongda Rd., Jhongli City, Taoyuan Country 32001, Taiwan, tswelu@gmai.com

Design of a Multi-Host Shared Memory Services System

[1]*Chung-Yang, Chen,* [2]*Wen-Lung, Tsai*
1, First Author Department of Information Management, National Central University,
No. 300, Jhongda Rd., Jhongli City, Taoyuan Country 32001, Taiwan
974403007@cc.ncu.edu.tw
*2,Corresponding Author Department of Information Management, National Central University,
No. 300, Jhongda Rd., Jhongli City, Taoyuan Country 32001, Taiwan
tswelu@gmai.com

Abstract: Memory cache is one kind of memory, through which data and objects are stored, thereby reducing the time required to access the database and hard disk I/O, and achieving accelerated technology effects by a significant application in large-scale web systems. In this paper, we design Memcahed Helper (MH), based on a set of memcached with the scalability of a distributed memory cache system, in line with the progress of the cloud environment. The experimental results show that this system and the more efficient use of memory, provides better performance and speed.

Keywords: Distributed memory cache system, Memcached Helper, Database

1. Introduction

Large websites are usually the object or data stored in parallel databases [1]. However, as the number of users increases, the performance of the database system cannot meet the demand and service performance bottlenecks. In order to solve this problem, the site began to increase the effectiveness of services by using memory cache object stores. Figure 1 is a common use of the memory cache. In addition to the original database systems, additional settings memory cache servers. AP servers simultaneously write to the cache servers and database servers, and data is read from the cache servers. Only when the required information is in the cache servers, do the AP servers obtain information from the database servers, and the results are written to the cache servers in order to increase the hit rate.

Memcached [12], under the BSD license by Danga Interactive, developed memory cache servers. Many websites, such as Yahoo and Facebook, use memcached as their memory cache servers [12]. The design concept in memcached is a client-server architecture that communicates with AP servers by simple text. In memcached, each object is a key-value composition. A key of the object is an index for searching an object.

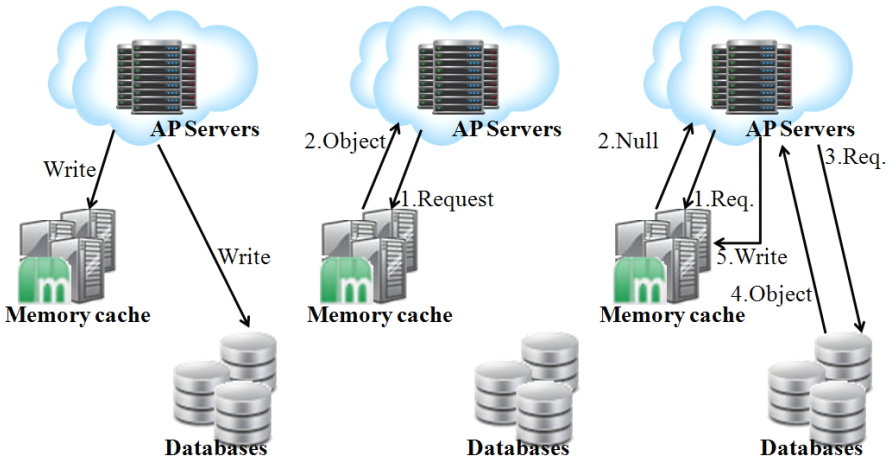

Figure 1: Memory cache diagram

Memcached is not simple, but it is fast. For the sake of simplicity, to show the performance difference with the general database systems, we simultaneously stored 16,000 objects in memcached and PostgreSQL, and compared the time spent. The results are shown in Figure 2, and it is clear that memcached performed better.

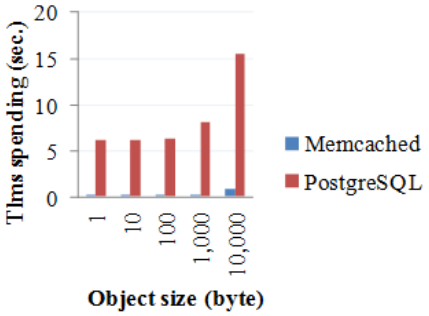

Figure 2. Comparison between memcached and PostgreSQL

The memcached protocol itself does not specify the way data is distributed. There is no specific program library or applications, so users can modify the demands when writing AP. In addition, there are many different languages, so different system programming can be used directly, and the way data distribution is used is not the same.

