Request Distribution with Pre-Learning for Distributed SSL Reverse Proxies

Haitao Dong, Jun Yang
Key Laboratory of Noise and Vibration Research
Institute of Acoustics, Chinese Academy of Sciences
Beijing, China
donght@dsp.ac.cn, jyang@mail.ioa.ac.cn

Yiqiang Sheng
National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
Beijing, China
shengyq@dsp.ac.cn

***Abstract*—As network data security becoming more and more universalized, distributed Secure Sockets Layer (SSL) reverse proxies are often used in Web systems to offload CPU exhausting SSL operations from Web servers and improve the execution performance of the SSL protocol. The distribution strategy of user requests to the SSL reverse proxies is a significant factor affecting the system's performance in processing SSL operations. Aiming at improving the quality of request distribution decisions, this paper proposes a new approach for SSL reverse proxy load estimation, i.e. the family of algorithms called Load Estimation with Pre-Learning (LEPL), which estimates load using pre-learned machine learning models. Using LEPL, high accuracy of load estimation can be achieved, so that better request distribution decisions can be made. Our experimental results show that by using pre-learning, the SSL reverse proxy system's average response time can be shortened by about 30% - 50%.**

Keywords—Web system; SSL reverse proxy; distributed system; request distribution; load estimation; machine learning

I. INTRODUCTION

Today, people's urgent requirements for communications security make network data security tend to achieve a universalization within the range of the whole Internet [1]. SSL protocol [2] and its successor Transport Layer Security (TLS) protocol [3-5] are presently the most popular protocols used to provide a secure transmission channel between a client and a server on the Internet, and have already become the de facto standard for transport layer security [6, 7]. In the rest of this paper, the term "SSL" is used to refer to both SSL and TLS. Nevertheless, the cryptographic operations, particularly public key operations used in negotiating SSL sessions, tend to be highly CPU intensive [2-6, 9, 10]. Due to this reason, SSL's popularity rate over the Internet is rather low [1, 8]. Only a small part of network traffic in people's everyday life is transmitted in cipher text, the most of which is traffic carrying sensitive information. Improving the execution performance of the SSL protocol has become a key technology in upgrading network data's secrecy level. In order to offload CPU exhausting SSL operations from Web servers and improve the execution performance of the SSL protocol, several locally or globally distributed SSL reverse proxies [9-13] which usually possess hardware devices dedicated to process cryptographic operations at great speed are often deployed in front of Web servers in a Web system that support SSL, as shown in Fig. 1. A particularly obvious problem is the distribution strategy of incoming requests to the SSL reverse proxies.

In this paper, we focus on improving the quality of request distribution decisions for distributed SSL reverse proxies. We assume an SSL reverse proxy system consisting of a front-end request distributor and several back-end SSL reverse proxy nodes. Our load estimation algorithms work on the request distributor, which is connected between clients and the back-ends in series. First, the request distributor establishes a TCP connection with a client. Second, the request distributor receives a ClientHello and checks it for information on SSL session resumption, then chooses a target node according to some request distribution strategy. Third, the request distributor transfers its end point of the TCP connection to the target node using the technique of TCP hand-off [14]. Then the client can establish SSL connection with the target node and send HTTPS request to it.

In our work, a new method for SSL reverse proxy load estimation is proposed. It is a family of algorithms called LEPL, which estimates load using machine learning models. These models are pre-learned, which means that they are trained offline and then brought online. By employing LEPL, high

