Software Enabled Load Balancer by Introducing the Concept of Sub Servers

Mamta Dhanda, Parul Gupta

Abstract: In computer networking, load balancing is a technique to spread work between two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, maximize throughput, and minimize response time. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy. The balancing service is usually provided by a dedicated program or hardware device (such as a multilayer switch). One of the most common applications of load balancing is to provide a single Internet service from multiple servers, sometimes known as a server farm. Commonly load-balanced systems include popular web sites, large Internet Relay Chat networks, high-bandwidth File Transfer Protocol sites, NNTP servers and DNS servers. The load balancing system is a set of substitute buffer to share the server load, when their load exceeds its limit. The proposed technique gives an effective way to overcome the load balancing problem. Serving to more number of client requests is the main aim of every web server, but due to some unexpected load, the server performance may degrade In this paper we proposes a software enabled load balancing model by introducing the concept of sub servers for regional services to overcome the overhead of the main server.

Index Terms: FTP, NNTP Servers, DNS Servers, SLB.

I. INTRODUCTION

Apache Traffic Server is an Open Source project, originally developed by Inktomi, as a commercial product and later donated by Yahoo! as Apache Software Foundation. Apache Traffic Server was accepted as a high-Level Project in April of 2010. Yahoo! has used the original Traffic Server software, serving HTTP requests for many types of applications:

- Serving static content for all of Yahoo's web sites, as a Content delivery Network.
- As an alternative to Hardware Server Load Balancers.
- For connection management across long distances and providing low-latency connectivity to user.

For several years, Traffic Server already has been a critical component of Yahoos! Network. By releasing Traffic Server to the Open Source Community, a new tool is now readily available for anyone to use.

A. Reason behind Apache Software Foundation

It is necessary to understand why Yahoo! chose ASF, and what benefits we derive from being an ASF high-Level Project. We does not focus on Yahoo!'s decision to open-source Traffic Server.

Being part of an already established and well-functioning Open Source community brings immediate benefits to the project:

- There is large number of existing source code, skills and experiences in the ASF community, which we can easily understand.
- We are part of a reputable and well-maintained Open Source community
- We benefit from the many years of experience of ASF leadership in Open Source technology.
- We immediately gained new contributors to the project.

B. Traffic server Technologies

Apache Traffic Server is different from most existing Open Source proxy servers. It combines two technologies for writing applications that deal with high concurrency:

i. Asynchronous event processing
ii. Multi-threading

Traffic Server can draw the benefits from by combining these two technologies. It sometimes makes the technology and code complex, and sometimes difficult to understand. This is a serious drawback, but we feel the positives outweigh the negatives. Before we discuss the advantages and the disadvantages of this decision, we will give a brief introduction to these two concepts

a. Asynchronous Event Processing

It is also a combination of two concepts:

i. An event loop
ii. Asynchronous I/O

Combination of both the terms is called as Asynchronous Event processing. The event loop will schedule event handlers to be executed as the events trigger. The asynchronous requirement means that such handlers are not allowed to block execution waiting for I/O or block it for any other reason.

Revised Manuscript Received on 30 June 2012
*Correspondence Author(s)

Mamta Dhanda*, Assistant Professor, Department of Computer Science and Engg., JMIT Radaur, Yamuna Nagar, Haryana, India.
Scholar, Parul Gupta, Department of Computer Science and Engg., JMIT Radaur, Yamuna Nagar, Haryana, India.

The event handler must continue his execution, and inform the event loop that it should continue execution when the task would not block, instead of blocking. Events are also automatically generated and dispatched appropriately, as sockets and other file descriptors change state and become ready for reading or writing or possibly both. It is important to understand that an event loop model does not necessarily require all I/O to be asynchronous. However, this is a fundamental design requirement in the Traffic Server case, and it impacts not only how the core code is written, but also how you implement plug-ins. A plug-ins could not block any I/O calls, but as doing so they would prevent the asynchronous event processor or scheduler from functioning properly.

b. *Multi-Threading*

Multithreading is mechanism to allow a process to split itself into two or more concurrently running tasks. These all threads exist within the same context of a single process. Different operating systems implement multi-threading in different ways. A fundamental difference between creating a thread and creating a new process is that threads are allowed to share resources not commonly shared between separate processes. It is typically much less expensive for an OS to switch execution between threads than between processes.

