Eliminating TCP Connection Failure in a Web Server Cluster

Anjali Verma
Computer Engineering Department
SIRT Bhopal, Madhya Pradesh, India

Vaishali Ughade Lokhande
Computer Engineering Department
SIRT Bhopal, Madhya Pradesh, India

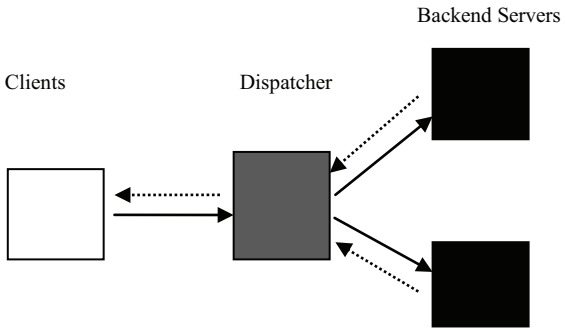
Saurabh Khare
Computer Engineering Department
SIRT Bhopal, Madhya Pradesh, India

Abstract- Web server clustering is most widely used technology for large enterprise applications to host online web services. It has been widely used to improve the performance of the web servers. Moreover, as the web has become an important business service delivery infrastructure, the need for supporting TCP connection failure in a web service has become more important. It is greatly affected by the amount of traffic on the backend servers. In such a scenario this paper proposes a method for eliminating the situation of TCP connection failure transparently from the end users in a dispatcher based, application level web server cluster and this reduces the connection time of the client with the server. The effectiveness of the proposed method conformed experimentally.

Keywords – **Web Server Cluster, Dispatcher, Backend Server, Load Distribution, TCP Socket accept queue**

I. INTRODUCTION AND BACKGROUND

Servers cluster are the most popular configuration of network used to meet the growing traffic demands imposed by the World Wide Web. This technique is used for improving the performance of the web server by sharing the work over multiple low capacity servers, to work together closely and to form a single logical, high capacity and high availability server. Clustered server is usually deployed to get better performance and availability over that of a single server through the redundancy, while typically being much more cost-effective than single server of comparable speed and availability. Due to the exponential growth in traffic, Server clustering needs an efficient load distribution policy to share workload across the multiple servers [12].



Figiure 1. Dispatcher Based Application Level Web Server Cluster

Issues which encourage load distribution -

1. In a clustered environment the specification and configuration of the servers are different. Server with different capacity may exist together i.e. they are heterogeneous in nature.

2. At all the time the server load is not always constant. Some request may offer much heavier load then other.
3. The server load may get imbalanced if request are distributed to each server with equal probabilities.
4. The nature of the traffic is changing (due to the multimedia application) and these applications require bandwidth reservation and delay guarantee. But the actual traffic is very difficult to predict. Efficient Load distribution policy can improve the network throughput for the arbitrary traffic pattern.

Load distribution policy can be represented by the two major groups -

1. Static
2. Dynamic

Static load distribution policy allocates the tasks in a round robin or cyclic manner to servers. But this is not the actual way of load distribution because some tasks are complex and they take long time to processes. This sort of algorithm having simplicity in terms of both implementation as well as overhead since there is no need to constantly monitor the servers for performance statistics. Static policies having limitation that they can only work well when there is not much variations in the load and they are not used in the environment where the load may vary significantly at various time in a day [4].

Dynamic load distribution policies have significant improvement in performance over static algorithms. They perform the task of the load distribution at run time [4]. They use current or recent load information when making distribution decision. At the time of distribution the load on each server is compared and the load is allocated to that server which is least loaded. When we came to know that particular server get imbalanced then we can migrate some of the load to the other server which is lightly loaded. We have to continuously monitor the servers for getting this information. Dynamic load distribution policies have limitation that they require the additional cost of collecting and maintaining load information so it is important to keep these overheads with reasonable limits. With this method, the load measurement program, running on the servers itself generates additional processing load on the server, degrading its performance. Moreover, if a server's performance is extremely degraded with excessive load, the measurement program may not work efficiently.

