Load Balancing Technology Based On Consistent Hashing For Database Cluster Systems

Zhenguo Xuan
Dept. of Computer Science,
Beijing University of Posts and Telecommunications
Beijing, China
xuanzg@bupt.edu.cn

***Abstract*—The database cluster is an effective mechanism to improve the throughput of database systems and reduce the response time of the database. Generally, it is used to solve the single-node bottleneck problem of data access in network services. Load balancing technology is the key factor to determine the performance and scalability of the entire database cluster. In this paper, the authors introduce the concept of the database cluster and several load balancing technologies, and then manage to improve the consistent hashing algorithm by dividing the hash ring equally and setting re-distributed node regularly to make it more balanced and predictable, at last we proposal a new database cluster model using the consistent hashing algorithm for specific scenarios.**

Keywords-database cluster; load balancing; consistent hashing

I. INTRODUCTION

With the rapid development of Internet, we are moving into a new age called "data explosion". The data which twitter produced each day can be over 20 trillion bytes. Anybody could get huge amounts of data with a personal computer or a smart mobile.

The increasing of mass data access makes the database nodes which provide data storage service the bottleneck of the network service system. The database cluster is the commonly used approach to solve the problem for its characteristics of high availability, high performance and high scalability, and load balancing technology is the key points and difficult points of the database cluster study for load balancing technology directly determines the architecture, efficiency and scalability of the database cluster.

There are a lot of classification methods for load balancing technology. According to the system architecture, it can be divided into two types: having front-end nodes and having no front-end nodes [1]. The former provide multiple IP addresses for clients to access, it is easy to implement have little bottleneck in the structure whereas it is unsatisfactory on reliability. The latter makes the users have the illusion of a single IP address which is responsible for receiving the requests and forwarding it to the comparatively lightly loaded destination nodes in the cluster. Additional security can be got by this way, yet in the case of very large network traffic, the front-end scheduling node may become the bottleneck of the system. Load balancing technology can also be divided into static scheduling and dynamic scheduling by the way of setting the scheduling information. The former dispatches the requests to different nodes in database cluster in accordance with established scheduling rules. The latter will collect the load information of each node in real time during scheduling process, and then take the information into the scheduling algorithm to choose the right node to which to distribute the requests.

Server clustering technology LVS (Linux Virtual Server) is currently the most popular cluster architecture, which is widely used to build scalable Web, Media, Cache and Mail, and other network services [2]. Meanwhile, the architecture can be applied to the database cluster service. What is more, the database cluster needs to consider the classification of the requests and the consistency of data among different nodes.

Considering the application of massive data access, especially when read requests constitute a big percentage of the requests from clients such as the query of IP, personal IP or train schedules etc, the database just handles query requests and a little insert or update requests after being set up. In this case, consistent hashing scheduling algorithm can provide excellent performance on throughout and minimize the effects of immigration and emigration of node.

The rest of this paper is organized as follows: section 2 introduces server cluster and then describe the characteristics of database cluster. Section 3 introduces the concept of load balancing, and then in section 4 different load balancing schedule algorithms are discussed, especially the feature of consistent hashing, and then we improve the algorithm for the specific scenario. Section 5 proposes a prototype database cluster system using load balancing policy of consistent hashing algorithm. Section 6 gives the final conclusion of this paper and the future work.

II. DATABASE CLUSTER

The computer cluster is a kind of computer system, it can complete computing works by the closely collaboration of computer software and/or hardware which are loosely coupled. In a sense, the computer cluster can be regarded as a single server. Every computer in the cluster system is often referred to as node and the nodes usually connected by LAN or other ways. The computer cluster is usually used to improve the computing speed of a single computer and/or reliability.

Generally speaking, the computer cluster has a better price/performance ratio than a single server or workstation. The cluster system can be classified as follows: High Availability Clusters (HAC), High Performance Clusters (HPC), Load Balancing Clusters and Grid Computing [3].

