RESEARCH ARTICLE OPEN ACCESS

Investigating on Web Server Load Balancing Using SSL Back-End Forwarding Method

Rayan Soqati

Department Of Computer Science and Engineering

ABSTRACT

The cluster-based data centres consist of three tiers (Web server, application server, and database server) are being used to host complex Web services such as e-commerce applications. The application server handles dynamic and sensitive Web contents that need protection from eavesdropping, tampering, and forgery. Although the Secure Sockets Layer (SSL) is the most popular protocol to provide a secure channel between a client and a cluster-based network server, its high overhead degrades the server performance considerably and, thus, affects the server scalability. Therefore, improving the performance of SSL-enabled network servers is critical for designing scalable and high-performance data centres. To improve the performance of application servers, the proposed back-end forwarding scheme can further enhance the performance due to better load balancing. The SSL backend forward scheme can minimize the average latency by about 40 percent and improve throughput across a variety of workloads.

Keywords: - Secure Socket Layer, Web Clusters, Load balancing, Protection from eavesdropping

I. INTRODUCTION

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a *server farm* or *server pool*.Server load balancing provides scalability and high availability for applications, Web sites and cloud services by monitoring the health of servers, evenly distributing loads across servers and maintaining session persistence and a seamless user experience in the event that one or more servers become overburdened or unresponsive.

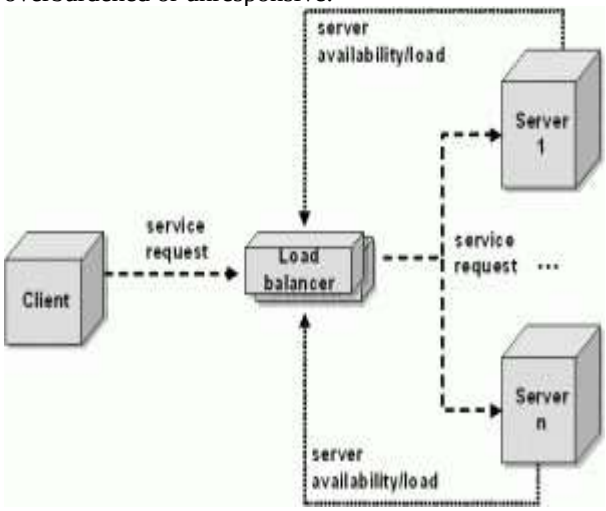

Fig1. Classic load balancer architecture (load dispatcher)

Load balancing is a staple solution in virtually every data centre. However, today's application delivery controllers (ADCs) represent a considerable evolution from simple server load balancing methods.A load balancer acts as the "traffic cop" sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance as shown in Fig 1. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it. In this manner, a load balancer performs the following functions:

- Distributes client requests or network load efficiently across multiple servers
- Ensures high availability and reliability by sending requests only to servers that are online
  - Provides the flexibility to add or subtract servers as demand dictates

To reach high availability, the load balancer must monitor the servers to avoid forwarding requests to overloaded or dead servers. Several different load balancing methods are available to choose from. When working with servers that differ significantly in processing speed and memory, one might want to use a method such as Ratio or Weighted Least Connections.
Load balancing calculations can be localized to each pool (member-based calculation) or they may apply to

all pools of which a server is a member (node-based calculation).

II. SERVER LOAD BALANCING TECHNIQUES

2.1 Round Robin

This is the default load balancing method. Round Robin mode passes each new connection request to the next server in line, eventually distributing connections evenly across the array of machines being load balanced.

Usage:
Round Robin mode works well in most configurations, especially if the equipment that you are load balancing is roughly equal in processing speed and memory.

2.2 Ratio (member) and Ratio (Node)

The BIG-IP system distributes connections among pool members or nodes in a static rotation according to ratio weights that you define. In this case, the number of connections that each system receives over time is proportionate to the ratio weight you defined for each pool member or node. You set a ratio weight when you create each pool member or node.

Usage:
These are static load balancing methods, basing distribution on user-specified ratio weights that are proportional to the capacity of the servers.

2.3 Dynamic Ratio (member) Dynamic Ratio (node)

The Dynamic Ratio methods select a server based on various aspects of real-time server performance analysis. These methods are similar to the Ratio methods, except that with Dynamic Ratio methods, the ratio weights are system-generated, and the values of the ratio weights are not static. These methods are based on continuous monitoring of the servers, and the ratio weights are therefore continually changing.

Usage:
The Dynamic Ratio methods are used specifically for load balancing traffic to Real Networks Real System Server platforms, Windows platforms equipped with Windows Management Instrumentation (WMI), or any server equipped with an SNMP agent such as the UC Davis SNMP agent or Windows 2000 Server SNMP agent

2.4 Fastest (node) Fastest (Application)

The Fastest methods select a server based on the least number of current sessions. The following rules apply to the fastest load balancing methods:

- These methods require that you assign both a Layer 7 and a TCP type of profile to the virtual server.
- If a Layer 7 profile is not configured, the virtual server falls back to Least Connections load balancing mode.