The proposed load distribution technique is used for any TCP client- server based services. Since WWW is the most heavily used service in the Internet, the efficient load distribution policy for WWW is more frequently required. Therefore, this paper proposes the load distribution policy which is based on passive measurement. This approach is known is the passive because here we improve the performance of the web server cluster without executing any additional program on the backend servers.

II. RELATED WORK AND PROBLEM DESCRIPTION

The exponential growth of the Internet and its application in the recent years has created the need for faster web servers to reduce connection time and provide better web services. An alternative to a powerful mainframe would be cluster of processors as web server. An important issue is the load distribution scheme adopted, which influences the performance and scalability of such architecture. Dispatcher server usually receives the connection requests from the clients and distributes them to the backend servers. Several studies indicated that the connection establishment of the clients with the server are closely related to the capacity of the socket accept queue of the backend servers [2, 3, 11]. This queue contains connection state information of all TCP connections. The capacity of the socket accept queue indicates the maximum number of simultaneously opened TCP connections whose connection state information can be kept in it without its overflow. This capacity varies from operating system to operating system. If the service demand exceeds the capacity of the socket accept queue, the socket buffer overflows with TCP SYN messages because now the queue capacity is insufficient to keep the information of new TCP connection requests. In this situation web server starts silently dropping new connection requests and displays the message like “Server is busy”, “The Connection request has timed out”, “Server is taking too long to respond” or “Connection Refused” on the web

browser. As a consequence, the clients retransmit the connection requests after the TCP timeout period. This causes the TCP connection failure and increases the connection time of the clients with the server. This situation is needed to be handled properly as it can lead to frustration and dissatisfaction among the users.

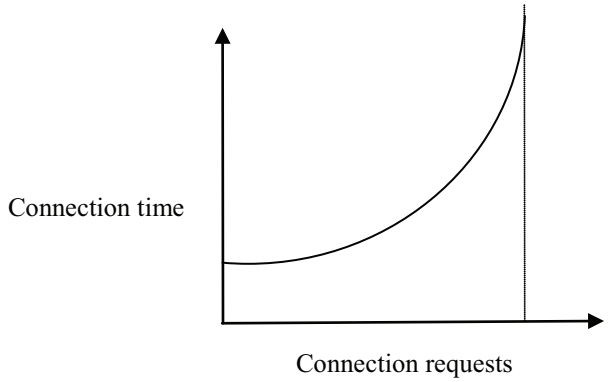


Figure 2. Number of Connection Requests versus Connection Time

The above graph shows the relationship between the number of connection request and connection time. The connection time is at minimum when the connection request on the servers is much less than its capacity. This time is composed of propagation time and processing time, both of which are negligible. However, when the number of connection request reaches the servers capacity, the connection time increases sharply [1].

In the existing techniques the connection requests are distributed in a dispatcher based web server cluster without considering the capacity of the socket accept queue of the backend servers. The connection requests are randomly transferred by the dispatcher to any available web server and it may cause connection failure and this increase the connection time of the clients with the server.

III. PROPOSED WORK

To solve the problem of TCP connection failure in the web server cluster, we measure the remaining capacity of the socket accept queue of the backend servers and distribute service requests based on their remaining capacity.

A. Theory-

Kernel allocates buffer space for socket accept queue. Kernel enforces maximum default buffer capacity. The amount of the buffer allocated for the socket depends upon the operating system. Small capacity socket queue is one of the major causes of poor web server performance. The kernel maintains following two queues in RAM for accepting connection requests [3, 8, 10]-

1. An incomplete connection queue, which contains an entry for each SYN that has arrived from a client for which the server is awaiting completion of the TCP three-way handshake.
2. A completed connection queue, which contains an entry for each client with whom the TCP three-way handshake has completed.

The connection creation mechanism is completely automatic; the server process is not involved. When a SYN arrives from a client, TCP creates a new entry on the incomplete queue and then responds with the second segment of the three-way handshake i.e. the server SYN with an ACK of the clients SYN. This entry will remain on the incomplete queue until the third segment of the three-way handshake arrives (the clients ACK of the server SYN), or until the entry times out. If the three-way handshake completes normally, the entry moves from the incomplete queue to the end of the completed queue. After this data transfer takes place between the client and the server [6].

