Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, February 15-17, 2006 (pp151-155)

Fault-Tolerant Framework for Load Balancing System

Y. K. LIU, L.M. CHENG, L.L.CHENG
Department of Electronic Engineering
City University of Hong Kong
Tat Chee Avenue, Kowloon, Hong Kong SAR
HONG KONG

Abstract: In this paper, a fault-tolerant framework for load-balancing system is proposed. Load Balancing is widely used in the market, however, they usually cannot recover the unfinished jobs under stateful protocol such as TCP. In the practical situations, duplicating servers are commonly used to handle this problem, but the cost of the system and maintenance for those systems are expensive. In the proposed framework, we solve the problem with a contention method. Servers are grouped together and share the information of clients' connections and monitor others servers in their group. When a server in a group does not response the other members, they will elect one server to take over the job of it and the whole process is totally transparent to the user.

Key-Words: Load-Balancing, Fault-tolerant, Stateful Protocol, Contention based, Connection Recovery

1 Introduction

As web technology is rapidly growing, the performance and reliability of the web systems is a great concern of many companies. Load balancing technique is usually used in the high traffic systems [1]. Load Balancing System distributes the workload of a specific service to a group of functional computers according to each server's situation, so the performance of the whole system will be enhanced.

The most popular Load Balancing approach is using a computer or router or DNS server to act as a job distributor [2]. According to the job distribution algorithms, the distributor redirects the incoming job from clients to the servers fairly [1, 2, 3].

Load-Balancing algorithm makes web system more reliable because it provides basic fault tolerant function for the systems. The load-balancer monitors the status of backend servers. It will not forward new jobs to a server if it finds that server is downed, so the systems can continue to work normally, if only some of servers downed. However, load-balancing system will not recover the connections which are held by the servers which are out of services. If stateful connection protocols such as TCP are used, the connections will be abandoned by the system. And the clients have to reconnect to the system again after the protocol timeout [4]. That is not acceptable for the most type of services provided though the Internet such as online game and remote terminal.

In this paper, a framework for fault-tolerant load balancing is proposed. This framework is based on traditional centralized load balancing structure and it enhances the fault recoverability of the whole system. Under this framework, the failed connection can be redirected to the other servers transparently.

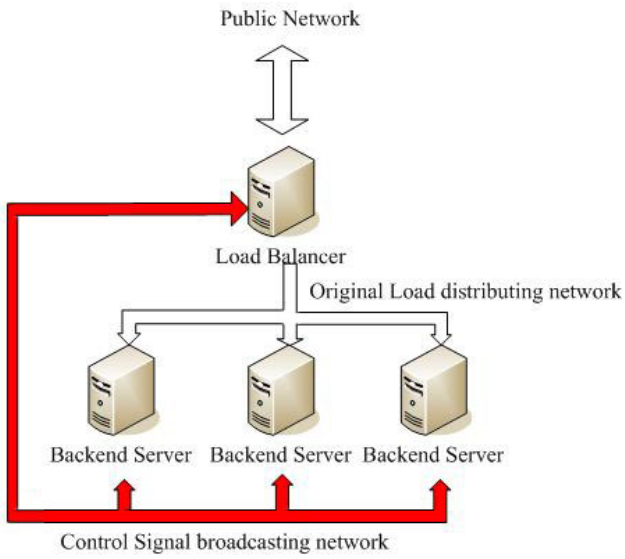

Fig.1: Network topology of the Fault-Tolerant Protocol.

2 Fault-Tolerant for Traditional Load Balancing

Fault-tolerant function is one of important features of Load Balancing algorithm. The load-balancer checks the healthiness of each server periodically. If a failed server was found, no more new job will be assigned to that server. The remained servers in that system will share the workload of the failed server. Although the performance of the services will be degraded, the services of the system will not be affected when a backend server downed. However, it cannot recover the current connections of the failed server. All of them will be lost when a server downed. Users have to wait for the connection timeout and reconnect to the system again. Also, it will cause other problems such as transaction and data losses. There are few methods to reassign the unfinished connections to the other servers such as ST-TCP [4], however, the redundancy of system resources is high. Thus, the cost of the system and maintenance is relativity high in such algorithm.

2.1 ST-TCP Protocol

ST-TCP Protocol relies on the existence of an active backup TCP server that takes over the TCP connections case of primary TCP failure [4]. The heartbeat packets send periodically from the primary server to the backup server to monitor the healthiness so the primary server. Whenever a primary is downed, the backup server will continue the connections accepted by the primary server. The backup server is a clone of the primary server, so it can completely restore the connections. And the transfer from primary to backup server is transparent to the clients.

In this algorithm, the ratio of the primary server and the backup server is 1:1. That is not an efficiency way to implement the fault-tolerant solution.