This paper will discuss memcached, the clients of memcached, and how it can be improved. We believe the optimal system architecture of memcached should have three specific items. First, the data should be distributed evenly, that is, clients should be able to decide in accordance with the server load and memory capability where to store data. A good hash function can be passive, preventing the uneven distribution of the workload, but it cannot take the initiative to adjust the loading system imbalance. Furthermore, clients should not spend too much time in determining how to distribute data, nor the host data store, as data distribution should be as simple as possible. Finally, server changes should not affect the clients, i.e. when servers join or leave the service group it should have minimal impact on the clients. This paper proposes a new architecture and protocol in order to achieve the above characteristics, using simple data distribution in a dynamic way to balance the system, and leaving a minimum impact on clients and servers that join and leave, with only a slight increase in the use of resources.

2. Literature review

2.1. Memcached

In this section, we refer to the open source of memcached and Petrovici's research [11], memcached internal mechanism with real explore.

Simple text protocol is used to communicate between memcached with clients, rather than the complicated format XML or JSON. This allows for direct connect using telnet programs, such as the previous section example. In addition, in order to increase performance, memcached also provides the binary mode protocol conversion from text protocol. Memcached itself has no security mechanism, so ideally it should be set in the firewall afterwards.

Memcached adopted the slab allocation to improve performance. The so-called slab allocation, refers to a pre-configured chunk of memory, and is divided into many different small-sized pieces of memory. When the program needs to use memory, it chooses the size then runs out, only to cancel the memory of the connection rather than its release. Figure 3 explains the operation of this mechanism. This mechanism can reduce the number of call system functions, and memory for a large number of operations due to memcached, as memcached enhances the performance impact of this mechanism, which is particularly significant.

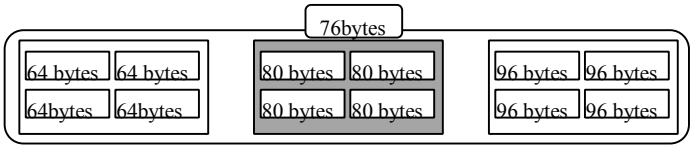

Figure 3. Slab allocation

2.2. Static hashing distributes objects

Static hashing is a hash table to increase or decrease the space mapping relationship with only a slight change, rather than the general hash table that requires re-mapping all the way. Most memcached practice is as follows: start with the first known server in the server list, then specify a location in each server code, and finally write and read using static hashing [2].

Figure 4 is a simple static hashing example, with five servers, and the location code for p1 to p5 object 1 saves to s5, because of the position after its hash before s5; saving s1 object 2 follows the same principle.

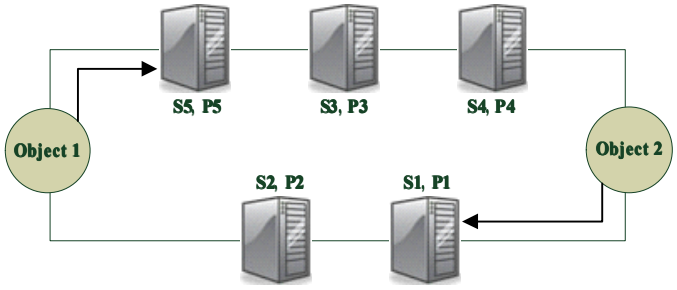

Figure 4. Static hashing diagram

2.3. Peer-to-peer protocol

Some peer-to-peer protocols, such as Chord [9], can organize and search the functionality of the network. The Chord itself is designed for large networks, to provide for the centralized functions, and can handle the new node join, leave, and recovery. Figure 5 shows how to deal with these situations in the Chord.

The Chord in peer-to-peer networks is very effective, but is not suitable for the cache system. All actions, such as JOIN, LEAVE, and RECOVERY after synchronization. In distributed cache systems, system delay may cause the redundant cache to fail and affect performance. In addition, the Chord does not provide a mechanism for the mobile node.

```
// create a new Chord ring.
 n.create()
   predecessor = nil;
   successor = n;
// join a Chord ring containing node n'.
 n.join(n')
   predecessor = nil;
   successor = n'.find_successor(n);
// called periodically. n asks the successor
 // about its predecessor, verifies if n's immediate
 // successor is consistent, and tells the successor about n
 n.stabilize()
   x = successor.predecessor;
   if (x∈(n, successor))
     successor = x;
   successor.notify(n);
// n' thinks it might be our predecessor.
 n.notify(n')
   if (predecessor is nil or n'∈(predecessor, n))
     predecessor = n';
```

Figure 5. Chord protocol

3. Design method and evaluation

3.1. System request analysis

1. Balance

Object distribution on networks generally complied with Zipf's Law [3]; especially in the cache system [4], it is more likely to differ from the normal distribution. The traditional static hashing normal distribution may be a higher than average points system, but this is not always the case. In addition, in the absence of a dynamic adjustment mechanism, the problem of uneven distribution will grow with time, becomes more serious, resources are wasted, and performance decreases.