B. Proposed algorithm used for eliminating tcp connection establishment failure –

This section describes a technique used for eliminating TCP connection failure in a web server cluster. Dispatcher performs the task of connection requests distribution. It distributes the connection requests by considering the capacity of the socket queue of the backend servers. Here we considered that web server cluster consist of two backend servers.

Implementation steps –

1. Sending the connection request to the backend server 1, till the load on the buffer does not exceed its capacity.
2. Traffic is transferred to the backend server 2, when the load on the server 1 reaches to its maximum capacity. During this time server 1 is busy in serving clients which are connected with it in step 1. Now server 2 is busy in serving clients which get connected with it in step 2.
3. Repeat step 1 and step 2 as long as both the backend servers work properly.

This approach eliminates the chance of TCP connection failure and reduces the connection time of the clients with the server, because here we distribute the connection requests by keeping track of the capacity of the backend servers.

Determining socket accept queue capacity -

The maximum number of TCP connection that can be opened simultaneous depends upon operating system [3].

For Windows server 2003 the maximum number of TCP connection that can be opened simultaneously is 16,777,214. Its buffer starts overflow if number of connection request exceeds this value.

[HKEY_LOCAL_MACHINE\System\CurrentControlSet \Services \Tcpip \Parameters]

TcpNumConnections=0X00fffffe (16,777,214).

For Windows-XP the maximum TCP connection that can be opened simultaneously is 61832.

For Linux 2.4 the maximum number of TCP connection that can be simultaneously opened is 18,888,162.

The main advantage of the above approach is that, dispatcher server keeps the tracks of the capacity of the socket accept queue of each of the backend servers. If one server is overloaded, dispatcher starts transferring connection requests to the next server and thus connection time of the client with the server reduces. Another advantage is that it reduces the traffic in the network as well as on the backend servers.

IV. EVALUATION

The effectiveness of the proposed approach was evaluated through an experiment performed on the network. The network is configured with two servers PC present in the cluster acting as backend servers, dispatcher PC, and a client PC. Dispatcher performs the task of distribution of connection requests on the basis of the socket accept queue capacity of the backend servers. Httperf tool was executed on client PC to generate http request and measures average connection time [5, 7, 9]. The proposed approach was executed on the Dispatcher.

The average connection time was measured by httperf running on the client machine, for the following two cases:

1. When connections requests are randomly distributed without considering socket accept queue capacity of the backend servers.
2. For the proposed approach in which connection requests is distributed by considering the capacity of the backend servers.

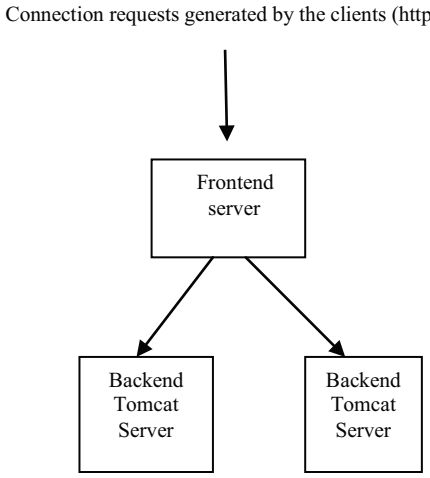


Figure 3. Network configuration for the proposed approach

The average connection time for this setting was measured by httperf, for both the case. It is measured for different number of TCP connection request generated by httperf.

