Balancing HTTP traffic using dynamically updated weights, an implementation approach

A. Karakos, D. Patsas, A. Bornea and S. Kontogiannis

Democritus University of Thrace,
Dept. Of Electrical & Computer Engineering, Xanthi, Greece.
{karakos, dpats, ancutzab, skontog}@ee.duth.gr

Abstract. In this paper we present a load balancing application for HTTP traffic that uses dynamic weights. We introduce a load balancing policy based on two criteria: "*process time*" and "*network delay*". The former describes Web servers ability to process a forthcoming request, while the latter tries to estimate network conditions. Calculation of the two criteria is periodically updated. A *Weighted Round Robin* algorithm was implemented using the two aforementioned metrics in order to dynamically estimate the balancing weights.
We confirm that the combination of the two criteria increases sensitivity and responsiveness of the application towards network conditions and therefore the performance of the whole load balancing system. Balancing decisions should not be only "load" or "connection" dependent, but also contention dependent.

1 Introduction

Web system scalability is defined as the ability to support large numbers of accesses and resources, while still providing adequate performance. Taxonomy of scale-up techniques, differentiates in cost effective hardware and software solutions. Furthermore, software scale-up is classified to operating system improvements, building more efficient Web servers and implementing different scheduling policies on requests. For example, to improve performance of the Apache Web server, Nanda et al. [21], have proposed several reduction techniques on the number of system calls in the typical I/O path. Zeus Web server [22] uses a small number of single-threaded I/O processes, that each one can handle many simultaneous connections. Queuing policies like: *Shortest Remaining Processing Time first*, proposed by Bansal et al. [9], improve system performance.

In addition to the previous scalable techniques, a distributed Web system comprised by several server nodes can also present scale-up enhancements. Distributed Web architectures are classified to: *Cluster-based Web systems (or Web-clusters)*, *Virtual Web clusters* and *Distributed Web systems*, as mentioned by Colajanni et al. [15]. Our Web system implementation uses the *Cluster based Web system* architecture and is comprised of the following structural parts; The *routing mechanisms* that redirect clients to appropriate target servers, the *dispatching algorithm* that selects the best suited target servers to respond to requests

and the *executor* that supports the routing mechanisms and carries out the dispatching algorithm. From the basic structural parts of our system we focus on the selection process of the dispatching algorithm. We present the estimation criteria and mechanism in detail and evaluate the benefits of our algorithm towards performance and efficiency of the balancing system. This paper structure is as follows: In section 2 we examine several Cluster-based Web system techniques and mechanisms. In section 3 we analyze our implementation, the dispatching algorithm structure and weight calculation. In section 4 we present our scenario and discuss our implementation results.

2 Cluster-Based Web Systems

A cluster-based Web system is a collection of Web server machines, joined together as a single unity. These Web server machines are interconnected through broadband networks and oppose a single system image for the outside world. A cluster-based Web system is advertised with a site name and a virtual IP address (VIP). Figure 1, depicts a clustered Web system. The front end node of such a

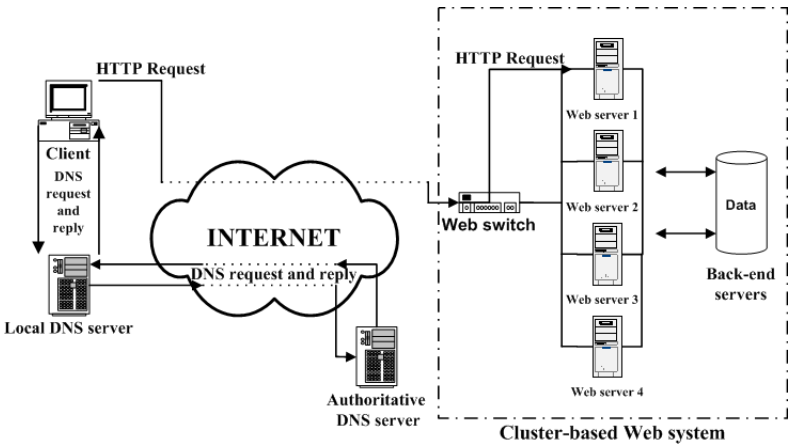


Fig. 1. The architecture of a cluster-based Web system.

system, called *"Web switch"*. The Web switch receives all in-bound packets that clients send to the VIP address and routes them equivalently to the Web-server nodes.