Usage:
The Fastest methods are useful in environments where nodes are distributed across separate logical networks.

2.5 Least Connections (member) Least Connections (node)

The Least Connections methods are relatively simple in that the BIG-IP system passes a new connection to the pool member or node that has the least number of active connections.
Note: If the One Connect feature is enabled, the Least Connections methods do not include idle connections in the calculations when selecting a pool member or node. The Least Connections methods use only active connections in their calculations.

Usage:
The Least Connections methods function best in environments where the servers have similar capabilities. Otherwise, some amount of latency can occur.

For example, consider the case where a pool has two servers of differing capacities, A and B. Server A has 95 active connections with a connection limit of 100, while server B has 96 active connections with a much larger connection limit of 500. In this case, the Least Connections method selects server A, the server with the lowest number of active connections, even though the server is close to reaching capacity. If you have servers with varying capacities, consider using the Weighted Least Connections methods instead.

2.6 Weighted Least Connections (member) Weighted Least Connections (node)

Like the Least Connections methods, these load balancing methods select pool members or nodes based on the number of active connections. However, the

Weighted Least Connections methods also base their selections on server capacity.

The Weighted Least Connections (member) method specifies that the system uses the value you specify in Connection Limit to establish a proportional algorithm for each pool member. The system bases the load balancing decision on that proportion and the number of current connections to that pool member. For example, member_a has 20 connections and its connection limit is 100, so it is at 20% of capacity. Similarly, member_b has 20 connections and its connection limit is 200, so it is at 10% of capacity. In this case, the system select selects member_b. This algorithm requires all pool members to have a non-zero connection limit specified. The Weighted Least Connections (node) method specifies that the system uses the value you specify in the node's Connection Limit setting and the number of current connections to a node to establish a proportional algorithm. This algorithm requires all nodes used by pool members to have a non-zero connection limit specified. If all servers have equal capacity, these load balancing methods behave in the same way as the Least Connections methods.

Note: If the One Connect feature is enabled, the Weighted Least Connections methods do not include idle connections in the calculations when selecting a pool member or node. The Weighted Least Connections methods use only active connections in their calculations.

Usage:

Weighted Least Connections methods work best in environments where the servers have differing capacities. For example, if two servers have the same number of active connections but one server has more capacity than the other, the BIG-IP system calculates the percentage of capacity being used on each server and uses that percentage in its calculations.

2.7 Observed (member) Observed (node)

With the Observed methods, nodes are ranked based on the number of connections. The Observed methods track the number of Layer 4 connections to each node over time and creates a ratio for load balancing. The need for the Observed methods is rare, and they are not recommended for large pools.

2.8 Predictive (member) Predictive (node)

The Predictive methods use the ranking methods used by the Observed methods, where servers are rated according to the number of current connections. However, with the Predictive methods, the BIG-IP system analyzes the trend of the ranking over time, determining whether a nodes performance is currently improving or declining. The servers with performance rankings that are currently improving, rather than declining, receive a higher proportion of the connections. The need for the Predictive methods is rare, and they are not recommended for large pools.

2.9 Least Sessions

The Least Sessions method selects the server that currently has the least number of entries in the persistence table. Use of this load balancing method requires that the virtual server reference a type of profile that tracks persistence connections, such as the Source Address Affinity or Universal profile type.

The Least Sessions method works best in environments where the servers or other equipment
Which the user is load balancing have similar capabilities.

2.10 L3 Address

This method functions in the same way as the Least Connections methods. It is not recommended for large pools and incompatible with cookie persistence.

III. PROBLEM ISSUES

Usually at this point, a problem arises like how does a load balancer decide which host to send the connection to? And what happens if the selected host is not working? If the selected host is not working it doesn't respond to the client request and the connection attempt eventually times out and fails. This is obviously not a preferred circumstance, as it doesn't ensure high availability. That's why most load balancing technology includes some level of health monitoring that determines whether a host is actually available before attempting to send connections to it. There are multiple levels of health monitoring, each with increasing granularity and focus. A basic monitor would simply PING the host itself. If the host does not respond to PING, it is a good assumption that any services defined on the host are probably down and should be removed from the cluster of available services. Unfortunately,

even if the host responds to PING, it doesn't necessarily mean the service itself is working. Therefore most devices can do "service PINGs" of some kind, ranging from simple TCP connections all the way to interacting with the application via a scripted or intelligent interaction. These higher-level health monitors not only provide greater confidence in the availability of the actual services (as opposed to the host), but they also allow the load balancer to differentiate between multiple services on a single host. The load balancer understands that while one service might be unavailable, other services on the same host might be working just fine and should still be considered as valid destinations for user traffic. While load balancer decides which host to send a connection request, each virtual server has a specific dedicated cluster of services (listing the hosts that offer that service) which makes up the list of possibilities as shown in Fig 2. Additionally, the health monitoring modifies that list to make a list of "currently available" hosts that provide the indicated service. It is this modified list from which the load balancer chooses the host that will receive a new connection.