LVS (Linux Virtual Server) take use of IP load balancing technology and content-based request distribution technology. There is a front-end scheduler in the architecture of LVS. The scheduler has a good throughput and transfers the requests to different servers, what is more, the scheduler automatically masked the failure of the nodes, and then make a group of nodes constitute a high-performance, high availability virtual server[2]. The architecture of LVS is as figure 2.1 shows.

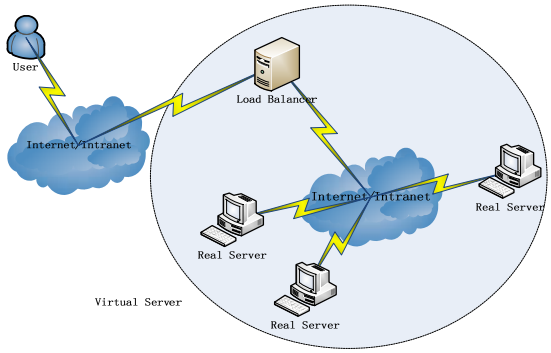

Figure 2.1 The architecture of LVS

The large-scale and high-performance database cluster system can be achieved by using cluster technology to database system. The difference is that the database system should handle information processing and information storage, so each node in the cluster need to ensure data consistency and this can be achieved by shared data model or using a synchronization mechanism. The models are shown in figure 2.2 and figure 2.3.

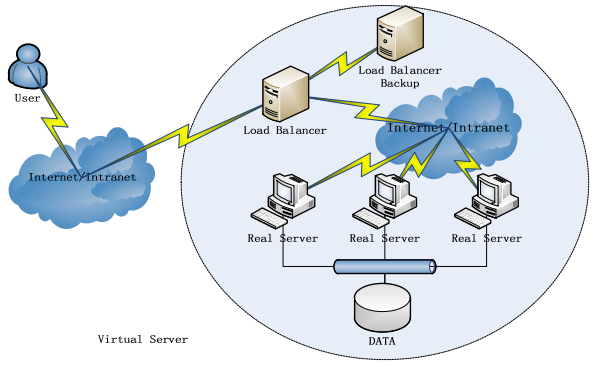

Figure 2.2 The architecture of cluster using shared data

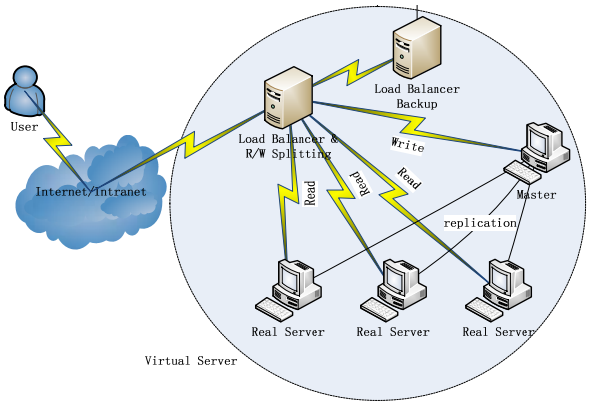

Figure 2.3 The architecture of cluster using synchronization mechanism

Usually the load balancer has a backup server to replace itself when it is crashed. The backup server can get all the information of the load balancer by hot copy technology and it can be seen as the shadow of the load balancer. By this way, the high availability of the cluster is ensured.

III. LOAD BALANCE

A. *Description of Load Balance*

In order to improve the performance of the system, the load should be distributed reasonably to different nodes in the cluster, and this form of sharing computing capacity is referred to as load balancing or load sharing [4].

Load balancing techniques involve five basic policies: information policy, job selection policy, location policy, transfer policy and initiation policy. The basic policies have covered the main technical concerns of load balancing for distributed systems [5].