2.1 Routing mechanisms

There are two basic categories of Web switches that differentiate on the usage of "content-aware" (Layer-7 switch) or "content-blind" (Layer-3,4 switch) routing mechanisms.

Content-blind routing process is instantiated upon the arrival of the first "SYN" packet at the Web switch, indicating a new TCP connection. The routing techniques commonly used in a content-blind environment are the following: 1. Packet rewriting based on the IP Network Address Translation approach [10], [11], 2. Packet tunneling and 3. Packet forwarding (also referred as "MAC address translation"). These techniques may apply for both in-bound and out-bound HTTP traffic. De-centralized approaches like DPR [1] that use the aforementioned methods and do not depend on the existence of centralized control may apply. Statefull inspection per connection at the Web switch is also an option, but degrades significantly system performance. Moreover, Content blind load balancing is commonly implemented by providing mappings from a single host name to multiple IP addresses. This prequisites the existence of a DNS authority that will advertise IP addresses in a Round Robin fashion [12].

Content-aware routing examines HTTP requests at the application layer. It is less efficient than *Content-blind* routing, but can support more sophisticated dispatching policies. Some routing methods that are used to such environments are: 1. TCP proxy getaways (transparent or not), 2. TCP splicing [4] and for one way architectures only: 1. TCP connection hop [13] and 2. TCP hand-off [18]. The major advantage of Layer-7 routing mechanisms is the use of content-aware dispatching algorithms at the Web switch. On the other hand Layer-7 routing mechanisms introduce severe processing overhead at the Web switch to the extent of degrading Web cluster scalability, as mentioned by Levy et al. [5].

2.2 Dispatching Algorithms

The dispatching policy used at the Web switch, coordinates the whole clustered Web system. There are several alternatives of dispatching algorithms, such as: connection sharing, load sharing, centralized, distributed, static and dynamic algorithms. Dispatching algorithms are the heart of a load balancing system, but their tendency is not towards absolute stability, especially when we are dealing with a highly dynamic case. Perfect cluster work-balance is not the aim of a load balancing system, but user efficiency is. In this paper we will present a load balancing implementation that uses content blind routing and dynamic dispatching algorithms.

Static dispatching algorithms do not consider any kind of system state information. Typical examples of such algorithms are *Random* and *Round-Robin*. Although the round-robin DNS [12] works this way, it is quite different from *Round-Robin*; Per host DNS caching that usually takes effect, will lead the system to a definite state of load imbalance. In contrast round robin implementation in virtual server [19] is more superior to round-robin DNS due to fine scheduling per connection granularity. Furthermore, the static *Weighted Round Robin* (WRR) can be used as dispatching algorithm on Web servers with different processing capacities in order to introduce fairness. In this case, each server is assigned a static integer weight by the network administrator that corresponds to an index of its own capacity. More specifically: $w_i = \frac{C_i}{min(C)}$, where $min(C)$ is the minimum server capacity. The dispatching Round-Robin sequence will be

generated according to the server weights. Moreover, a "fair WRR" implementation was proposed at [19] and implemented by [7], [20].

Dynamic dispatching algorithms take into account client-server state information in order to achieve more efficient load balancing. Client state algorithms categorize clients to client families using administrative criteria or hash functions that are applied to client IP addresses. Server state algorithms take into account information advertised by servers or agent look-ups, in order to investigate several server conditions. These conditions are important for the selection of the appropriate dispatching policy. Some the criteria commonly used are: "number of connections", "server uptime" or "ICMP reply time".