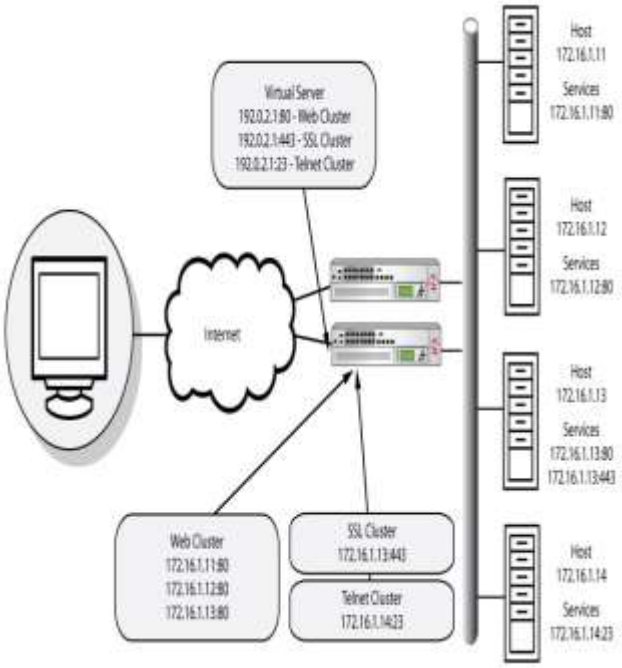

Fig 2: Load balancing comprises four basic concepts-virtual servers, clusters, services and hosts

Deciding the exact host depends on the load balancing algorithm associated with that particular cluster. The most common is simple round-robin where the load balancer simply goes down the list starting at the top and allocates each new connection to the next host; when it reaches the bottom of the list, it simply starts again at the top. While this is simple and very predictable, it assumes that all connections will have a similar load and duration on the back-end host, which is not always true. More advanced algorithms use things like current-connection counts, host utilization, and even real-world response times for existing traffic to the host in order to pick the most appropriate host from the available cluster services. Sufficiently advanced load balancing systems will also be able to synthesize health monitoring information with load balancing algorithms to include an understanding of service dependency. This is the case when a single host has multiple services, all of which are necessary to complete the user's request. A common example would be in e-commerce situations where a single host will provide both standard HTTP services (port 80) as well as HTTPS (SSL/TLS at port 443). In many of these circumstances, you don't want a user going to a host that has one service operational, but not the other. In other words, if the HTTPS services should fail on a host, you also want that host's HTTP service to be taken out of the cluster list of available services. This functionality is increasingly important as HTTP-like services become more differentiated with XML and scripting.

Connection maintenance

If the user is trying to utilize a long-lived TCP connection (telnet, FTP, and more) that doesn't immediately close, the load balancer must ensure that multiple data packets carried across that connection do not get load balanced to other available service hosts. This is connection maintenance and requires two key capabilities:

1) the ability to keep track of open connections and the host service they belong to; and 2) the ability to continue to monitor that connection so the connection table can be updated when the connection closes. This is rather standard fare for most load balancers.

Persistence

Increasingly more common, however, is when the client uses multiple short-lived TCP connections (for example, HTTP) to accomplish a single task. In some cases, like standard web browsing, it doesn't matter and each new request can go to any of the back-end service hosts; however, there are many more instances (XML, ecommerce "shopping cart," HTTPS, and so on) where it is extremely important that multiple connections from the same user go to the same back-end service host and not be load balanced. This concept is called persistence, or server affinity. There are multiple ways to address this depending on the protocol and the desired results. For example, in modern HTTP transactions, the server

can specify a "keep-alive" connection, which turns those multiple short-lived connections into a single long-lived connection that can be handled just like the other longlived connections. However, this provides little relief. Even worse, as the use of web services increases, keeping all of these connections open longer than necessary would strain the resources of the entire system. In these cases, most load balancers provide other mechanisms for creating artificial server affinity. One of the most basic forms of persistence is sourceaddress affinity. This involves simply recording the source IP address of incoming requests and the service host they were load balanced to, and making all future transaction go to the same host. This is also an easy way to deal with application dependency as it can be applied across all virtual servers and all services. In practice however, the wide-spread use of proxy servers on the Internet and internally in enterprise networks renders this form of persistence almost useless; in theory it works, but proxy-servers inherently hide many users behind a single IP address resulting in none of those users being load balanced after the first user's request— essentially nullifying the load balancing capability. Today, the intelligence of load balancer–based devices allows organizations to actually open up the data packets and create persistence tables for virtually anything within it. This enables them to use much more unique and identifiable information, such as user name, to maintain persistence. However, organizations one must take care to ensure that this identifiable client information will be present in every request made, as any packets without it will not be persisted and will be load balanced again, most likely breaking the application. Server load balancing is essential to keep resources properly distributed in a virtual infrastructure. If the infrastructure is expanding to a private cloud, which is an automated environment, virtual machine load balancing becomes even more critical.

