Cache and Consistency in NOSQL

Peng Xiang, Ruichun Hou and Zhiming Zhou
Information Engineering Center, Ocean University of China
Qingdao, China
xiangpeng2007@163.com

***Abstract*—In recent years a new type of database (NOSQL) has emerged in the field of persistence. June 2009, a global gathering of NOSQL Movement triggered the fuse of "database revolution", Non-relational database has now become an extremely popular new areas, NOSQL aims to solve the needs of high concurrent read-write, efficient mass data storage and access, database scalability and high availability. In these large-scale concurrent systems, cluster cache and data consistency become the focus of attention. This article proposes a fast and effective approach to estimate the solution of the data consistency. We use node split and consistent hashing techniques to reduce the servers load by analyzing the request from the client. Each server node can be divided into a set of virtual nodes, which located in a ring. The complexity caused by the size of the set is determined by the target system. Through physical node split, we can reduce the cache miss rate. In order to reduce the complexity of the algorithm of consistency hashing, we will introduce a football game example.**

Keywords-persistence;NOSQL;clustercache;consistency; algorithm

I. INTRODUCTION

With the coming of Web 2.0, the most serious challenge so far to the supremacy of RDBMSs in managing data is the increasing need of mass data and concurrent access [2]. When we running web applications, the backend database systems are often the performance bottleneck and the existing database products seem to write data to increase the load to much. Recently, Twitter, Digg and Reddit and many other Web2.0 companies that switch from MySQL to non-relational database (NOSQL) to provide scalable data storage solutions, has aroused strong developers interest on NOSQL. Joe Stump who was the former chief architect of Digg said his own business project now is to use NOSQL, and he lists a series of questions to challenge SQL camps. He said, in terms of scalability, speed, cost, non-relational database have great advantage over relational databases in all the way. These "NoSQL" databases tend to originate from large internet companies that have to serve simple data structures to millions of customers daily [4]. Such as Google File System based on BigTable and Amazon's Dynamo are very successful business NOSQL cases. The databases specialize for certain use cases or data structures and run on commodity hardware, as opposed to large traditional database clusters. Their primary advantage is that, unlike relational databases, they handle unstructured data such as word-processing files, e-mail, multimedia, and social media efficiently.

As we all know, the network traffic problem became the largest obstacle. It has been shown that caching is the key solution that reduces network traffic. In recently years, some relation databases use the cache or to guarantee the data consistency and to reduce the database server's load. Just as the hibernate L2 cache (Ehcache) in J2EE, but the traditional single cache can't work well especially in the concurrent access. Voldemort, a distributed key-value storage system, unlike relational database, it is basically just a big, distributed, persistent, fault-tolerant hash table, and for applications that can use an O/R mapper like active-record or hibernate this will provide horizontal scalability and much higher availability but at great loss of convenience. In the traditional persistence application, there are always two or three sets servers, so, the scalable is not good, if one of the servers crashed, the system may face paralysis. And now, the Cluster Layer Cache become increasingly popular, and it has become a trend.MemCached (engineering team operating Live Journal's), a distributed memory object caching system, used to reduce the database load in a dynamic system[1] and improve the performance.

The paper is structured as follows. Section 2 presents cluster cache, we will introduce a distribute cache model ms-mc in non-relational database with MemCached and discuss how to read and modify data to in order to maintain data consistency and here we will reference Google App Engine. Section 3 details the data consistency, we will introduce a data consistency mechanism, Paxos, a very efficient transaction processing consistency algorithm, which can deal the consistency management well and Section 4 concludes the paper.

II. CLUSTER LAYER CACHE

High performance scalable web applications often use a distributed in-memory data cache in front of or in place of robust persistent storage for some tasks. Traditionally, replication of the database among several servers removes the performance bottleneck of a centralized database. However, this solution requires low-latency communication protocols between servers to implement update consistency protocols. Memcached has two core components: the server (ms) and client (mc), in a memcached query, mc first calculated the hash value of the given key to determine the kV's location in the ms.when ms is confirmed, the client will send a query request to the corresponding ms, and the ms will find the exact data. Because there is no interaction and multicast protocol among them, so the interaction of

978-1-4244-5540-9/10/$26.00 ©2010 IEEE

memcached brings the minimal impact to the internet. Fig. 1 shows an abstract view of the architecture of distributed system, which reducing the system load.

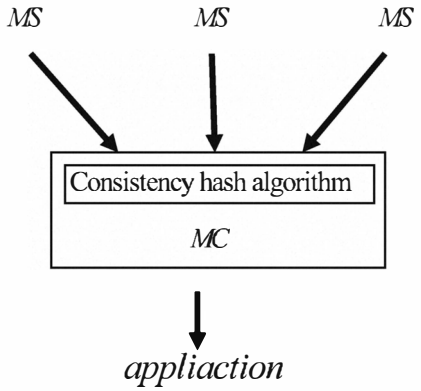

Figure 1. The architecture of distributed system

We can envisage that ms can also be divided, every ms can be treated as a physical node (PN), within each PN, there will be a variable number of virtual nodes (VN) running according to the available hardware capacity of the PN. So the query content is divided into several parts, which distributed in different ms. Fig. 2 describes the ms-mc structure, inside the ms, each virtual node connected through the network.