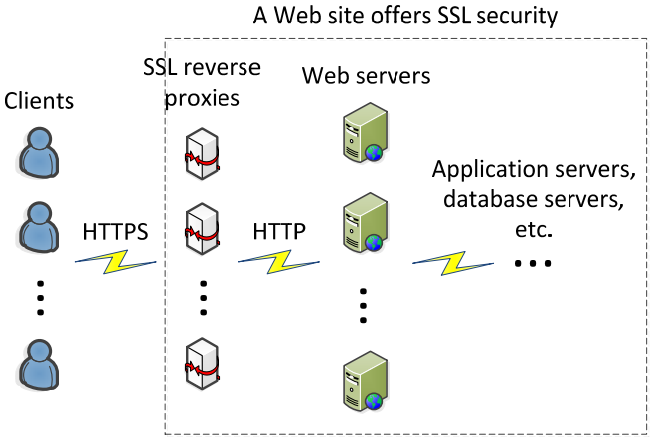

Fig. 1. Architecture of a Web site offers SSL security using distributed SSL reverse proxies.

Project supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA06010302).

978-1-5090-2239-7/16/$31.00 copyright 2016 IEEE
SNPD 2016, May 30-June 1, 2016, Shanghai, China

accuracy of load estimation can be achieved, so that better request distribution decisions can be made and as a result the average response time can be shortened. The LEPL algorithms are highly real-time, in which a node's current load is estimated using the amount of load distributed to it during a period of time in the past, without gathering the node's state information. This paper presents the following contributions:

- The load estimation method of LEPL for SSL reverse proxies.
- Two specific LEPL algorithms using machine learning models of back propagation neural network (BPNN) and regression tree with post-pruning (RTPoP) respectively. These models are trained offline to be aware of the relationship between the nodes' current load and the amount of load distributed to them during a certain period of time in the past.
- A set of simulation that evaluates the performance of the LEPL algorithms.

The rest of this paper is organized as follows: In Section II we give some related work. In section III we describe the LEPL algorithms in detail. In Section IV we describe the design of simulation used to evaluate the performance of our algorithms and then present the experimental results. Finally, we conclude this paper in Section V.

II. Related Work

In previous research on request distribution strategies for a Web server system, the calculation methods of the nodes' load can be divided into two categories. Methods of the first category need to monitor the nodes' state in real time, measuring the nodes' load as the occupation situation of the nodes' one or multiple hardware resources (such as CPU, memory, disk, etc.) or network resources [15-17]. However, a mass of works disagree with monitoring the nodes' state. They hold the view that it is difficult to monitor the nodes' state [18, 19], or the nodes' state information is of little use in load balancing [20], or monitoring the nodes' state may reduce the system's performance [21]. In the second category of load calculation methods, the request distributor obtains the nodes' load information by estimating, rather than by monitoring the nodes' state. The basis of the estimation is diversified. For example, it can be the number of active TCP connections on each node [19], the data throughput of each node [18], the HTTP response time and network delay of each node [22], etc. The idea of not monitoring the nodes' state makes these methods simple and fast. However, the lack of the nodes' state information makes these methods perform relatively poor in accuracy of estimation. The LEPL algorithms belong to this category, with their estimation accuracy improved by the mechanism of pre-learning. In the authors' previous work [23], a real-time load estimation algorithm based on discrete sliding window was proposed. In this algorithm, firstly the time axis is split into time slices of duration ΔT, then the load of a node at time t is estimated using the normalized load distributed to it during the former time slice and the current one. The LEPL algorithms are designed based on the expansion of this discrete sliding window. In previous works, load calculation methods seldom predict the nodes' load between load information updating intervals in real-time. This idea of real-time predicting is proposed in [15]. We adopt this idea in our estimation algorithms as well.