With any virtualization platform, a private cloud requires virtual machines (VMs) that can live-migrate anywhere to balance resource loads. The most common load-balancing services are Microsoft System Center Virtual Machine Manager's Performance and Resource Optimization feature and VMware's Distributed Resource Scheduler (DRS).

Most virtualization administrators already rely on some degree of server load balancing in their infrastructure, so you're probably closer to private cloud computing than you may think.

But when server load balancing doesn't work correctly, a virtual infrastructure can suffer from painful performance problems. There will be a check box with a connected option next to the disk drives inside VM configuration screen to select the box unless you have disk data transferred to a VM.

But connecting the disk drive creates a dependency between a VM and the physical disk, which can in turn cause load balancing to fail. When disk drivers are not used, disconnect them, or server loads may not be balanced.

Affinity and anti-affinity

Affinity in the virtual world refers to how VMs can be configured to always (or never) collocate on the same virtual host. By configuring affinity rules, we prevent both domain controllers from residing on the same host and, if a host experiences a failure, both from going down.

VMware and Microsoft allow configuring VMs to follow (or not follow) one another as they live migrate. But user shouldn't use these features unless they're absolutely necessary, because affinity rules create dependencies between VMs that can affect server load balancing. It is advisable to steer clear of affinity unless if it is absolutely needed.

Resource restrictions

Resource restrictions protect virtual machines from others that overuse resources. One can limit the resources that a VM is allowed to consume. It can also reserve a minimum quantity of resources that a VM must always have available. Both settings are great when resources are tight, but they also create dependencies that can cause server load balancing to fail -- or make it more difficult for a load-balancing service to do its job.

Unnecessarily powerful VMs

This one's a rookie mistake. Most of us are used to the notion of nearly unlimited physical resources for Windows. It's been years since servers lacked the processing power or RAM to support a workload. The idea of "Just give it lots of RAM and plenty of processors" tends to seep into our virtual infrastructure as well.

The problem with this line of thinking is that unnecessarily powerful VMs consume lots of resources. When machines use too many processors or too much RAM, target host servers aren't powerful enough to

support the VM's configuration. As a result, the machine can't fail over or is limited to specific targets where it can fail over.

Start with one processor per virtual machine and as little RAM as possible, then work upward. That way server load-balancing service can allocate resources only where they're most needed -- and none go to waste.

Most of us remember that it's necessary to have storage for VM files themselves, but we sometimes forget about the other storage requirements: Raw Device Mappings for a VMware virtual machine or pass-through drives for a Hyper-V machine. Storage connections are always on a per-host basis, which means that every host must be correctly masked and zoned so VMs can see their storage. If not, server load balancing suffers, because VMs and their resources can't migrate to the target host.

Disabling load balancing

Some admins don't realize that VM load balancing is still considered an advanced capability. As a result, they haven't created a cluster in their vSphere data center or haven't enabled DRS.

For a Hyper-V infrastructure, both System Center Virtual Machine Manager and System Center Operations Manager are required for automated server load balancing to work.

My final and somewhat tongue-in-cheek recommendation: If we intend to use server load balancing, then capability should be turned on.

IV. RELATED WORK

Anoop Reddy [1] developed a system to protect applications from session stealing/hijacking attacks by tracking and blocking anomalies in end point characteristics. In this proposal Systems and methods for protection against session stealing is described. In embodiments of the present solution, a device intermediary to the client and the server may identify first properties of the client and associate the first properties with the session key. When the device receives subsequent request comprising the session key, the device matches the associated first properties with second properties of the second device that is sending the subsequent request. If there is a match, the subsequent request transmitted to the server. Otherwise, the subsequent request is rejected.

Dipesh Gupta, Hardeep Singh [2] proposed SSL session sharing based web cluster load balancing**.** Internet users increase the traffic on the servers and server security is the major concern with which the user's privacy needs to be protect. TLS (Transport Layer Security) is a widely deployed protocol that establishes a secure channel between communicating parties over the internet. But TLS/SSL has huge impact on webserver's performance by degrading it to a considerable amount. When TLS/SSL session is generated it is broadcasted to all servers in the cluster with which session reuse can be used to save time in negotiation. TLS Handshake and Session resume is occur at the server end so in future if client requests again and its session is not expired then it can again joins that its own session without renegotiating which saves the session initialization time. Ultimately a new load balancing cluster design is proposed that can share TLS sessions in the cluster to effectively improve the performance of TLS web cluster. The web cluster server shares the sessions of users within the cluster. The another technique for improving the latency and throughput of the server SSL/TLS with backend forwarding technique is compare and is analysed. The traditional method has flaws in the load balancing of the server but with the new implanted technique on the server improves the performance during the high load .The results are reviewed with 16 and 32 node cluster system. With new technique the latency of system has been decreased by the 40 % and throughput of the system is extremely better than classical balancing technique.