Every operating system has limitations on how many threads it can handle. Even though switching threads is lightweight, it still has overhead and consumes CPU. Threads also consume some extra memory, but of course, not as much as individual processes will. Threading is a simpler abstraction of concurrency than the asynchronous event processing.

c. *Flaws of both above technologies*

Why Traffic Server decided to use both the technologies i.e. Asynchronous event processing and Multithreading. This is an important discussion because it will help you decide which intermediary solutions you should choose and now we have a basic understanding of these concurrency mechanisms provide.

Multi-threading is a famous method for solving concurrency issues because it is easily understand and is a proven technology. It solves the concurrency problem .It is also well-supported on most modern Operating Systems but it does have a few problems and concerns, such as:

- They still create context switches in the Operating System even though threads are lightweight. Each thread also requires its own “private” data, particularly on the stack. As such, the more threads you have, the more context switches you will see, and memory consumption will increase linearly as the number of threads increases.
- If the application is to take advantage of shared memory then writing multi-threaded applications is difficult.
- Lock contention, deadlocks, priority inversion and race conditions are some of the difficulties with which developers will need to confront.

Generally it is easier to write a program for asynchronous event loops and there are many abstractions and libraries available that provide good APIs. Some examples include libevent[2] and libev[3] for C and C++ developers. There are also bindings for many higher-level languages for both these libraries, and others. But there are a few limitations with event loops:

- To avoid blocking the event loop, all I/O needs to be asynchronous. This makes it slightly more difficult for programmers, which may be synchronous by nature
- The event loop and handler supports for running on a single CPU.
- If the event loop needs to deal with a large number of events, increased latency can occur before an event is processed (by the nature of the events being queued).

.Traffic Server decided to combine both of these techniques, thus eliminating many of the issues and limitations associated with each of them. In Traffic Server, there are a small number of “worker threads”; each such worker thread is running its own asynchronous event processor. In a typical setup, this means Traffic Server will run with around 20-40 threads only. This is configurable, but increasing the number of threads above the default (which is 3 threads per CPU core) will yield worse performance due to the overhead caused by more threads.

C. *Content delivery Networks*

A content delivery network (CDN) is a system of distributed servers (network) that deliver web pages and other Web content to a user based on the geographic locations of the user, the origin of the webpage and a content delivery server. A Content Delivery Network, is a service used to deliver certain types of HTTP content. This content is usually static by nature. Examples of CDN-type content are JavaScript, CSS, and all types of images and other static media content. Serving such content out of a caching HTTP intermediary makes deployment and management significantly easier, since the content distribution is automatic.

A CDN automates content distribution to many collocations, simplifying the operational tasks and costs.

To improve end-user experience, a CDN is commonly deployed on the Edge networks, assuring that the content is as close as possible to the users.
Reason for using this network:

- Pages are loaded very fast
- Less expensive
- More effective utilization of resources
- Redundancy and resilience to network outages.

The biggest question we face when deciding on a CDN is whether to buy it as a service from one of the many commercial CDN vendors or to build it yourself. Every time, we are probably better off buying CDN services initially. There are initial costs associated with setting up your own private CDN on the Edge, and this should be considered when doing these evaluations.
To consider the above limitations, we are strong in favour of building your own CDN, particularly if your traffic is large enough that the costs of buying the services from a CDN vendor are considerable. Further building a CDN is not rocket science. Any organization with a good infrastructure and operations team can easily do it. All you need is to configure and deploy a small number of servers running as reverse proxy servers for HTTP and sometimes HTTPS.

a. Creating a Content Delivery Network with Apache Traffic Server

Apache Traffic Server is an excellent choice for building your own CDN. First, it has a feature of scalability in case on a large number of CPUs, and well beyond Gigabit network cards. Additionally, the technology behind Traffic Server is well-geared toward a CDN:

- The server is easy to deploy and manage as a reverse proxy server.
- The configuration tasks and changes can be done on live systems, and never require server restarts.
- The Traffic Server cache is fast and scales very well.
- It is also very resilient to corruptions and crashes.
- It scales well for a large number of concurrent connections, and supports all necessary HTTP/1.1 protocol features such as SSL and Keep-Alive.