The following 10 kinds of scheduling algorithms have been achieved in LVS:

a) Round-Robin Scheduling

b) Weighted Round-Robin Scheduling

c) Least-Connection Scheduling

d) Weighted Least-Connection Scheduling

e) Locality-Based Least Connections Scheduling

f) Locality-Based Least Connections with Replication Scheduling

g) Destination Hashing Scheduling

h) Source Hashing Scheduling

i) Never Queue scheduling

j) Shortest Expected Delay scheduling

In addition, there are many new dynamic scheduling algorithms for specific scenarios. [1] proposed a new scheduling algorithm based on virtual router cluster, [5] achieved load balance in RFID middleware system using multiple load balancing strategies, [6] achieved a dynamic scheduling algorithm which can monitor node performance in

real time, [7] achieved load balancing in the stock trading application using dynamic load balancing strategy, [8] proposed an load balancing algorithm without load balancer by doing packet filtering on the switch.

B. Consistent Hashing Algorithm

Consistent Hashing algorithm is widely used in load balancing for its characteristics of availability, scalability and avoiding hotspots easily. Consistent Hashing is proposed in [9] aiming at solving the problem caused by the number of nodes changed in cache system using classical hash algorithm. Consider what happens when the set of active caching machines changes, or when each client is aware of a different set of caches. (Such situations are very plausible on the Internet.) If the distribution was done with a classical hash function (for example， the linear congruential function：$x \rightarrow (ax + b) \pmod p$), such inconsistencies would be catastrophic. When the range of the hash function (p in the example) changed, almost every item would be hashed to a new location. Suddenly, all cached data is useless because clients are looking for it in a different location [9].

To understand how consistent hashing works, imagine wrapping the unit interval [0,1） onto a circle, as shown in figure 3.1.

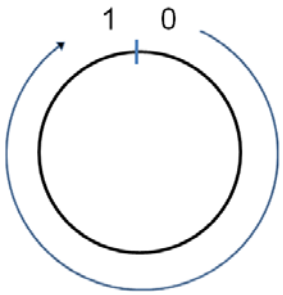

Figure 3.1 Hash ring

Suppose we number the machines [0 … n-1]. If the hash function has range [0, R), then we rescale the hash function via x→hash(x)/R, so that the hash function maps into the range [0,1）, i.e., effectively onto the circle. Then we can hash machine number j to a point hash(j) on the circle, for each machine in the range j=[0 … n-1]. Figure 3.2 is what it might look like for an n=3 machine cluster.

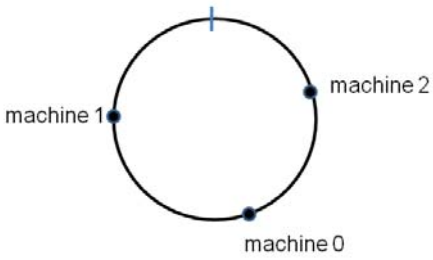

Figure 3.2 Set machines point by number

The points will be randomly distributed around the circle. Now suppose we have a key-value pair we want to store in the distributed dictionary. We simply hash the key onto the circle, and then store the key-value pair on the first machine that appears clockwise of the hash point of the key. As shown in figure 3.3, for the key shown here, the key-value pair is stored on machine1.

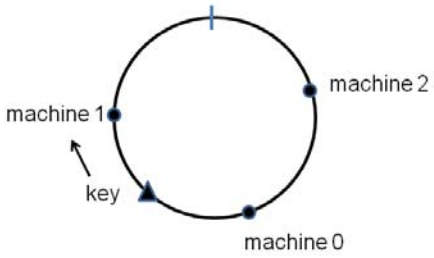

Figure 3.3 How to find cache node

Because of the uniformity of the hash function, a fraction roughly 1/n of the key-value pairs will get stored on any single machine. Now imagine we add an extra machine into the cluster. It goes to the point hash(n), see figure 3.4.