According to De Grande [3] dynamic balancing of computation and communication load is vital for the execution stability and performance of distributed, parallel simulations deployed on shared, unreliable resources of large-scale environments. High Level Architecture (HLA) based simulations can experience a decrease in performance due to imbalances that are produced initially and/or during run-time. These imbalances are generated by the dynamic load changes of distributed simulations or by unknown, non-managed background processes resulting from the non-dedication of shared resources. Due to the dynamic execution characteristics of elements that compose distributed simulation applications, the computational load and interaction dependencies of each simulation entity change during run-time. These dynamic changes lead to an irregular load and communication distribution, which increases overhead of resources and execution delays. A static partitioning of load is limited to deterministic applications and is incapable of predicting the dynamic changes caused by distributed applications or by external background processes. Due to the

relevance in dynamically balancing load for distributed simulations, many balancing approaches have been proposed in order to offer a sub-optimal balancing solution, but they are limited to certain simulation aspects, specific to determined applications, or unaware of HLA-based simulation characteristics. Therefore, schemes for balancing the communication and computational load during the execution of distributed simulations are devised, adopting a hierarchical architecture. First, in order to enable the development of such balancing schemes, a migration technique is also employed to perform reliable and low-latency simulation load transfers. Then, a centralized balancing scheme is designed; this scheme employs local and cluster monitoring mechanisms in order to observe the distributed load changes and identify imbalances, and it uses load reallocation policies to determine a distribution of load and minimize imbalances. As a measure to overcome the drawbacks of this scheme, such as bottlenecks, overheads, global synchronization, and single point of failure, a distributed redistribution algorithm is designed. Extensions of the distributed balancing scheme are also developed to improve the detection of and the reaction to load imbalances. These extensions introduce communication delay detection, migration latency awareness, self-adaptation, and load oscillation prediction in the load redistribution algorithm. Such developed balancing systems successfully improved the use of shared resources and increased distributed simulations' performance.

K Kungumaraj, T Ravichandran proposed A distributed system consists of independent workstations connected usually by a local area network. [4] Load balancing system puts forward to a new proposal to balance the server load in the distributed system. 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. To overcome these issues, network provides an efficient way to distribute their work with the sub servers which is also known as proxy servers. Allocating work to the sub server by their response time is the proposed technique. The secure socket layer with Load balancing scheme has been introduced to overcome those server load problems. Storing and serving effectively and securely is more important so that desired algorithm is going to implement for load distribution and security enhancement named as Secure Socket Layer with Load Balancing and RSA Security algorithm respectively. Calculating response time of each request from the clients has been done by sending an empty packet over the networking to all the sub servers and response time for each sub server is calculated using the Queuing theory. In this Load Balancing system, the SSL based load distribution schemes have been introduced for better performance.

In systems and methods for supporting a SNMP request over a cluster [5] the present disclosure is directed towards systems and methods for supporting Simple Network Management Protocol (SNMP) request operations over clustered networking devices. The system includes a cluster that includes a plurality of intermediary devices and an SNMP agent executing on a first intermediary device of the plurality of intermediary devices. The SNMP agent receives an SNMP GETNEXT request for an entity. Responsive to receipt of the SNMP GETNEXT request, the SNMP agent requests a next entity from each intermediary device of the plurality of intermediary devices of the cluster. To respond to the SNMP request, the SNMP agent selects a lexicographically minimum entity. The SNMP agent may select the lexicographically minimum entity from a plurality of next entities received via responses from each intermediary device of the plurality of intermediary devices.

Branko Radojević [6] analysed issues with Load Balancing Algorithms in Hosted (Cloud) Environments. In order to provide valuable information and influence the decision-making process of a load balancer, thus maintaining optimal load balancing in hosted (or cloud) environments, it is not enough just to provide information from networking part of the computer system or from external load balancer. Load balancing models and algorithms proposed in the literature or applied in open-source or commercial load balancers rely either on session-switching at the application layer, packet-switching mode at the network layer or processor load balancing mode. The analysis of detected issues for those load balancing algorithms is presented in this paper, as a preparation phase for a new load balancing model (algorithm) proposition. The new algorithm incorporates information from virtualized computer environments and end user experience in order to be able to proactively influence load balancing decisions or reactively change decision in handling critical situations.

Archana B.Saxena1 and Deepti Sharma [7] proposed Analysis of Threshold Based Centralized Load Balancing Policy for Heterogeneous Machines. Heterogeneous machines can be significantly better than homogeneous machines but for that an effective workload distribution policy is required. Maximum

realization of the performance can be achieved when system designer will overcome load imbalance condition within the system. Load distribution and load balancing policy together can reduce total execution time and increase system throughput. In this paper; we provide algorithm analysis of a threshold based job allocation and load balancing policy for heterogeneous system where all incoming jobs are judiciously and transparently distributed among sharing nodes on the basis of jobs' requirement and processor capability for the maximization of performance and decline in execution time. A brief discussion of job allocation, transfer and location policy is given with explanation of how load imbalance condition is solved within the system. A flow of scheme is given with essential code and analysis of present algorithm is given to show how this algorithm is better.