Table -1 Experiment Result for Average Connection Time

Number of Connection Requests	Average Connection Time in ms (Random distribution Without considering servers capacity)	Average Connection Time in ms (proposed approach)
8000	24.1	17.2
10000	25.8	19.3
12000	28.9	21.8
14000	30.6	25.9
16000	32.3	28.1

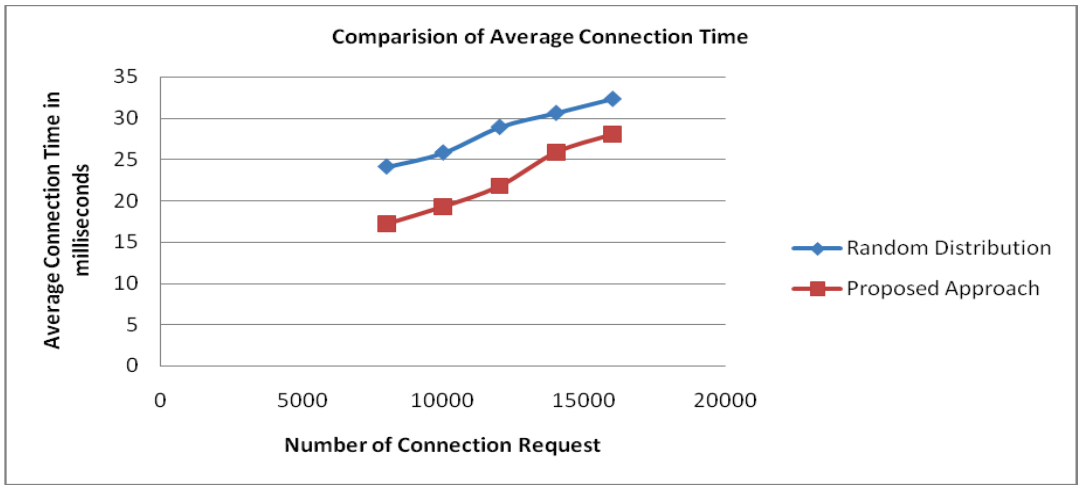


Figure 4. Shows that the average connection time is reduced if we distribute traffic by considering the remaining capacity of the backend servers.

V. IMPLEMENTATION

Apache 2.2 and Tomcat 6.0.32 web server has been used for the cluster creation. The language used for the implementation is Java. The experiment was performed on Intel Pentium(R) 4CPU 3.000 GHZ processors machines having Linux OS and 512 MB RAM.

VI. CONCLUSIONS

This paper presented a technique for eliminating the TCP connections failure in a web server cluster. Existing techniques distribute connection requests either randomly or in a round robin manner without considering the capacity of the backend server and results server overloading that is the major case of TCP connection failure. Experimental results show that average connection time is reduced by 78.74%, for the proposed approach.

REFERENCES

[1] H. Bryhni, E. Klovning, and Kure, "A comparison of load balancing techniques for scalable Web servers," IEEE Network, 14, 4 pp.58-64, July/Aug. 2000.

[2] WWW Server Load Balancing Technique Based on Passive Performance MeasurementSatoru Ohta and Ryuichi Andou.

[3] Self-prevention of socket buffer overflows Jin-Hee Choi, Young-Pil Kim, Chuck Yoo.

[4] T. Bourke, Server load balancing, O'Reilly, Aug. 2001.

[5] D. Mosberger and T. Jin, "httperf – a tool for measuring web server performance," ACM SIGMETRICS Performance Evaluation Review, 26, 3, pp.31-37, Dec. 1998.

[6] The TCP Split Handshake: Practical Effects on Modern Network Equipment Tod Beardsley (Corresponding author).

[7] httperf—A Tool for Measuring Web Server Performanc David Mosberger.

[8] Gaurav Banga and Peter Druschel. Measuring the capacity of a web server. In USENIX Symposium on Internet Technologies and Systems, pages 61–71, Monterey, CA, December 1997.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol– HTTP/1.1. Internet Engineering Task Force, January 1997.

[10] Rai Jain. The Art of Computer Systems Performance Analysis. John Wiley & Sons, New York, NY, 1991.

[11] C.-h. Tsai, K.G. Shin, J. Reumann and S. Singhal, "Online web cluster capacity estimation," IM2005, Application Sessions, Session 5, Nice, May 2005.

[12] Emiliano casalicchio, Michele Colajini "A Client-Aware Dispatching Algorithm For Web Cluster Providing Multiple Services".