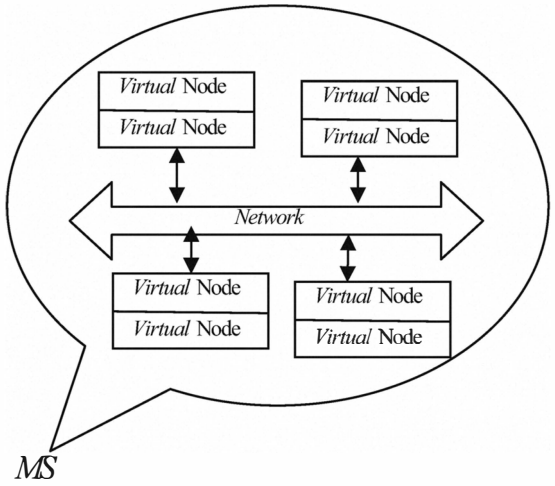

Figure 2. The ms-mc structure

With the support of distributed cache, the query efficiency has been greatly improved. In this paper, we use JCache, a proposed interface standard for memory caches, as an interface to the App Engine memcache. This interface is not yet an official standard, App Engine provides this interface using the net.sf.jsr107 interface package. JCache provides a Map-like interface to cached data. You store and retrieve values in the cache using keys. Keys and values can be of any Serializable type or class.

```
import java.util.Collections;
import net.sf.jsr107.Cache;
import net.sf.jsr107.CacheException;
import net.sf.jsr107.CacheManager;
// …
  Cache cache;
  Try
{cache=CacheManager.getInstance().getCacheFactory().crea
teCache(Collections.emptyMap()) ;} catch (CacheException
e) {
// …
}
String key;   // …
byte [ ] value;  // …
// Put the value into the cache.
Cache. put (key, value);
// Get the value from the cache.
Value = (byte [ ]) cache. get (key);
```

App Engine uses distributed cache in order to maintain data consistency, and in this paper, we assume that our system is based on the master/slaves system [7], there is a master server and a large number of slave servers. Each server includes a mc and ms, and we use memcache caching mechanism in each server. Each mc is a client, when you need a query from a proxy server, and the proxy server you choose will check if the content you query is in the ms, and if the ms can't find the required content, and then turn the next proxy server. Some NOSQL databases use the index and query processing mechanism in the local database (node), and we can use a DHT indexing [3] scheme that enables the proxy server to directly deal the cache query. In this case, we can use the system's query processing program to broadcast the query to all other nodes on the DHT, each node complete the local search, and then return the result to the system processor as a response. Note that the search is parallel in all the nodes on the DHT.

III. DATA CONSISTENCY

With distributed caching mechanism, our next task is to ensure data consistency. We know that there are many types of NOSQL databases, such as Wide Column/Column Families, Document Store, Key Value/Tuple Store, and Eventually Consistent Key Value Store etc; our data needs to partitioned and replicated using consistent hashing, and consistency is facilitated by object versioning [5], we need a suitable consistent hashing algorithm to ensure the consistency of replicas. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol.[6]the initial distributed architecture use hash () mod n, but its weakness is that if single point failure, the system can not automatically recover. In order to solve the single point of failure, we use hash () mod (n/2), so anyone have two options which can be selected by client, but this program still has problems: load imbalance, in particular, if one of the servers failures, the other server has excessive pressure. In consistent hashing, the output range of a hash function is treated as a fixed circular space or "ring" (i.e. the largest hash value wraps

around to the smallest hash value). Each node in the system is assigned a random value within this space which represents its "position" on the ring. Each data item identified by a key is assigned to a node by hashing the data item's key to yield its position on the ring, and then walking the ring clockwise to find the first node with a position larger than the item's position [8]. In order to solve load problem, we introduce Consistent Hashing, as mentioned above, we can use n servers and we can divide each server into several virtual nodes (v), each server can be seen as a physical node (pnode) and all the virtual nodes connected to a ring. Then we assigned the virtual nodes (n*v) to the consistory hash ring randomly, so the users through the hash algorithm to find the first vnode in the circle, and if the vnode failures, we can use the next vnode with a clockwise way. As Fig. 3 shows that if the hash value located in vnode a, but vnode a crashed, then we will use vnode b. If all the vnodes failured, then turn to the next vnode of next pnode.

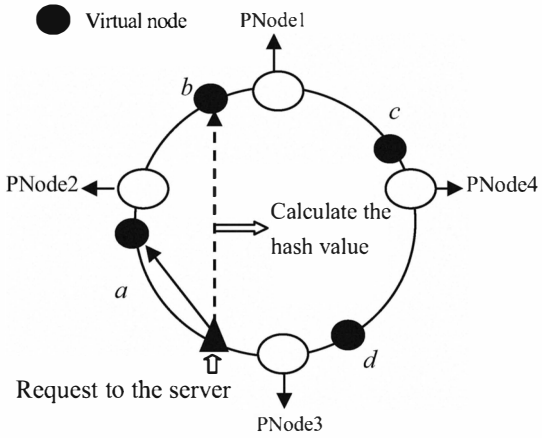

Figure 3. Example of hashing search

At last, we will introduce a very useful consistent hashing way: Paxos. Here we first give an example: A football game question.