In the past years, machine learning methods have already been applied in many works on request distribution for distributed systems, mainly used to improve the accuracy of request distribution decision making. Machine learning methods used in the field of request distribution can be divided into two categories: adaptive learning and pre-learning. Adaptive learning methods keep learning and updating system parameters in real-time while the system is working. The request distribution strategy of Fuzzy Adaptive Request Distribution (FARD) [24-26] used traditional error back propagation algorithm in training the learning system, so as to predict the response time of each back-end node in real-time. Reference [27] used an adaptive learning BPNN to get hold of the relationship between server nodes' percentage of CPU usage, processor clock speed, memory utilization and capacity, network bandwidth utilization, server's NIC bandwidth and their response time. In pre-learning methods, the learning system is trained offline by sufficient training samples and then brought online. In comparison with adaptive learning, pre-learning sacrifices some response accuracy. However, the characteristic of no real-time training makes it perform better in response speed. In [28], the authors used two pre-learning models of radial basis function neural network (RBFNN) and adaptive neuro-fuzzy inference system (ANFIS) to correct the error of load estimation. These learning models use the estimated value and the stretch error as inputs, and output the corrected estimated value. In the case of SSL workload, the computational overhead brought by each user request can be relatively precisely predicted, so we try to find the relationship between a node's current load and the amount of load distributed to it during a period of time in the past, with the help of machine learning methods. In selecting target node for every user request, the estimation of the nodes' load should be fast and highly real-time. So we adopt pre-learning in this paper.

III. The LEPL Algorithms

In this section, we describe the load estimation method of LEPL in detail. The family of LEPL algorithms is designed based on a model of discrete sliding window, which we will firstly present in the following subsection. Next, we will present two specific LEPL algorithms using the machine learning models of BPNN and RTPoP respectively.

A. The Model of Discrete Sliding Window for Load Estimation

Fig. 2 depicts the model of discrete sliding window, in which the time axis is split into time slices of duration ΔT. Suppose that the size of the window is m, then the window at current time t covers the time slice t belongs to (end with t) and the former m–1 time slices. As shown in Fig. 2, suppose that the amount of load the request distributor sent to a back-end node G_i (i = 1, 2, 3, …, N, where N is the total number of the back-ends) during these m time slices is x_{i1}, x_{i2}, …, x_{im},

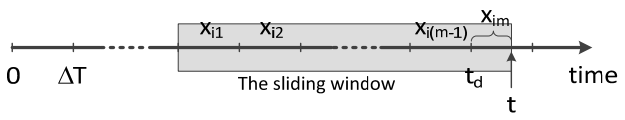

Fig. 2. The model of discrete sliding window.

respectively. Since the last time slice is usually shorter than the other m–1 ones, we normalize x_{im} as follows:

$$x'_{im} = \frac{\Delta T}{t - t_d} \cdot x_{im} \tag{1}$$

Here t_d is the beginning of the time slice t belongs to. We estimate G_i's current load using $x_{i1}, x_{i2}, \ldots, x_{i(m-1)}, x'_{im}$, that is:

$$l_i(t) = f\left(x_{i1}, x_{i2}, \ldots, x_{i(m-1)}, x'_{im}\right) \tag{2}$$

In the LEPL algorithms, the function f is some pre-learned machine learning model.

For load estimation algorithms designed based on (2), the way the request distributor obtains the nodes' load information is proactively estimating without periodically receiving load information from the nodes. Monitoring the nodes' state is not needed, either. As a result, the network delay, bandwidth consumption and extra work on the SSL reverse proxies can be minimized.

For SSL workload, on measuring the amount of load each request brings to the back-ends, we assume that the HTTPS request content size is small enough that the difference in bulk encryption overhead among requests can be neglected. We consider the difference in SSL handshake overhead only and divide the requests into two categories: requests that willing to negotiate a new SSL session and requests willing to resume an existed session s. Each request of the first category is expected to cause a load of $p_{i,new}$ to G_i, while each request of the second category is expected to cause a load of $p_{i,reu}(s)$ to G_i. If s is cached on G_i, then $p_{i,reu}(s) < p_{i,new}$; or otherwise $p_{i,reu}(s) = p_{i,new}$.