P Rafiq, J Kann [8] proposed methods for self-loading balancing access gateways. The present invention is directed towards systems and methods for self-load balancing access gateways. The systems and methods include a master access gateway that receives load metrics and capabilities from a plurality of access gateways. The master access gateway also receives requests to determine if a request to start a new session is to be redirected to access gateways. The master access gateways uses the load metrics and capabilities to select an access gateway to service the request.

D Goel, JR Kurma [9] proposed systems and methods are described for link load balancing, by a multi-core intermediary device, a plurality of Internet links. The method may include load balancing, by a multi-core device intermediary to a plurality of devices and a plurality of Internet links, network traffic across the plurality of Internet links. The multi-core device providing persistence of network traffic to a selected Internet link based on a persistence type. A first core of the multi-core device receives a packet to be transmitted via an Internet link to be selected from the plurality of Internet links. The first core sends to a second core of the multi-core device a request for persistence information responsive to identifying that the second core is an owner core of a session for persistence based on the persistence type. The first core receives the persistence information from the second core and determines to transmit the packet to the Internet link previously selected based on the persistence information received from the second core.

T. Abdelzaher, K. Shin[10] proposed the Internet is undergoing substantial changes from a communication and browsing infrastructure to a medium for conducting business and marketing a myriad of services. The World Wide Web provides a uniform and widely-accepted application interface used by these services to reach multitudes of clients. These changes place the Web server at the center of a gradually emerging e-service infrastructure with increasing requirements for service quality and reliability guarantees in an unpredictable and highly-dynamic environment. This paper describes performance control of a Web server using classical feedback control theory. We use feedback control theory to achieve overload protection, performance guarantees, and service differentiation in the presence of load unpredictability. We show that feedback control theory offers a promising analytic foundation for providing service differentiation and performance guarantees. We demonstrate how a general Web server may be modeled for purposes of performance control, present the equivalents of sensors and actuators, formulate a simple feedback loop, describe how it can leverage on real-time scheduling and feedback-control theories to achieve perclass response-time and throughput guarantees, and evaluate the efficacy of the scheme on an experimental testbed using the most popular Web server, Apache.
Experimental results indicate that control-theoretic techniques offer a sound way of achieving desired performance in performance-critical Internet applications. Our QoS (Quality-of-Service) management solutions can be implemented either in middleware that is transparent to the server, or as a library called by server code

JH Kim, GS Choi [11] proposed load balancing scheme for cluster-based secure network servers. Although the secure sockets layer (SSL) is the most popular protocol to provide a secure channel between a client and a cluster-based network server, its high overhead degrades the server performance considerably, and thus, affects the server scalability. Therefore, improving the performance of SSL-enabled network servers is critical for designing scalable and high performance data centers. In this paper, we examine the impact of SSL offering and SSL-session aware distribution in cluster-based network servers. We propose a backend forwarding scheme, called
SSL_WITH_BF that employs a low-overhead user-level communication mechanism like VIA to achieve good load balance among server nodes. We compare three distribution models for network servers: Round Robin (RR), SSL_With_Session and SSL_WITH_BF through simulation. The experimental results with 16-node and
32-node cluster configurations show that while session reuse of SSL_With_Session is critical to improve the performance of application servers, the proposed backend forwarding scheme can further enhance the

performance due to better load balancing. The SSL_With_BF scheme can minimize average latency by about 40% and improve throughput across a variety of workloads.

Mohit Aron Peter Druschel Willy Zwaenepoel [12] Proposed a resource management framework for providing predictable quality of service (QoS) in Web servers. The framework allows Web server and proxy operators to ensure a probabilistic minimal QoS level, expressed as an average request rate, for a certain class of requests (called a S*ervice*), irrespective of the load imposed by other requests. A measurement-based admission control framework determines whether a service can be hosted on a given server or proxy, based on the measured statistics of the resource consumptions and the desired QoS levels of all the co-located services. In addition, we present a feedback-based resource scheduling framework that ensures that QoS levels are maintained among admitted, co-located services. Experimental results obtained with a prototype implementation of our framework on trace-based workloads show its effectiveness in providing desired QoS levels with high confidence, while achieving high average utilization of the hardware.