If you (coordinator) would like to organize a football game, the other side has 11 players, so, in addition to yourself, you have to organize 10 members. The questions arise: first you have to call participants(10),and proposed that the game will organized on Saturday 2am-11am.Every participant should tell you if he can accept your proposal, if accept then the participant can not accept other requests at that time. In 2PC principle, if all participants agree the game, you notify them to give you a commit, or the game will cancel. The disadvantage of this approach is that if one participant node crash suddenly, there will be inconsistency in the cluster. Therefore, the 3PC replaced the 2PC theory: the3PC divide the commit process into two, precommit and commit. The disadvantage is that you and participants belong to two engine room, if the network between you disconnect, then your size will never receive the commit from participants and you will choose to abort. "There is only one consensus protocol, and that's Paxos, all other approaches are just broken versions of Paxos."Said by Mike Burrows (the author of Google Chubby). Compared to 2PC/3PC, Paxos algorithm improved.

P1a. Every time, Paxos instance will assigned a number which needs to increment, each replica doesn't accept the proposal which is smaller to the greatest number in the current.

P2. Once a value v is passed by replica, then this value can not be changed, that have no Byzantine question. Take the above football game as an example; if the participant accepts the plan, he can't change his idea no matter who ask him to change. If most participants accept the plan, and the proposal can be adopted. So the Paxos is more flexible than 2PC/3PC, in a cluster with 2f+1 nodes, we allow f nodes unavailable.

IV. CONCLUSIONS

Applications deployed on the Internet are immediately accessible to a vast population of potential users, and with the coming of WEB2.0, users need more and more data. As a result, the current relational databases product exist serious load problem, a large number of unstructured information need to interacted with the database, so we need a distributed and scalable framework. Relational databases can also be distributed, but in many ways, compared with relational databases, NOSQL databases deal better, in particular in the data consistency. In terms of technical system, relational databases more stress concentration or a limited nodes cluster structure, but, don't design the content with low-cost of distributed computing unit. So that RDBMS can not support world-class information management technology like GoogleBigTable, Amazon Dynamo etc. In this paper, we discussed distributed caching mechanism, and introduce the data consistency plan; there are many consistent hash algorithms, such as gossip, DHT, and Map Reduce Execution and Data Partitioning etc. They all have their own advantages, here due to limited space; we can't give more details about them. In Master/Slave model, we can treat the slave server as proxy server, the distribute cache we introduced is equivalent to query caching to the database. We use the approach to scale the database in order to reduce the database load. Our future works are: provide a high cache hit rate algorithm to reduce the load on the central database server and efficiently ensure the cache consistency as the database is updated. As Ricky Ho said, compared with relation database, the query search is nosql's short board. At present, many nosql databases are based on DHT (Distributed Hash Table) model, so the query is equivalent to access the hashtable, and in the future we will further study query processing combined with NOSQL design patterns.

REFERENCES

[1] A.P.Atlee and H.Gannon, "Specifying and Verifying Requirements of Real-Time Systems," Proc. IEEE Trans.Software Engineering, IEEE Press, Jan.1993, pp. 41-45, doi:10.1109/32.210306.

[2] R.V.Nageshwara and V.Kumar, "Concurrent Access," Proc. IEEE Trans.Computer, IEEE Press, Dec.l988, pp.1657-1665, doi:10.1109/12.9744.

[3] Yuzhe Tang, Shuigeng Zhou, and Jianliang Xu, "LIGHT: A Query-Efficient Yet Low-Maintenance Indexing Scheme over DHTs,"Proc. IEEE Trans.Knowledge and Data Engineering, IEEE Press, Jan.2010, pp.59-75, doi:10.1109/TKDE.2009.47

[4] Neal Leavitt, "Will NoSQL Databases Live Up to Their Promise?" Compuer,Vol. 43, pp.12-14, Feb.2010.

[5] M. Andrew, ""NoSQL" databases in CMS Data and Workflow Management," 13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research, Physics Department, University of Rajasthan . LNM Institute of Information Technology. Jaipur, pp. 22-27, Feb 2010.

[6] L. Lamport, "Time, clocks, and the ordering of events in a distributed system,"Communications of the ACM,Vol.7, pp. 558-565, July.l978.

[7] A.Olson and K.G.Shin, "Probabilistic Clock Synchronization in Large Distributed Systems," Proc.IEEE Trans.Computers, IEEE Press, Sep.1994, pp.1106-1112, doi:10.1109/12.312120

[8] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," In Proceedings of the Twenty-NinthAnnual ACM Symposium on theory of computing. El Paso.Texas.United States, pp.654-663,1997.