Single server's memory capabilities are limited, and when the memory is fully loaded and then stored in the new object, one of two conditions will occur. First, as it ignores the increasing requirements of the new object, another is to delete the old object. The two situations reduce the cache hit rate; clients should take the object storage to other servers while there is still room to avoid this situation. In addition, as CPU load is high, memcached can effectivey and rapidly decrease load. Therefore, the client need not often to request the server in high load in order to maintain overall performance.

2. Scalability

The easiest way to get the server to cluster is to assume that the information of clients is known by all servers. However, when the system load is high or memory is exhausted, new servers can be added to share the work. For now, memcached and the clients do not provide this feature.

3. Fault tolerance

Servers are faulty. In order to reduce the impact of a faulty server, in a service cluster, faulty servers are automatically removed from the network, in order to avoid clients visiting failed servers.

3.2. System design

Figure 6 in this paper shows an overview of the structure mentioned. Memcached helper (MH) is required in memcached servers to perform their work. The MH protocol was implemented to integrate server networks.

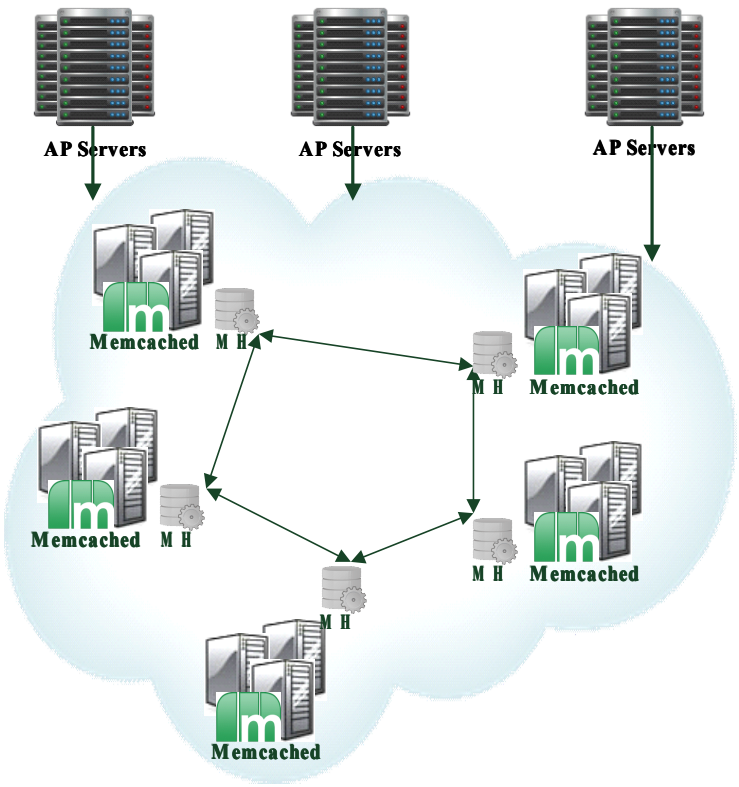

Figure 6. MH system overview

Each MH memcached protocol is tailored to a corresponding memcached server's communication, and the MH protocol in all memcached servers integrates them into the network. Clients use memcached protocal to operate memcached server in order to get the synchronous information. In addition, clients are responsible for data distribution. Figure 7 shows MH and its client architecture; we will first introduce MH, and then introduce clients.