Both Random and Round-Robin policies can easily extend to treat web servers of heterogeneous capacities, as mentioned at [8] and [15]. For example if C_i is an indication of the server capacity, the relative server capacity can be defined as:

$$R_i = \frac{C_i}{max(C_1, ..C_i, ..C_k)} \ , \ 0 \leq R_i \leq 1 \tag{1}$$

For Random policy, different probabilities can be assigned to heterogeneous capacities. For Round-Robin policy, a random generated number p, where $0 \leq p \leq 1$, can be compared with relative server capacity in order to circulate to another server. That is:

$$S_{i+1} \leftarrow \ if \ \ p \leq R_i$$

else:

$$S_{i+2} \leftarrow Next$$

The *Weighted Least Connections* approach [20], [19], that is used also in many commercial systems [3], takes into account current server connections in order to assign new requests to the server with the least number of active connections. Similarly, the *Least Loaded* approach uses the WRR algorithm, with weights depended on the advertised snmp protocol load index. Moreover, the *Fastest Response Time* policy, assigns new connections to the Web servers that respond faster. The basic criteria used is the HTTP response time which equals to the amount of time that is needed for Web servers HTML objects to be downloaded by the Web switch. Finally, the dynamic *Geographically* distributed systems and algorithms place one ore more Web clusters in different strategic Internet regions and use HTTP redirection mechanisms trying to "divide and conquer" clients. Such an approach uses locational information accommodated with periodically updated Web server status advertisements [16].

The term "Load balancing" for Cisco describes also a functionality in a router that distributes packets across multiple links based on Layer-3 routing information. This means that the balancing process is either per destination or per packet and the scheduling policy depends on routing protocols (RIP, IGRP). Cisco's main representative is IGRP routing protocol [2]. IGRP is a protocol that allows balancing switches (getaways) to build up their routing tables based on broadcasting routing updates. It uses the generic Bellman-Ford algorithm, modified in three aspects: First, instead of a simple metric, a vector of metrics

is used to characterize paths. Second, instead of picking a single path with the smallest metric, traffic is split among several paths (where metrics fall into a specified range). Third, several features are introduced for stability, like *split horizon*, *hold-down* and fix of *erroneous routes* [2]. Composite metric computation is achieved using the following equation. The best path is selected based on a composite metric:

$$CM = (K_1 \cdot BW + \frac{K_2 \cdot BW}{256 - Load} + K_3 \cdot Dc)(\frac{K_5}{Reliability + K_4}) \qquad (2)$$

where Dc equals to the sum of switching delay between interfaces, circuit delay (propagation delay of 1 bit) and transmission delay (for a 1500 bit broadcast packet). BW is the Bandwidth of the narrowest bandwidth segment (interface) of the path. Reliability is the fraction of packets that arrive undamaged at the destination. IGRP-based load balancers are fast, general purpose load balancers that use at some extent network information n order to score a "correct" balancing decision, but not in thorough extent (queueing delay). From the other hand being fast is not a precondition for efficiency. Being more selective and less fast might be more efficient than being less selective and fast. Summarizing, the drawbacks of Cisco IGRP balancing solutions are: 1) The use of broadcast as updates 2) high cost and 3) the fact that a generic network specific balancing might not be always efficient for specific application needs and from an application-specific load balancer additionally.

2.3 Web-content distribution

HTTP content distribution among Web systems is very important in terms of performance. Several techniques have been proposed in order to improve efficiency of the load balancing systems. The use of a distributed file system among the Web servers, like NFS, ensures the consistency of data. The major drawback of a distributed file-system is the multiple I/O requests between Web servers and the remote shared file-system. Multiple I/O requests degrade significantly system performance and constitute the NFS share as the system's weakest point.

Another commonly used technique for Web-content distribution, is the periodic file content replication method. With this technique each cluster node maintains a local copy of the Web documents. During periods of low traffic, replication agents update the contents of Web server nodes, while control agents supervise the replication process. The main drawback of this technique is that it introduces heavy disk and network overheads when data are highly volatile and frequently updated.

Finally, the most promising technique for content distribution is data partitioning. With Web data partitioning, each Web server maintains some part of the Web documents and is responsible only for its replication. Partitioning techniques increase scalability and offer significant reduction of the overhead due to periodic file replication. On the other hand partitioning may eventually lead to uneven distribution of Web document popularity and thus producing load imbalance.

3 Implementation

We implemented a load balancing application that tries to estimate dynamically both network conditions and Web systems capability, in order to make balancing decisions.

3.1 Introduction of the criteria

Two basic criteria were used in order to calculate and adjust the Web servers weights; These are "*HTTP response time*" and "*network delay*". Criteria calculation is periodically updated. More specifically, the load balancer sends an HTTP request to each server and waits for a reply. The application estimates the time that the request was sent and the time that the reply was received. The time difference between request and reply, corresponds to the *HTTP response time*. If, however, the server does not respond within a fixed "timeout" interval, then the application marks it as being down and excludes it from receiving further requests (for another fixed period).