Traffic Server has already proved that it delivers over 350,000 requests/second and over 30Gbps in the Yahoo! CDN alone. With over 150 servers deployed worldwide, this is an unusually large private CDN. Most setups will be much smaller. Of course, many of the other existing HTTP caches can be used to build a CDN. We believe Traffic Server is a serious contender in this area, but there is healthy competition.

b. Configuration

To configure Apache Traffic Server for building your CDN, there are primarily two configuration files relevant for setting up Traffic Server as a caching intermediary:

- remap.config – This file is empty by default, holds the mapping rules so that Traffic Server can function as a reverse proxy
- records.config – This file holds a number of key-value pair, and in most situations the defaults are good enough.

Of course, there can be much more complex configurations, particularly in the remap configuration, but the examples demonstrate how little configuration would be required to get a functional CDN with almost zero configuration using Apache Traffic Server.

D. Connection management with Apache Traffic Serve

The purpose of connection management service is to reduce latency for the end-user. Connection management is very similar to a CDN; in fact, many CDN vendors also provide such services as well. Living on the Edge, the connection management service can effectively fight two enemies of web performance:

- TCP 3-way handshake: the latency introduced by the handshake is reduced. Allowing for long-lived Keep-Alive connections can eliminate such latency entirely.
- TCP congestion control for example “Slow Start”: The more user is farther away from the server, the more visible the congestion control mechanisms become. Being on the Edge, users will always connect to an HTTP server i.e. an Origin Server or another intermediary that is close.

.

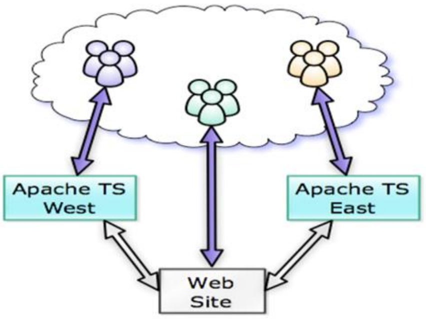

Fig 1. Connection management

The above drawn figure shows that in various areas of the world, how user connect to different servers. Some users might connect directly to the HTTP web server while others might connect to an intermediary server that is close to the user.

Connections between the intermediaries (the connection managers) and Origin Servers ("web site") are long-lived, thanks to HTTP Keep-Alive. Reducing the distance between a user and the server, as well as eliminating many new TCP connections, will reduce page-load times significantly. In some cases, we've measured up to 1 second or more reduction in first page-load time, only by introducing the connection manager intermediaries.

E. Http proxy

Proxy server is a server acts as an intermediary for requests from client seeking resources from other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource available from a different server. The proxy server evaluates the request as a way to simplify and control their complexity.

HTTP proxy servers, with or without caching, are implementations of an HTTP server, act as an intermediary between a client (User-Agent), and another HTTP server mainly referred as an Origin Server. It's quite possible to have multiple intermediaries in a hierarchy, and many ISPs will proxy all HTTP requests through a mandatory intermediary.

There are three main types of a proxy server:

- Forward Proxy-Forward proxies are proxies where the client server names the target server to connect to .It requires user agents(e.g. browsers) to be configured and aware of the proxy server
- Reverse Proxy – A reverse proxy (or surrogate) is a proxy server that appears to clients to be an ordinary server. Requests are forwarded to one or more origin servers which handle the request. The response is returned as if it came directly from the proxy server.
- Intercepting Proxy – This is similar to Forward Proxy, except the intermediary intercepts the HTTP requests from the User-Agent. This is also typically done by ISPs or corporate firewalls, but has the advantage that it is transparent to the user. This usually is also referred to as Transparent Proxy.

Any HTTP intermediary must of course function as a basic HTTP web server. There is definite overlap in functionality between a proxy server and a regular HTTP server. Both typically provide support for access control, SSL termination and IPv6. In addition, many HTTP intermediaries also provide features such as:

- Act as a firewall to access HTTP content: by providing content filtering, anti-spam filtering, audit logs, etc
- Finding the most appropriate Origin Server or another intermediary from which to fetch the document based on the incoming request;
- Cache documents locally, for faster access and less load on Origin Servers
- Server Load Balancing (SLB), by providing features such as sticky sessions, URL-based routing, etc
- Providing infrastructure to build redundant and resilient HTTP services;
- Implementing various Edge services, such as Edge Side Includes (ESI);

Traffic Server can perform many of these tasks, but not all of them obviously. Some tasks would require development of plug-in and some tasks would require changes to the internals of the code. Fortunately, Traffic Server, similar to Apache HTTPD, has a feature-rich plug-in API to develop extensions. We aimed to improve and extend the plug-in APIs to allow for even more complex development. Therefore efforts are being not made to release only a useful number of useful plug-in to the Open Source community. We are also starting to see the community contribute new Traffic Server plug-in.