B. LEPL Algorithm using BPNN

The establishment and employment of a neural network can be split into three steps: the determination of network architecture, the determination of connection weights, and the working stage. The architecture of a neural network can be defined using three elements: network topological structure, activation function and learning law [27]. Neural networks can be divided into feedforward networks and feedback networks according to their topological structures. In a feedforward network, neurons are divided into layers. Neurons in each layer are connected to the next layer, and are isolated from each other without any connections. In our first LEPL algorithm, we model (2) using a feedforward network, specifically, a BPNN [29]. BPNN is a commonly used kind of feedforward neural network, which has one or several hidden layers besides the input layer and the output layer. The learning law of BPNN is back propagation algorithm while the activation functions are often set as sigmoid function. Our BPNN designed for LEPL is shown in Fig. 3. This network has two hidden layers,

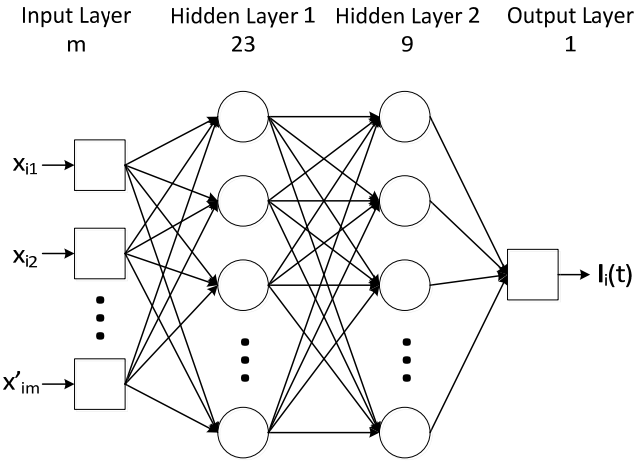

Fig. 3. Back propagation neural network for LEPL.

consisting of 23 and 9 neurons respectively. The nodes in this network are all sigmoid. Through training, this neural network can gradually establish the relationship between $l_i(t)$ and x_{i1}, $x_{i2}, \ldots, x_{i(m-1)}, x'_{im}$. The training set should have sufficient inputs-output pairs, and it should cover full range of variation in inputs as well.

C. LEPL Algorithm using RTPoP

The learning result of decision tree machine learning method is a tree, called decision tree, which can sort objects into different classes. In a decision tree, each non-leaf node refers to a property of the objects; each edge refers to some feasible values of these properties; while each leaf node refers to a class of the objects. The procedure of classifying an object starts from the root node. First, the object is checked on the node's corresponding property. Second, according to the value of this property, a corresponding edge is selected and a new node is encountered. Then, the above two steps are repeated until a leaf node is encountered, so that the class of the object is obtained. The procedure of generating a decision tree also starts from the root node. A node's corresponding property and the edges that stretch from it are determined by some learning law, then the child nodes of this node are generated. These operations are repeated until some prescribed stopping criterion is reached. Then the node stops expanding and becomes a leaf node. When the training set is large enough, the generated decision tree is often used in regression analysis. Such a decision tree is also called a regression tree. Sometimes there are noises in the training set, which make the generated decision tree fit the training set well but perform rather badly in processing the subsequent data. Pruning the decision tree is an effective way of avoiding this phenomenon of overfitting [30, 31]. The pruning methods can be divided into pre-pruning and post-pruning. Post-pruning is done after the decision tree has been generated, whose goal is to improve the accuracy of classification, whereas the pre-pruning is done while the decision tree is being generated. In designing our second load estimation algorithm, to fit the relationship between $l_i(t)$ and x_{i1}, $x_{i2}, \ldots, x_{i(m-1)}, x'_{im}$, we generated a regression tree using the classic learning law of C4.5 and the post-pruning method of reduced-error pruning. An example of such an RTPoP is shown

in Fig. 4. It is worth noting that the regression model is very sensitive to noises. Therefore, in generating the regression tree, the training set should be composed of very large amount of inputs-output pairs, and it should cover full range of variation in inputs as well.