The *HTTP response time* metric is equal to the sum of the network propagation delay, network queuing delay and Web server processing delay:

$$http\_resp = Q_{Prop} + Q_n + Q_{Proc} \tag{3}$$

Propagation delay is the time required for a signal to travel from one point to another and it is assumed equally divided among all Web servers. *Queuing delay* in a packet-switched network is the sum of all the delays encountered by a packet from the time of its insertion into the network until the delivery to its destination. *Processing delay* of a Web server is the time needed for a process to perform a request and construct an appropriate reply message.

In order to estimate network delay a "TCP SYN" packet is constructed, with source IP the IP address of the Web switch and destination IP address, each one of the Web servers. An appropriate timeout is set for "SYN" packet transmission, in order to avoid the application to be characterised as a malicious one. The application running at the Web switch, sends the packet and awaits for an acknowledgment. Then it calculates the inter-arrival time between request and reply. This inter-arrival time is an index of the propagation and queuing delay between the Web server and the Web switch; An approximation of the Web server *network delay*:

$$ndelay = Q_{Prop} + Q_n \tag{4}$$

Network delay metric is a fair estimation of the propagation and queuing delay of the Web server. We assume a constant propagation delay for all Web servers and therefore we extract it from the previous equations. The time difference between *HTTP response time* and *network delay* metrics equals to the processing delay at the Web server. If the Web server does not respond within a "timeout" interval then the application marks the server as being down and excludes it from receiving any further requests (for a period of time). When the application starts,

an initial static weight is distributed among the Web servers. This static weight is introduced by the administrator during installation of the application. The value of the initial weight must be analogous to the maximum *HTTP response time* (in seconds), that according to the administrator, satisfies his network perceptive needs.

3.2 Dispatching Algorithm structure

The dispatching algorithm used by the application is as follows: If all the Web servers have *HTTP response time* greater than the initial criterion, then the weights are calculated using the product of process time for each server with the network delay (propagation plus queuing delay). The reason for using the product instead of the sum is to increase sensitivity of the load balancing system towards network conditions.

3.3 Weight Calculation

When the load balancing application starts, it parses the configuration file in order to collect information about the initial weights. These weights are introduced by the network administrator as a primer estimation of the system's balance point. Then a round-robin database is created where the load balancing criteria will be held. Monitoring pages with "real-time" graphs of Web servers criteria status are also created. The criteria used in the current application are: *HTTP response time*, *network delay* and static weights that were introduced initially by the administrator. At first, the "dynamically updated" criteria are not known and therefore the requests received by the load balancer will be periodically redirected to all servers.

After each time interval (period), defined at the configuration file, the metrics of *HTTP response time* and *network delay* for each server are updated. If the *HTTP response time* of all Web servers takes a value different than zero or at least one Web server has *HTTP response time* less than the initial criteria (introduced by the administrator), then the weights for each Web server are calculated using the following equation:

$$\mathbf{Weight}_i = \frac{\frac{\mathbf{1}}{\mathbf{http\_resp}_i}}{\sum_{i=1}^{\mathbf{n}} \frac{\mathbf{1}}{\mathbf{http\_resp}_i}} \tag{5}$$

where $i = 1..n$ is the number of Web servers that load balance HTTP traffic. If there is at least one Web server with *HTTP response* time less than its initial criterion, then the weight calculation is performed according to equation (5). For those servers that their corresponding *HTTP response time* is greater than their initial criterion their weights are assigned to zero. However if there are no Web servers that have *HTTP response time* less than the static criteria, then the weights of all Web servers that are up and running are estimated based on

equation (6):

$$\mathbf{Weight}_i = \frac{\frac{\mathbf{1}}{\mathbf{ndelay}_i \cdot (\mathbf{http\_resp}_i - \mathbf{ndelay}_i)}}{\sum_{i=1}^{\mathbf{n}} \frac{\mathbf{1}}{\mathbf{delay}_i \cdot (\mathbf{http\_resp}_i - \mathbf{ndelay}_i)}} \tag{6}$$