2.2 M-TCP

The M-TCP is a Protocol under Layer 3 and 4 and providing connection migration service []. When a client starts a connection to the server, the server will supply the addresses of its cooperating servers. The client-side M-TCP initiates migration of a connection by opening a new connection to a cooperating server. And the server will synchronize the status information of the connection with the original server. This process has to be initiated by the client and it is assume that the original server can provide the status information to the target server.

3 Structure of Fault-Tolerant Framework

In the proposed framework, the clients' connections can be recovered in the failure of servers. Also the resources redundancy in the structure is very low. Load balancer and backend servers are connected by two separate communication networks -- a "load distribution network" and a "control signal network". While the first one is used for load balancing purposes, the other one is mainly used for backup and recovery. The structure is shown in Fig.1.

```
while(true){
    If(Heartbeat of Server1 timeout){
        Delay=τ+P
        while(true){
            Wait(Delay)
            if(Req or GUI is received)
                break;
            Broadcast(Req)
            Wait(T1)
            if(GUI is received)
                break;
            if(Req is received){
                Delay=2*(Delay+τ)
                continue;
            }
            else{
                Broadcast(GUJ)
                Wait(T2)
                if(GUI is received){
                    Delay=τ+P
                    continue;
                }
                Recover the unfinished connections
                break;
            }
        }
    }
}
```

Fig.2 Algorithm of a server in Contention Phase

When a server received a packet form the load-balancer, it determines the time (T_n) required for the next broadcast packet. Next, it packs T_n and other control information with the original packet received from the load-balancer and broadcast it to the peer server in the same group. Whenever a peer server received these packets, it will store them in the cache for error recovery. To minimize the overhead of status storage and usage of bandwidth, the backend servers can be divided into groups, so that each server will handle the packets from only parts of servers in the system.

3.1 Failure Recover of System

As mentioned before, when a server broadcasts a packet, the packet includes an item that shows the time limit for the next broadcast packet and the server will send out another packet within that time. A timer will be set when a server receives a broadcast packet from the same group and it will be reset if another packet is received form the sender. If the sender cannot send out a packet within that period, it will broadcast an extend-reply message to acknowledge the other peer server that it is currently working. Otherwise, the other servers will assume the sender was malfunctioned and try to handle the connections of it if the timer is timeout.

3.2 Contention Phase

When a failed server is detected in the system, the protocol will enter contention phase. In this phase, the peer servers in the same group will elect one server handle the unfinished jobs of it. First, servers broadcast a "Req packet" which contains the identify number of the downed server after a predefined delay. It is the message to inform other servers that it is ready to handle the unfinished connection. The delay is defined to be τ + P(Server) where τ is the random time delay of job recover, and P is priority of each backend server determined by the administrators. And the servers will wait an interval T1 after the Req packet is send. If other Req packets from other peer machine are received before the end of the interval, it waits another longer interval and sends Req packet again until Req packet from other servers is received before delay time finished or after the next T1 interval finished. For each unsuccessful request, the T1 will be changed:

$$Delay[n] = 2*(Delay[n-1]+ \tau) \qquad (1)$$

After the T1 period, if there is no Req packet received from other servers, it will send out a GUJ packet which is an acknowledgment to the peer servers that it won in the contention phase and the server will wait T2 interval. This server will start to handle the connections if there is no other GUJ packet is received within T2. If GUJ packets received within T2, the process starts the contention phase again by sending out the Req packet with interval τ + P(Server). After the server take over the unfinished job, the connection can be recovered with the status storage in the cache. The detail of the

algorithm is shown in Fig.2.

4 Simulation

Simulations are performed to investigate the servers switching time during server failure occur. NS is used as the simulating program.

4.1 Environment Setting of Simulation

The framework is simulated under the topology showed in the Fig.3. It contains 300 clients and 100 servers, and they are connected via a load-balancer though external network. At the same time, servers are grouped together and each server in a group was connected with star topology as internal network. Each client was connected to the load-balancer with 10Mbps link, and 100Mbps were used in the internal network. Each client generates job requests to the load-balancer at a random time and these jobs were distributed to the servers by the load-balancer. To distribute the jobs to the servers fairly, Random Distribution Algorithm was used in the simulation. Servers process the job requests from clients and return the results to them with source address in incoming packets. Also each server distributes the incoming packets through multicasting method. Centralized Multicast Protocol [5] is used in the simulation. Detail of environment in the simulation is showed in Table 1.