IV. Performance Evaluation

In order to evaluate the performance of the LEPL algorithms, a set of simulation was designed based on the system architecture presented in Section I and then carried out. We implemented the two LEPL algorithms described in Section III in the request distributor together with a certain request distribution decision making algorithm. Next, a request flow with bursty arrivals arrived at the request distributor and the requests in it were distributed to the back-end SSL reverse proxies. Two other request distribution strategies were selected as benchmarks to evaluate the performance of the proposed algorithms. In the following subsections, the design details of our simulation and the experimental results will be presented.

A. System Parameters

Table I provides system parameters in our simulation. The SSL reverse proxy system has 4 homogeneous nodes with the same computing performance. The lifetime of SSL session for a typical Web site can be set from 5s to 24h [9]. In our simulation, we set it as 5min.

B. Request Parameters

In order to test the proposed algorithms' performance in system response time when coming across bursty arrivals, we designed and implemented a request flow which was generated by adding bursty arrivals to a gentle client request flow. These bursty arrivals are caused by a large number of rude clients that crowd into the system in one second every other a fixed interval of time. For such a rude client, the total number of requests is extremely varied, while the time interval between two successive requests is extremely short.

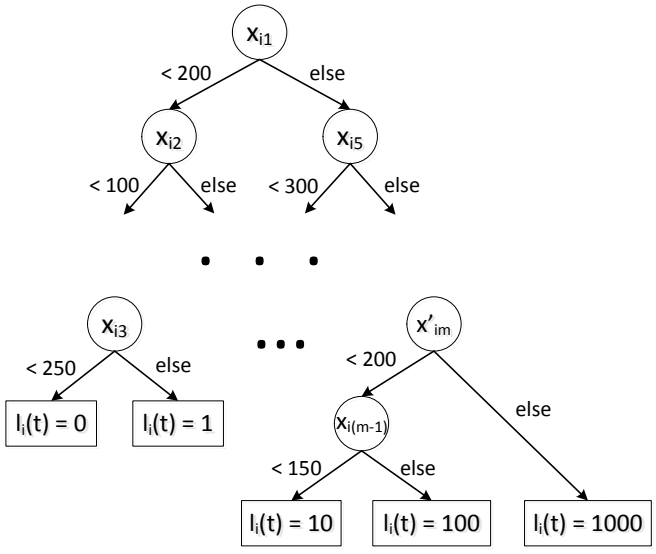

Fig. 4. An example of regression tree with post-pruning for LEPL.

Table II provides request parameters of the gentle request flow, which was designed as normal as possible. Experience suggests that the number of newcome clients per second obeys Poisson distribution, while the moment when a new client arrives obeys uniform distribution. The mean value of newcome clients per second is 20. Our SSL reverse proxy system does not have any pressure at such level of load. The study on online user behavior in [32] provided statistical data of average requests per user session. We did a curve fitting to these statistical data and as a result, we modeled the number of requests per user session according to a negative binomial distribution. The user think time refers to the time between the retrieval of two successive requests from the same client. In our simulation, the user think time was modeled according to a Pareto distribution [33]. The object of each HTTPS request is a 128KB Web page, which conforms to the assumption that the HTTPS request content size is small enough.

Request parameters of the rude clients are provided in Table III. In the second that the rude clients crowd in, the number of newcome rude clients and the moment when a new rude client arrives also obey Poisson distribution and uniform distribution respectively. The number of requests per user session needs to be extremely varied, so we modeled it according to a discrete uniform distribution from 1 to 2000. The user think time was modeled according to a normal distribution in which both the parameters μ and σ are very

TABLE I. System Parameters in the Simulation

Parameters of the SSL reverse proxies	
Number of the SSL reverse proxies (N)	4
Performance in private key decryption	80times/s
Performance in symmetric encryption (or decryption)	800Mbps
Lifetime of SSL sessions	5min
Public key algorithm	RSA, key size = 2048bits
Symmetric key algorithm	AES, key size = 256bits
Parameters of the request distributor (Configurations of the LEPL algorithms)	
Size of the sliding window (m)	11
Length of each time slice (ΔT)	0.2s