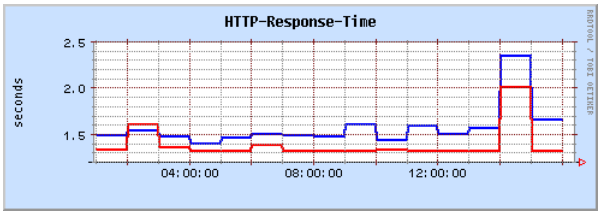
As it can be seen from (6), both *network delay* and *HTTP response time* are used for the weight estimation. In addition, if none of the servers are up for more than three time periods, then the administrator is notified. Based on the weight calculation, the server that will receive the first requests is found and WRR is implemented. Threaded Agents are called to modify the kernel "iptables" and redirect (NAT) the incoming requests.

4 Scenario results and discussion

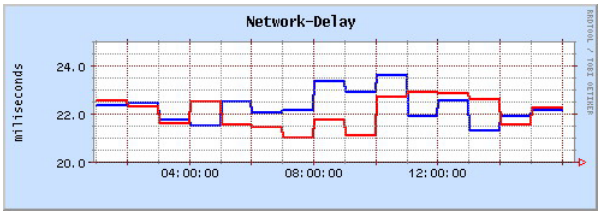
Application output is consisted of two graphs per Web server that plot in "real-time" the variation of *HTTP response time* and *network delay* correspondingly. In an experimental scenario, we implemented the load balancing application on a Web switch that runs a Linux operating system. This Web switch supports two Web servers of equivalent processing power. The first Web server ("*blue-one - black*") sustains twice the number of HTTP requests (HTTP GET 250 Mbyte File) than the second Web server ("*red-one - light gray*"), for a period of twelve(12) hours. We measured both *HTTP response time* and *network delay* metrics (from the application output) and the dynamically updated weights. The time variations of the two metrics (using one minute update interval), are depicted in figure 2.

It is obvious from the graphs that the "*red*" Web server has less processing delay and therefore *HTTP response time*, than the "*blue*" one. This is because it serves less requests. So the "*red*" Web server will be selected for handling HTTP traffic for almost the whole duration. What about congestion and network delay [17]? Changing the value of the initial criteria to less than one second, *network delay* metric will be taken into account and will be multiplied to *HTTP response time* metric, for the final weight calculation. This will increase the sensitivity of the dispatching algorithm towards congestive incidents, introducing queuing delay as important factor as the processing delay for the final Web server selection. So when *network delay* metric of the "*red*" Web server overcomes the value of the "*blue-one*" (congestive network conditions [17]), the "*blue*" Web server will also be as applicable as the "*red-one*", (for handling part of the HTTP traffic), as depicted in figure 2(b) for the time interval 11h-2h. Furthermore, another information that can be extracted from the two figures is that both metrics converge in time, representing the actual meaning of load balancing.

TOS field in the IP header and IEFT differentiating services architecture [14], are commonly used by network administrators in order to provide QoS and traffic classification. This type of provision is mainly application performance dependent. In addition, HTTP traffic alone is in need for further classification so as to meet different application requirements. For instance, someone might



(a) HTTP response time variation over time



(b) Network delay variation over time

Fig. 2. Output results of our implementation using dynamically updated weights

be using HTTP protocol in order to download files from an HTTP service while someone else might be using HTTP protocol for an interactive application (like chat) from the same service. Moreover someone might require different types of service for a single HTTP flow. For instance he might need to perform a fast HTTP database query (non-congestive), while the HTTP database reply might not be that fast (congestive). As proposed by the authors of [6] TCP traffic can be differentiated according to the queueing delay that it may cause to a router to: congestive or non-congestive. This differentiation is based on the packet length and can be easily translated by a router without packet rewriting, marking or further calculation delays. In a higher level of abstraction, HTTP traffic can also be classified as congestive or non-congestive and handled appropriately by the Web switch. This can be performed with the use of an NCQ [6] or a "*metric of congestion*" that will be incorporated into the weight calculation formula. This capability will be included in the following version of our Web switch application (dispatcher).