Suresha and Jayant R. Haritsa [13] proposed techniques on reducing Dynamic Web Page Construction Times Many web sites incorporate dynamic web pages to deliver customized contents to their users. However, dynamic pages result in increased user response times due to their construction overheads. They proposed mechanisms for reducing these overheads by utilizing the excess capacity with which web servers are typically provisioned. Specifically, we present a caching technique that integrates fragment caching with anticipatory page pre-generation in order to deliver dynamic pages faster during normal operating situations. A feedback mechanism is used to tune the page pre-generation process to match the current system load. The experimental results from a detailed simulation study of our technique indicate that, given a fixed cache budget, page construction speedups of more than fifty percent can be consistently achieved as compared to a pure fragment caching approach. We have proposed a hybrid approach to reduce dynamic web page construction times by integrating fragment caching with page pre-generation, utilizing the spare capacity with which web servers are typically provisioned. Through the use of a simple linear feedback mechanism, we ensure that the peak load performance is no worse than that of pure fragment caching. A detailed study of the hybrid approach over a range of cache ability levels and prediction accuracies, for a given cache budget. Experimental results show that an even 50-50 partitioning between the page cache and the fragment cache works very well across all environments. With this partitioning, we are able to achieve over fifty percent reduction in server latencies as compared to fragment caching. This approach achieves both the long-term benefit through fragment caching and the immediate benefit through anticipatory page pre-generation. An investigation can be done on the performance effects of pre-generating a set of pages, rather than just a single page.

J Guitart, D Carrera, V Beltran, J Torres [14] proposed Session-Based Adaptive Overload Control for Secure Dynamic Web Applications. As dynamic web content and security capabilities are becoming popular in current web sites, the performance demand on application servers that host the sites is increasing, leading sometimes these servers to overload. As a result, response times may grow to unacceptable levels and the server may saturate or even crash. In this paper we present a session-based adaptive overload control mechanism based on SSL (Secure Socket Layer) connections differentiation and admission control. The SSL connections differentiation is a key factor because the cost of establishing a new SSL connection is much greater than establishing a resumed SSL connection (it reuses an existing SSL session on server). Considering this big difference, we have implemented an admission control algorithm that Prioritizes the resumed SSL connections to maximize performance on session-based environments and limits dynamically the number of new SSL connections accepted depending on the available resources and the current number of connections in the system to avoid server overload. In order to allow the differentiation of resumed SSL connections from new SSL connections. They proposed a possible extension of the Java Secure Sockets Extension (JSSE) API. Their evaluation on Tomcat server demonstrates the benefit of our proposal for preventing server overload.

T. Abdelzaher, K. Shin [15] proposed mechanisms and policies for supporting HTTP/1.1 persistent connections in cluster-based Web servers that employ content-based request distribution. We present two mechanisms for the efficient, content-based distribution of HTTP/1.1 requests among the back-end nodes of a cluster server. A trace-driven simulation shows that these mechanisms, combined with an extension of the locality-aware request distribution (LARD) policy, are effective in yielding scalable performance for HTTP/1.1 requests. We implemented the simpler of these two mechanisms, *back-end forwarding*. Measurements of this mechanism in connection with extended LARD on a prototype

cluster, driven with traces from actual Web servers, confirm the simulation results. The throughput of the prototype is up to four times better than that achieved by conventional weighted round-robin request distribution. In addition, throughput with persistent connections is up to 26% better than without.

J Brendel, CJ Kring, Z Liu, CC Marino [16] proposed world-wide-web server with delayed resource-binding for resource-based load balancing on a distributed resource multi-node network. A multi-node server transmits world-wide-web pages to network-based browser clients. A load balancer receives all requests from clients because they use a virtual address for the entire site. The load balancer makes a connection with the client and waits for the URL from the client. The URL specifies the requested resource. The load balancer waits to perform load balancing until after the location of the requested resource is known. The connection and URL request are passed from the load balancer to a second node having the requested resource. The load balancer re-plays the initial connection packet sequence to the second node, but modifies the address to that for the second node. The network software is modified to generate the physical network address of the second node, but then changes the destination address back to the virtual address. The second node transmits the requested resource directly to the client, with the virtual address as its source. Since all requests are first received by the load balancer which determines the physical location of the requested resource, nodes may contain different resources. The entire contents of the web site are not mirrored onto all nodes. Network bottlenecks are avoided since the nodes transmit the large files back to the client directly, bypassing the load balancer. Client browsers can cache the virtual address, even though different nodes with different physical addresses service requests.

Deniz Ersoz, Mazin S. Yousif and Chita R. Das proposed [17] Characterizing Network Traffic in a Cluster-based, Multi-tier Data Centre. With the increasing use of various Web-based services, design of high performance, scalable and dependable datacentres has become a critical issue. Recent studies show that a clustered, multi-tier architecture is a cost-effective approach to design such servers. Since these servers are highly distributed and complex, understanding the workloads driving them is crucial for the success of the ongoing research to improve them. In view of this, there has been a significant amount of work to characterize the workloads of Web-based services. However, all of the previous studies focus on a high level view of these servers, and analyse request-based or session-based characteristics of the workloads. In this paper, we focus on the characteristics of the network behaviour within a clustered, multi-tiered data centre. Using a real implementation of a clustered three-tier data centre, we analyse the arrival rate and inter-arrival time distribution of the requests to individual server nodes, the network traffic between tiers, and the average size of messages exchanged between tiers. The main results of this study are; (1) in most cases, the request inter-arrival rates follow log-normal distribution, and self-similarity exists when the data centre is heavily loaded, (2) message sizes can be modelled by the log-normal distribution, and (3) Service times fit reasonably well with the Pareto distribution and show heavy tailed behaviour at heavy loads.