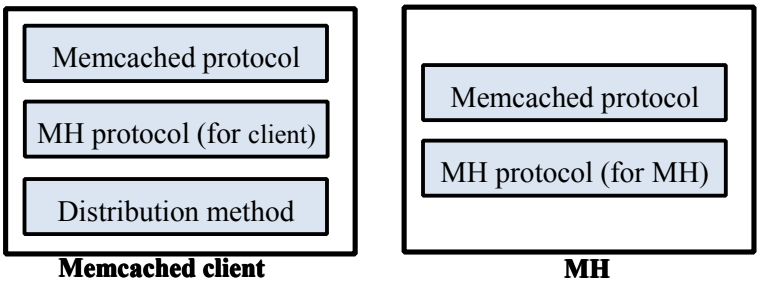

Figure 7. MH & Clients architecture

MH adopts a Chord-like manner, maintaining its fore-and-aft server information. In addition, it also contains the client list and recovery information. The client list is for the organization's server network, which is used for synchronization and error recovery. Both can be updated through events triggered by messages issued by other MH or clients.

1. MH JOIN

When MH tries to join the server network, it will send out a MH JOIN message, which contains hash out, the results of which are in accordance with its own IP and node n. If n is empty, it becomes the first node in the network; otherwise, node n is checked and joins the servers in their own front or rear, and updates its information. The direction of message transmission is determined by the message recorded in the status, and there are three directions possible: Forward, Backward, and Native. When the message is created, its status is Native, and then becomes Forward or Backward, depending on the location of the first node encountered. MH receives the Forward message, then the message is sent to the next node, and finally the message goes Backward when sent to the last node. The termination condition is very simple. First, the message to Forward to the new node becomes the next node, and another message for a new Backward node becomes the last node. On the other hand, when MH receives a Native MH JOIN status message, the message will be passed on to synchronize all nodes.

2. CLIENT JOIN

MH must know the clients, and be in sync when it pushes information to the clients. When MH receives a CLIENT JOIN message, it adds the clients own list of clients, and information is sent to the clients to join. Then the message continues to pass to the following nodes until it finishes a lap.

3. MH LEAVE

When you want to leave the network, MH sends a MH LEAVE message to the fore-and-aft servers, which contains the information.

4. CLIENT LEAVE

When MH receives a CLIENT LEAVE message, the corresponding client is removed from the client list.

5. MH MOVE

The initial location of the servers is determined by its IP-hash results, but their fore-and-aft servers may be quite distant and cause a system imbalance. In order to solve the balance problem, MH will regularly check to see if the load is too high, detected when the server itself is loading higher than a given value, compared to the

next server in the hash ring. The hash ring moves in a clockwise direction in order to balance the system. Figure 8 depicts a mobile computing formula, where Dn represents the distance and Ln represents loading.

$$\frac{Dpredecessor - x}{Dsuccessor + x} = \frac{Lsuccessor}{Lidentity}$$

$$x = \frac{Lidentity \times Dpredecessor - Lsuccessor \times Dsuccessor}{Lidentity + Lsuccessor}$$

Figure 8. Formula for equilibrium shifting

Loading, CPU loading, and memcached utilization thresholds are three projects with the right weight and are defined by the value of each project as 1 to the integer γ, where γ represents the maximum. In this paper, γ is defined as 5.

In order to define CPU loading, our first experiment was to find out the relationship between CPU loading and performance, in order to define suitable parameters. Memcached utilization thresholds are defined as follows: when the utilization threshold is 0%, the value is 1; when the utilization threshold is 100%, the value is γ. The value is in accordance with the proportion of growth.

Loading is defined for memory loading. When upcoming memory loading is increased to a maximum of γ, its value decreases according to the proportion of growth when the server's remaining memory space is less than a certain amount (10% in this paper).

If x is the amount of movement, it will be a positive number; a negative could lead to read errors. When clients read data, they assume that the data is immediate new information. However, if MH moves forward or backward in a short while, clients may obtain older data. Figure 9 illustrates this phenomenon.

MH only deals with excessive loading, rather than dealing with loading that is too low. This is because there will be another higher loading server taking the load of lower servers to exchange equilibrium.

6. MH RECOVERY

Servers will malfunction and cause errors. MH regularly checks that their fore-and-aft servers are normal, and sends MH RECOVERY messages to other servers, which contains the information of their neighbors. When another MH receives this message, it checks the status message: if the status is Native, the message will be passed to the next server; otherwise, it would save this information. When MH detects a faulty neighbor, it will require this information in order to carry out the repair work.

Clients include three main components: memcached protocol, data distribution, and MH protocol. Memcached protocol is responsible for communicating with memcached servers and supports data distribution MH protocol so clients with MH servers are synchronized in order to obtain the status of the network.