TABLE II. Request Parameters of the Gentle Request Flow in the Simulation

Newcome clients per second	Poisson distribution, $\lambda = 20$
Moment when a new client arrives	Uniform distribution
Requests per user session	Negative binomial distribution, $r = 1$, $p = 0.023$
User think time (s)	Pareto distribution, $\alpha = 1.4$, $k = 2$
Content size	128KB

TABLE III. Request Parameters of the Rude Clients in the Simulation

Newcome clients in one second	Poisson distribution
Moment when a new client arrives	Uniform distribution
Interval between two successive crowds of rude clients	30s
Requests per user session	Discrete uniform distribution, min = 1, max = 2000
User think time (s)	Normal distribution, $\mu = 0.005$, $\sigma = 0.001$
Content size	128KB

small. As a result, the time interval between two successive requests from the same client tends to be short and fixed.

C. The Selected Request Distribution Algorithm and Benchmarks

To evaluate the performance of the two LEPL algorithms, these algorithms should work in the request distributor together with some request distribution decision making algorithm. In our simulation, we adopted a request distribution decision making algorithm based on multi-objective optimization [23]. The core idea of this algorithm is making compromise between two contradictory factors: load balance and high SSL session resumption rate. For requests willing to negotiate a new SSL session, since there is no session resumption, the distribution decision-making follows the principle of load balancing. For requests willing to resume an existed session, the distribution decision makes compromise between load balance and high SSL session resumption rate. If $\mathbf{x}$ stands for a request distribution decision-making, $f_1(\mathbf{x})$ stands for the load caused by $\mathbf{x}$, $f_2(\mathbf{x})$ stands for the standard deviation of all the back-end nodes' load after distributing a request according to $\mathbf{x}$, then the utility function of distributing a request willing to resume an existed session is:

$$U(\mathbf{x}) = \alpha_1 \cdot \left\{ \frac{f_1(\mathbf{x}) - \min\left[f_1(\mathbf{x})\right]}{f_1(\mathbf{x})} \right\}^{\lambda_1} + \alpha_2 \cdot \left\{ \frac{f_2(\mathbf{x}) - \min\left[f_2(\mathbf{x})\right]}{f_2(\mathbf{x})} \right\}^{\lambda_2} \tag{3}$$

Among all the noninferior solutions, the one that minimizes $U(\mathbf{x})$ is the optimal solution of this algorithm. In (3), α_1, α_2, λ_1 and λ_2 are constants whose values depend on the number and processing ability of the nodes. Through adjusting these four parameters, the relative weights on load balance and SSL session resumption can be changed freely, so as to adapt the algorithm to SSL reverse proxy systems of different specifications. In our simulation, the values of these parameters were $\alpha_1 = 1$, $\alpha_2 = 1$, $\lambda_1 = 1.5$, $\lambda_2 = 2$.

Request distribution for SSL reverse proxies can be done at the granularity of client or at the granularity of request [23]. The family of LEPL algorithms is designed for request distribution at the granularity of request. Client-granularity request distribution strategy is a classical method, which was selected as a benchmark in our simulation. In this strategy, requests from the same client are always distributed to the same node, so that the SSL session resumption mechanism can be fully utilized. The first request from each client can be distributed using algorithms like WRR, LL, etc. In our simulation we adopted WRR.

The second benchmark we selected [23] is a request-granularity request distribution strategy, whose distribution decision is also made according to (3). The load estimation algorithm of this benchmark uses a basic discrete sliding window method in which the window size is a fixed value of 2. The discrete sliding window method described in Section III *A*. is its expansion. In this load estimation algorithm, the current load of G_i can be calculated as follows:

$$l_i(t) = \frac{x_1 + x_2}{\Delta T + t - t_d} \tag{4}$$

In our simulation, ΔT in (4) was set as 1s.