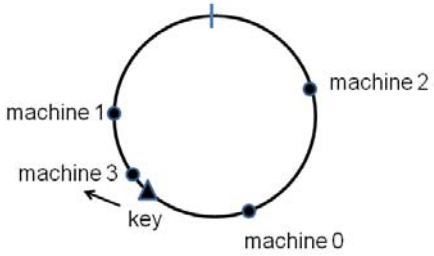

Figure 3.4 Add node

Most of the key-value pairs are completely unaffected by this change. But we can see that some of the key-value pairs that were formerly stored on machine 1 (including our example key-value pair) will need to be moved to the new machine. But the fraction that needs to be moved will typically be 1/(n+1) of the total, a much smaller fraction than was the case for classical hashing [10].

There is a problem that when a machine is added to the cluster, all the keys redistributed to that machine come from just one other machine. Ideally, the keys would come in a more balanced way from several other machines. So take use of a new strategy based on Amazon Dynamo [11]. This strategy divides the hash space into Q equally sized partitions and the placement of partition is decoupled from the partitioning scheme. Moreover, each node is assigned Q/S tokens where S is the number of nodes in the system. The figure 3.5 gives an example with 12 partitions and 3 nodes (Q=12, S=3). N should be a very big integer like 2^{20} and the hash function should be like $x \rightarrow (\text{hash}(x) \bmod N)$ by which the requests are assigned to the tokens owned by nodes.

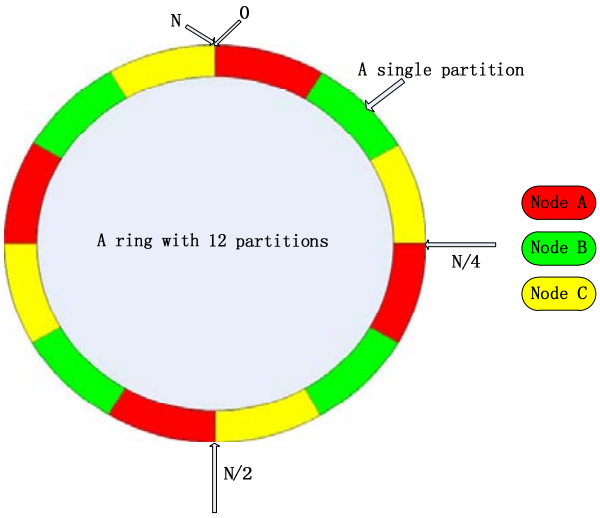

Figure 3.5 Improve hashing by equally divided ring

When a node leaves the system, its tokens are distributed to the remaining nodes next to it in the clockwise direction such that these properties are preserved and the load is predictable. Similarly, when a node joins the system it "steals" tokens from nodes in the system in a way that preserves these properties. Figure 3.6 can simulate this process. To keep it simple, the value of N is set to be equal to Partitions (N=12).

```
Node_num = 3;
Partitions = 12;
Node [Partitions] = {A,B,C, A,B,C, A,B,C, A,B,C};
//load balance
N = hash(request_id)/ Partitions;
If Node[N] = 0; N++;
Send request to Node[N];
//add node D
Node [Partitions] = {A,B,C, D,B,C, A,D,C, A,B,D};
//remove node B
Node [Partitions] = {A,0,C, D,0,C, A,D,C, A,0,D};
```

Figure 3.6 Process simulation

When a machine is added to the cluster, the keys redistributed to that machine come from the other machines and so it is a more balanced way.

IV. DESIGN THE PROTOTYPE SYSTEM

This section will draw on the common architecture of LVS, using the model with front-end scheduling node on which load balancing strategy is employed. In order to ensure consistency of data, read/write splitting strategy is adopted. For the reason that read requests account for the very great proportion in all requests in the scenarios, the design is to forward read requests to a node determined by load balancing strategy and distribute write requests to each node directly in order to guarantee data consistency without synchronization strategy, and the efficiency is improved. The specific implementation can refer to [3]. The model can be shown in figure 4.1.