F. Http server load balancer

Load balancing is a set of techniques that configure the servers and other hardware in such a way that the workload is distributed equally amongst the servers, thereby reducing the dependency on any single server and making it easy to add resources in such a way that performance is scaled according to the resources added. It is s a basic technique for routing traffic, such as HTTP requests, to a server in a way that achieves optimal performance, high availability, or easier service implementation. Hardware Server Load Balancer can handle any TCP and UDP protocols, while an HTTP specific SLB would obviously only do HTTP and perhaps HTTPS. With an HTTP Server Load Balancer, we can assure that:

- There is always at least one real-server available to serve any type of request
- a particular user always hits the same backend real server
- a particular URL is served by the same backend real server

Getting users, or requests, associated with a smaller number of servers can significantly improve the performance of your applications. You can see better cache affinity, smaller active data sets, and easier (and faster) code to evaluate. The follow picture depicts a typical HTTP Server Load Balancer setup:

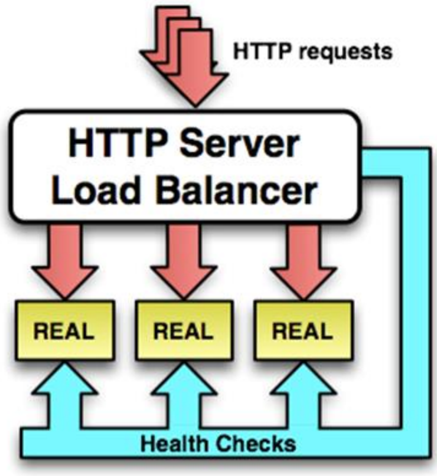

Fig 2. Server Load Balancer

Unfortunately, this is an area where Traffic Server is currently behind the curve, and we openly admit it. Basic Load Balancing can be done with the configurations available, but anything advanced will require coding for a custom plug-in. There is some hope that an additional piece of technology can be open-sourced, but there is no ETA at this time. So why we are talking about this at all? Well, it is an important feature that is somewhat lacking in Apache Traffic Server, and our hope is that discussing this openly will attract attention and interest from other developers who would like to work on these features.

II.PROPOSED ARCHITECTURE

A. Exiting system

In existing system they are not using sub servers for centralized server so there is a chance to occur time delay in message delivering and packet discarding and complexity in receiving more messages these are the main draw backs available in existing system.

B. Proposed System

In our proposed system we are overcoming the draw backs available in existing system like time delay in message delivering and packet discarding and we can increase the number of messages to be delivered to the destination. These are main advantages of our proposed system.

For Internet services, the load balancer is usually a software program which is listening on the port where external clients connect to access services. The load balancer forwards requests to one of the "backend" servers, which usually replies to the load balancer. This allows the load balancer to reply to the client without the client ever knowing about the internal separation of functions. It also prevents clients from contacting backend servers directly, which may have security benefits by hiding the structure of the internal network and preventing attacks on the kernel's network stack or unrelated services running on other ports.

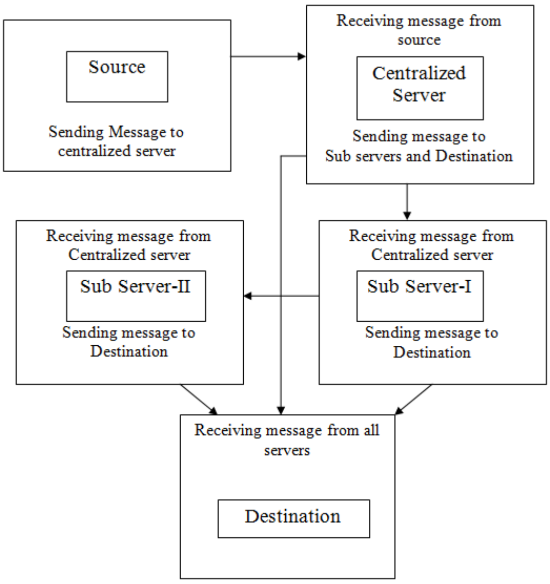
Source
Sending Message to centralized server
Receiving message from source
Centralized Server
Sending message to Sub servers and Destination
Receiving message from Centralized server
Sub Server-II
Sending message to Destination
Receiving message from Centralized server
Sub Server-I
Sending message to Destination
Receiving message from all servers
Destination