V. PROPOSED METHOD

The proposed system is designed to increase throughput and balance the servers based on different workloads. The traditional method has flaws in the load balancing of the server but with the new implanted technique on the server improves the performance during the high load. The secure socket layer with Load balancing scheme has been introduced to overcome server load problems. Storing and serving effectively and securely is more important so that desired algorithm is going to implement for load distribution and security enhancement named as Secure Socket Layer with Load Balancing and RSA Security algorithm respectively. The results are reviewed with 16 and 32 node cluster system. With new technique the latency of system has been decreased by the 40 % and throughput of the system is extremely better than classical balancing technique. We provide algorithm analysis of a threshold based job allocation and load balancing policy for heterogeneous system where all incoming jobs are judiciously and transparently distributed among sharing nodes on the basis of jobs' requirement and processor capability for the maximization of performance and decline in execution time.

VI. CONCLUSION

The performance implications of the SSL protocol for providing a secure service in a cluster-based application server will be investigated and proposed a back-end forwarding scheme for improving server performance through a better load balance. The proposed scheme exploits the underlying user-level communication in order to minimize the intracluster communication overhead. The prosed system will be more robust in handling variable file sizes.

REFERENCES

[1] Anoop Reddy, Rama Rao Katta, Bhanu Prakash Valluri, Craig Anderson, Ratnesh Singh Thakur: Protect applications from session stealing/hijacking attacks by tracking and blocking anomalies in end point characteristics, published on November 26, 2011 in Search International and National Patent collections, No.WO2015179310.

[2] Dipesh Gupta, Hardeep Singh: Review on TLS or SSL session sharing based web cluster load balancing. In proceedings of International Journal of Research in Engineering and Technology Volume: 03 Issue: 11, Nov-2014.

[3] Robson Eduardo, De Grande: Dynamic Load Balancing Schemes for Large Scale HLA Based Simulations. In proceedings of 15th IEEE/ACM International Symposium on Distributed Simulation and Real time Applications, Published on September 2011, pp.4 – 11.

[4] S Annamalaisami, R Holla: Systems and methods for supporting a snmp request over a cluster, Published on Dec 19, 2013, Citrix Systems, No. WO2013188780 A1.

[5] K. Kungumaraj, T. Ravichandran: Load balancing as a strategy learning task. In proceedings Scholarly Journal of Scientific Research and Essay (SJSRE) Vol. 1(2) on, April 2012, pp. 30-34

[6] Branko Radojević MIPRO: Analysis of issues with load balancing algorithms in hosted (cloud) environments, proceedings in 34th International Convention 2011, on May 23-27, pp. 416 – 420

[7] Archana B.Saxena1 and Deepti Sharma: Analysis Of Threshold Based Centralized Load Balancing Policy For Heterogeneous Machines. In proceedings International Journal of Advanced Information Technology (IJAIT) Vol. 1, No.5, October 2011

[8] P Rafiq, J Kann : Systems and methods for self-loading balancing access gateways, published on May 19, 2015

[9] D Goel, JR Kurma: Citrix Systems, IncSystems and methods for link load balancing on a multi-core device, published on Jan 12, 2012

[10] T. Abdelzaher, K. Shin and N. Bhatti. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. IEEE Transactions on Parallel and Distributed Systems Vol. 13 (1), pp. 80-96. January 2002.

[11] JH Kim, GS Choi, CR Das: A Load Balancing Scheme for Cluster-based Secure Network Servers, proceedings in IEEE International Conference on Cluster Computing, on September 2005,pp. 1-10

[12] Mohit Aron Peter Druschel Willy Zwaenepoel. A resource management framework for providing predictable quality of service (QoS) in Web servers Available online : www.researchgate.net /publication/228537697

[13] Suresha and Jayant R. Haritsa: Techniques on reducing Dynamic Web Page Construction Times Volume 3007 of the series Lecture Notes in Computer Science, pp. 722-731. Available online: link.springer.com/chapter/10.1007

[14] J Guitart, D Carrera, V Beltran, J Torres: Session-based adaptive overload control for secure dynamic Web applications. In proceedings International Conference on Parallel Processing (ICPP), on June 2005, pp.341 - 349

[15] T. Abdelzaher, K. Shin and N. Bhatti Efficient Support for P-Http in ClusterBasedweb Servers. In proceedings USENIX Annual Technical Conference,Monterey, California, USA, June 6-11, 1999.

[16] J Brendel, CJ Kring, Z Liu, CC Marino: world-wide-web server with delayed resource-binding for resource-based load balancing on a distributed resource multinode network, published on Jun 30, 1998, Patent no. 5,774,660.

[17] Deniz Ersoz, Mazin S. Yousif and Chita R. Das: Characterizing Network Traffic in a Cluster-based, Multi-tier Data Centre. In proceedings 27th International Conference on Distributed Computing Systems (ICDCS '07), published on June 2007, pp-59.