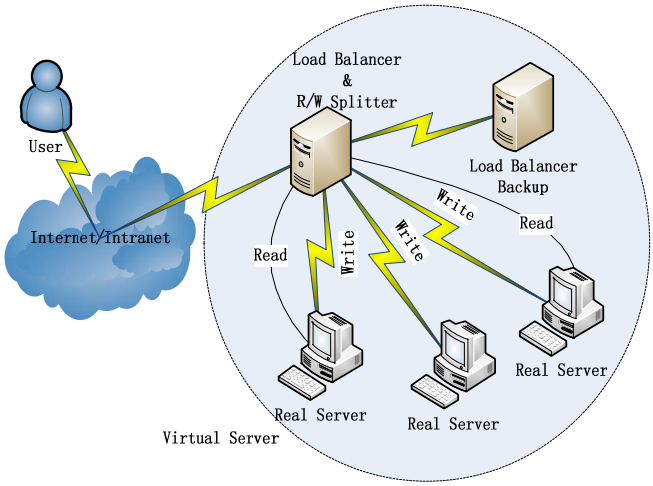

Figure 4.1 The architecture of the model

In this model, the load balancer and the splitter is the same server and it has a backup for itself to ensure high availability and security of the cluster. The splitter sends write requests to all the nodes in the cluster to ensure the consistency of the data. When a read request comes, it is sent to the node chosen by load balancing algorithm to achieve high throughput. Consistent hashing is the key to ensure the scalability of the cluster. Sending write requests to all the nodes influences the performance but the lucky is there is little write requests in our scenarios. The whole cluster can be seen as one virtual server for the users.

V. CONCLUSIONS

The paper introduces the concept of database clustering and load balancing, improves consistent hashing algorithm and applies it to the scene of high concurrent queries. In the following, the paper designs a model of a database cluster based on the general architecture of the LVS using a front-end load balancer node and R/W slitting technology in order to ensure the high availability, reliability and scalability of the cluster. In addition, data consistency can be resolved by this architecture and so the model is of high reference value when building distributed database cluster system. Also there are some shortcoming in the improved algorithm, for instance, it does not take the performance of each node into account, so it can be further optimize.

REFERENCES

[1] Cheng Wei, Lu Ze-xin, Wang Hong. A New Kind of Load Balancing Technology for Server Cluster Systems. COMPUTER ENGI NEERI NG & SCIENCE, Vol. 28 No. 2 2006

[2] Zhang Wen-song. Linux Virtual Server Clusters. IBM developerWorks Magazine 2002.

[3] Yan Xian-you, Zhang Xiao-ling. Study and Realization of dynamic load balancing based on Linux virtual server cluster. Beijing University of Technology. 2009

[4] Liu Tong, Yang Shu-qiang. Study and Realization of load balancing Technology in Database cluster. National University of Defense Technology. 2009

[5] Heung Seok Chae, Jae Geol Park, Cui Jian-feng, Joon Sang Lee. www.interscience.wiley.com,. 2010-3-23

[6] He Jun, Xiong Wei, Chen Luo, YinJia-xin. Study and implementation of dynamic load balancing based on database cluster. Network and Communication. 2011

[7] Yang Xiao-hu, Zhao Li-ping, Wang Xin-yu, Wang Ye, Jie Sun and Albert Jerry Cristoforo. Satisfying quality requirements in the design of a partition-based, distributed stock trading system. Wiley InterScience, www.interscience.wiley.com. 2011-1-27

[8] Xie Zuo-gui, Qi Xiao-ya. High-availability and Load-balance Cluster System without Load balancer. Computer Engineering, VOL.33, No.3 2007

[9] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, Rina Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. 1997.

[10] Michael Nielsen. Consistent hashing. http://michaelnielsen.org/blog/consistent-hashing/. 2009

[11] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. www.Amazon.com. 2007