Fig 3. Proposed Model with sub-servers

III. PROPOSED METHOLOGY

This paper focuses on the concept of software enabled load balancer using java platform. We will simulate the java code using simulator to get better results considering following parameters mentioned below:

- Load balancing
- Message distribution

A. Load Balancing

In this module we have checking the availability of the server space and its sub server's space before receiving the message. If there is no space to receive message then intimate to the source no more space in server to receive message.

B. Message Distribution

Centralized server receives message from source it checks its own availability if it is exceed then send message to the sub servers before send the message it checks sub server's availability too if it is exceed then inform to the source no more space to receive message.

REFERENCES

1. G. Huang, W. Wang, T. Liu, H. Mei, "Simulation-based analysis of middleware service impact on system reliability: Experiment on Java application server", Journal of Systems and Software, Volume 84, Issue 7, Pages 1160-1170, ISSN 0164-1212, July 2011.
2. J. Guitart, D. Carrera, V. Beltran, J. Torres, E. Ayguade, "Designing an overload control strategy for secure e-commerce applications", Computer Networks, Volume 51, Issue 15, 24, Pages 4492-4510,October 2007.
3. K. Birman, R. van Renesse, W. Vogels, "Adding high availability and autonomic behavior to Web services," Software Engineering, 2004.ICSE 2004. Proceedings. 26th International Conference on , vol., no., pp. 17- 26, 23-28 May 2004.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

4. A. Guruge, Java and Web Services, Web Services, Digital Press, Burlington, 2004, Pages 227-270, ISBN 978-1-55-558282-1, DOI:10.1016/B978-155558282-1/50008-7.
5. H. Xiaotao, D. Chaozhi, “Design of high-available architecture for distributed application based on J2EE and its analysis[JA]. Huazhong Keji Daxue Xuebao (Ziran Kexue Ban)/Journal of Huazhong University of Science and Technology (Natural Science Edition).2005,44-47.
6. Y. Liu, L. Wang, S. Li, "Research on self-adaptive load balancing in EJB clustering system," Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on , vol.1, no., pp.1388-1392, 17-19 Nov. 2008.
7. T. Bourke, T. Server Load Balancing. O’Reilly & Associates Press. 2001.
8. V.Viswanathan. Load Balancing Web Applications. OnJava.com.2001. http://onjava.com/pub/a/onjava/2001/09/26/loa d.html
9. G. Lodi, F. Panzieri,, D. Rossi, E. Turrini, "Experimental Evaluation of a QoS-aware Application Server," Network Computing and Applications, Fourth IEEE International Symposium on , vol., no., pp.259-262, 27-29 July 2005.
10. Apache Software Foundation. The Apache Tomcat ConnectorReferenceGuide.2010.http://tomcat.apache.org/connectorsdoc/ reference/workers.html
11. K. Gilly de la Sierra-Llamazares. Tesis: An adaptive admission control and load balancing algorithm for a QoS-aware Web system. Universitat de les Illes Balears. 2009. http://www.tesisenxarxa.net/TDX-1211109-113725/index_cs.html
12. H. Elmeleegy, N. Adly, and M. Nagi. “Adaptive Cache-Driven Request Distribution in Clustered EJB Systems”. Proceedings of the Tenth International Conference on parallel and Distributed Systems (ICPADS’04), 179-186, 2004,
13. R. Johnson,. Expert One-on-One J2EE Design and Development. Wrox Press. 2003.
14. R. B’Far, Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML. Cambridge University Press. 2005.

AUTHOR PROFILE

Mrs. Mamta Dhanda is working as Assistant Professor, department of Computer Science &Engg in JMIT, Radaur, Haryana. She has completed her B.sc in 2003 and M.Sc in 2005 in Computer Science from Kurukshetra University and then completed her M.Tech in Computer Science & Engg in 2007.she presented her paper in RIET national conference.

Ms.Parul Gupta received a bachelor degree in computer science in 2009 form haryana engineering conllege affiliated to kurukshetra university.she is currently doing master degree in computer science and engineering form JMIT Radaur haryana.Her area of interest are cloud computing,wireless networks.