5 Conclusions

It is important for an efficient load balancing system to take decisions both according to processing time and network delay. Classic WRR algorithms and

dispatchers don't take into consideration network conditions. Usually, snmp protocol requests that carry information about the Web server's load and network connections are used to dynamically update dispatching algorithms and routing selections. This is a minor, since queueing and therefore network delay does not always follow the processing ability of the Web server.

It should also be taken into consideration that network queues that are responsible for handling in "general" TCP traffic, are not used only by HTTP traffic. Routers that stand in the way of a multi-server load balancing system might get congested by external factors and start dropping HTTP packets. Such congestive situations may cause unpredictable behavior to the performance and scalability of a cluster-based Web system. The introduction of network delay into the selection process of a load balancing system, will lead the system to a more operational equilibrium, improving both its performance and efficiency. Differentiation and classification among HTTP flows according to the application requirements, will take us a step further. Congestive or Non-congestive, hi or low priority HTTP traffic that will experience different types of service, may lead to better and more fair resource sharing.

References

1. A. Bestavros, M. Crovella, J. Liu and D. Martin. "Distributed Packet Rewriting and its Application to Scalable Web Server Architectures". *In Proc. of the 6th International Conference on Network Protocols ICNP*, Octomber 1998.
2. Cisco Company. "Cisco - An introduction to IGRP", 2004.
3. Cisco Inc. "Cisco Distributed Director", `http://www.cisco.com/warp/public/cc/pd/cxsr/dd/`, 2004.
4. D. Maltz and P. Bhagwat. "Application layer proxy performance using TCP splice". Technical report, IBM T.J. Watson Research Center, 1998.
5. J. Song, E. Levy-Abegnoli and D. Dias. "Design alternatives for scalable Web server accelerators". *In Proc. of the 2000 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2000.
6. L. Mamatas and V. Tsaoussidis. "A new approach to Service Differentiation: Non-Congestive Queueing". Technical report, Democritus University of Thrace, 2004.
7. Linux VS Team. "Linux Virtual Server Implementation", 2002.
8. M. Colajanni, P. Yu and M. D. Dias. "Analysis of task assignment policies in scalable distributed Web-server systems". *IEEE Trans. on Parallel Distributed Systems*, June 1998.
9. N. Bansal and M. Harchol Balter. "Analysis of SRPT scheduling: Investigating unfairness". *In Proc. of the 2001 ACM/IFIP Joint International Conference on Measurement and Modeling of Computer Systems*, March 2001.
10. P. Srisuresh and D. Gan. "Load sharing using IP Network Address Translation", RFC 2391, 1999.
11. P. Srisuresh and K. Egevang. "Traditional IP Network Address Translation", RFC 3022, 2001.
12. R. J. Schemers. "ldnamed: A load Balancing Name Server in Perl". *In Proc. of the 9th Systems Administration Conference*, 1995.
13. RESONATE Team,. "TCP Connection Hop". *White paper*, April 2001.

14. S. Blake, D. Black, M. Carlson, E. Davies et al. "An Architecture for Differentiated Service", RFC 2475, December 1998.
15. V. Cardellini, E. Casalicchio, M. Colajanni and P. Yu. "The State of the Art in Locally Distributed Web-Server Systems". *ACM Computing Surveys*, Vol. 34 No. 2:263–311, June 2002.
16. V. Cardellini, M. Colajanni, P.S. Yu. "Geographic load balancing for scalable distributed Web systems". *In Proc. of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2000.
17. V. Jacobson. "Congestion Avoidance and Control". *In Proc. of the ACM SIGCOMM '88*, August 1988.
18. V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel and E Nahum. "Locality-aware request distribution in cluster-based network servers". *In Proc. of the 8th ACM Conference on Architectural Support fro Programming Languages and Operating Systems*, Octomber 1998.
19. W. Zhang. "Linux Server Clusters for Scalable Network Services". *Free Software Symposim, China*, 2002.
20. W. Zhang and W. Zhang. "Linux Virtual Server Clusters". *Linux Magazine*, November 2003.
21. Y. Hu, A. Nanda and Q. Yang. "Measurement, analysis and performance improvement of Apache Web server". *In Proc. of the 18th IEEE International Performance, Computing and Communications Conference*, February 1999.
22. Zeus Development Team. "Zeus Web Server", `http://www.zeus.com.uk`. Technical report, Zeus Technology, 2002.