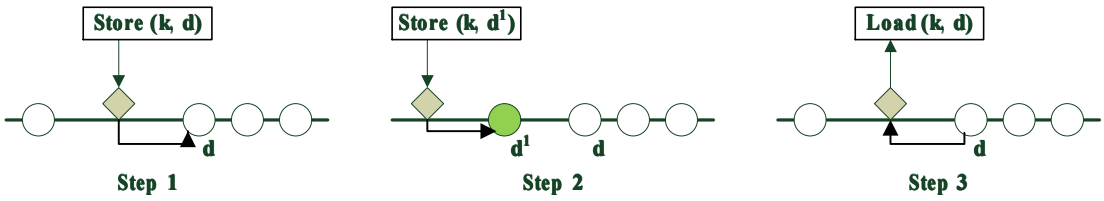

Figure 9. MH movement

At startup, clients will send a CLIENT JOIN message for MH servers to any network and from the server end, receive the online servers list. When clients need to write or read from the object, they decide which server will read the data connection with memcached protocol, in accordance with the definition of data distribution.

The clients MH protocol will receive three servers end messages: MH JOIN, MH LEAVE, and MH MOVE. When receiving MH JOIN, clients are added to the server list, and then removed from the list when receiving MH LEAVE. Receiving MH MOVE will update the server position.

Data is distributed by way of the general static hashing method, coupled with the number of replications, r, or number of retries, t, as parameters. At the time of writing, the clients will write data to consecutive r servers; read to the last servers and it reads until the cache hits or reaches the number of retries, t. In this system, the case cannot find the cache data, for two reasons: first, the data does not exist in the cache system, and second, the servers that store the data changed positions. We chose the last servers as the next target in the failed read because the server's position is in accordance with the counter-clockwise direction.

5. Conclusion

We designed a distributed architecture between memory cache and client servers. The designed architecture can be seamless to change members of the servers and client clusters, and the loading effect is balanced, so that clients can access the data in the most simplistic and fastest way. This architecture is scalable, with no need to change the existing memcached servers and client code. As described in the previous sections, the cost of the implementation of MH is not high, and most of the time, during a suspended status, the impact on the system performance is rather limited.

REFERENCES

[1] Bellatreche Ladjel, Benkrid Soumia, Crolotte Alain, Cuzzocrea Alfredo, Ghazal Ahmad, "The F&A Methodology and Its Experimental Validation on a Real-Life Parallel Processing Database System ", In Proceedings of 2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), pp.114-121, 2012.

[2] David Karger1, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, Yoav Yerushalmi, "Web caching with consistent hashing", Computer Networks, vol.31, no.11-16, pp.1203-1213, 1999.

[3] D.N. Serpanos, G. Karakostas, W.H. Wolf. “Effective caching of Web objects using Zipf’s law”, In Proceedings of 2000 IEEE International Conference on Multimedia and Expo, pp.727-730, 2000.

[4] George Karakostas, D.N. Serpanos. “Exploitation of different types of locality for Web caches”. In Proceedings of 2002 Seventh International Symposium on Computers and Communications, pp. 207-212, 2002.

[5] Jeff Bonwick, Sun Microsystems, “The Slab Allocator: An Object-Caching Kernel Memory Allocator”. In Proceedings of the USENIX Summer 1994 Technical Conference on USENIX, pp. 6-17, 1994.

[6] Kin-Yeung Wong, “Web cache replacement policies: a pragmatic approach”. IEEE Network, vol.20, no.1 pp.28-34, 2006.

[7] Christian Schindelhauer, Gunnar Schomaker, “Weighted distributed hash tables”. In Proceedings of 2005 ACM Conference on Parallelism in algorithms and architectures, pp. 218-227, 2005.

[8] Shan Lei, Ananth Grama, “Extended consistent hashing: an efficient framework for object location”. In Proceedings of 2004 International Conference on Distributed Computing Systems, pp.254-262, 2004.

[9] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, “Chord: a scalable peer-to-peer lookup protocol for Internet applications”, IEEE/ACM Transactions on Networking, vol.11, no.1, pp.17-32, 2003.

[10] Rongling Lang, Zhiqun Deng, “Data Distribution Algorithm using Time based WeightedDistributed Hash Tables”. In Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing, pp. 210-213, 2008.

[11] Jure Petrovic, “Using Memcached for Data Distribution in Industrial Environment”. In Proceedings of the 2008 Third International Conference on Systems, pp.368-372, 2008.

[12] Memcached, “What is Memcached?” http://memcached.org/, 2012.