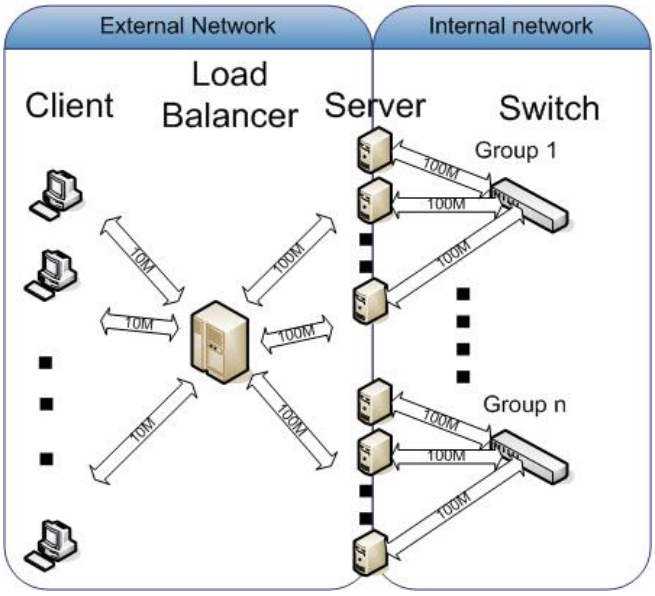

Fig.3: Topology of simulating

Parameters of network	
External Link Capacity	10Mbps
External Link Propagation Delay	Random 10-50ms
Internal Link Capacity	100Mbps
Internal Propagation Delay	20ms
Server Operation Delay	Random 10-20 ms
Rate of Request of Client	Random 3.75-18.75 per sec.
Average size of request packet	1000 bytes
Average size of reply packet	2000 bytes

Table1: Parameters of network

Simulation parameter	
T1	80ms
T2	80ms
Max τ	160ms
P()	Fixed 8ms

Table2: Simulation parameter

4.2 Simulation Result

In this experiment, the relationship between size of server groups and the job switching duration in server failure is determined. Server failure is simulated in a straight forward way by switching off the corresponding computer. We put different numbers of servers into a server group and the time required for job switching between servers during server failure occur are recorded. For each case, the experiment is repeated 10 times, and the average duration time will be used as result. The result of the experiment is showed on Fig.4. Parameters of the simulation are list on Table 2.

Fig.4 shows the switching time of connection in the server failure. As could be expected, the switching time

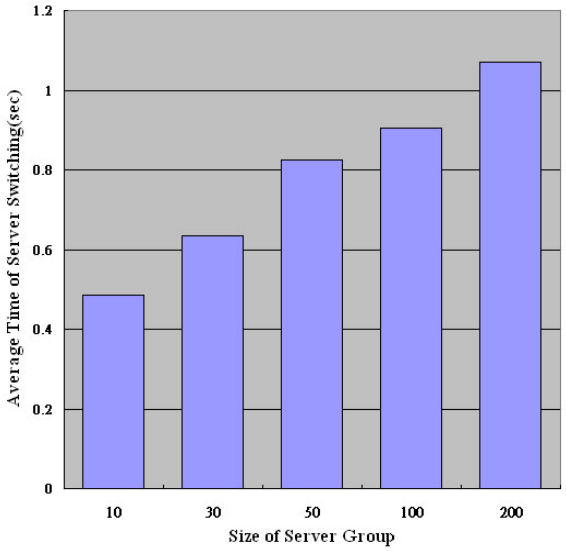

Fig.4: Average Time of Job Switching at different Group Size

of the connections between servers is proportional to the number of servers in the server group. All the connections switching times in the experiments are less then 1.07sec. And reach minimum 0.49sec at 10 servers in a group.

5 Conclusion

In this paper we have proposed a structure to provide fault-tolerant function in load-balancing systems. Under this framework, a low cost and high efficient load-balancing system with fault tolerant function can be build. Compared with the solution based on backup server method which highly duplicates the resources in system, this framework minimizes the cost of the system. In addition, the clients do not notice any server failure during the connection recovering time. Moreover, no special wrapper or libraries are needed in the client machines. It is obvious that the proposed scheme is a cost controlled solution to enhance the reliability of load-balancing system..

References

[1] Hui, C. C. and Chanson, S. T. “Improved Strategies for Dynamic Load Balancing,” *IEEE Concurrency*, vol. 7, July 1999.

[2] Cardellini, V., Colajanni, M. and Yu, P. S., “Dynamic Load Balancing on Web-server Systems,” IEEE Internet Computing, 3(3):28--39, May/June 1999.

[3] Wolf, J. L. and Yu, P. S., “On balancing the load in a clustered web farm”, ACM Transactions on Internet Technology (TOIT), v.1, n.2, November 2001, p.231-261.

[4] Marwah, M., Mishra, S. and Fetzer, C., “TCP Server Fault Tolerance Using Connection Migration to Backup Server”, Proc. Of IEEE Int. Conf. on Dependable Systems and Networks (DSN 2003), June 2003.

[5] Paul, P. and Raghavan, S. V., “Survey of Multicast Routing Algorithms and Protocols.”, Proceedings of the Fifteenth International Conference on Computer Communication(ICCC 2002.