D. Experimental Results

In coming across short-term congestion caused by bursty arrivals, an excellent load estimation algorithm enables the request distributor to get precise information about the occupation situation of the back-ends' resources. As a result, it is much easier for the request distributor to find the most suitable target node and the response time can be effectively shortened. We tested the average response time on processing the request flow described in Section IV *B*. using the following four request distribution strategies: client-granularity request distribution (CG), request-granularity request distribution with the basic discrete sliding window method (RG-Basic), request-granularity request distribution with LEPL algorithm using BPNN (RG-BPNN), and request-granularity request distribution with LEPL algorithm using RTPoP (RG-RTPoP). To emphasize the effectiveness of our methods, we added a group of experiment in which an ideal request-granularity request distribution strategy (RG-ideal) was used. In this ideal strategy, the request distributor always knows the exact load information of each node. The experimental results are shown in Fig. 5. Using client-granularity request distribution, requests from the same client are always distributed to the same node. In the situation where there are a large amount of clients whose requests are frequent, load skewness is very likely to happen. Therefore, this strategy performs worst in average response time. Comparing with client-granularity request distribution, request-granularity request distribution with the basic discrete sliding window method can greatly improve load balance degree. However, its load estimation algorithm is too simple that the request distributor can't learn about the back-ends' load precisely enough in the face of a request flow with rapid fluctuations. The proposed algorithms achieve much shorter response times due to their mechanism of pre-learning. Through pre-learning, the relative absolute estimation error of the algorithm based on BPNN can be kept below 20%, while

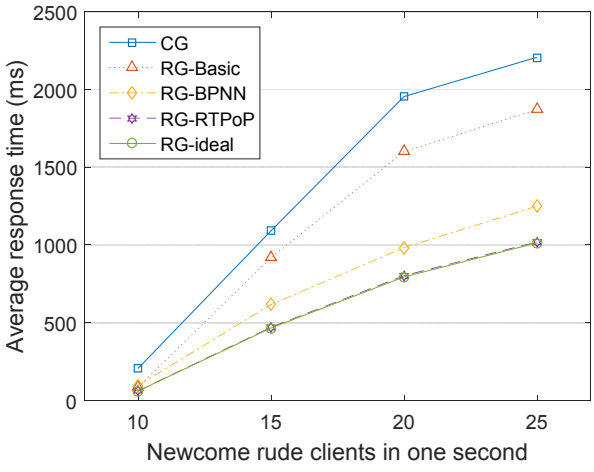

Fig. 5. Average response time of the SSL reverse proxy system.

that of the algorithm based on RTPoP can be kept around 2%. We can observe that the average response time of RG-RTPoP is very close to RG-ideal, indicating that the load estimation accuracy of RG-RTPoP is extremely high.

We define load imbalance degree as the standard deviation of all the nodes' load, and show in Fig. 6 the SSL reverse proxy system's load imbalance degree during the first 15 minutes of our simulation using the four request distribution strategies, with the number of newcome rude clients per second set as 20. We can observe that the load imbalance degree increases at the arrival of the rude clients. In comparison with client-granularity request distribution, request-granularity request distribution can greatly improve the system's load balance degree, while the mechanism of pre learning can further increase this benefit.

V. Conclusions

In this paper, we researched on the problem of improving the quality of request distribution among distributed SSL reverse proxies in a Web system. We proposed a family of highly real-time load estimation algorithms based on pre-learned machine learning models called LEPL, which was specially designed for SSL workload on SSL reverse proxy nodes. Through pre-learning, the accuracy of load estimation can be greatly improved and as a result, the request distributor can make more precise distribution decisions. Two algorithms belong to the family of LEPL were presented and then evaluated in our simulation. The results of our simulation indicate that with the help of pre-learning, the average response time of the SSL reverse proxy system can be shortened by about 30% - 50%.

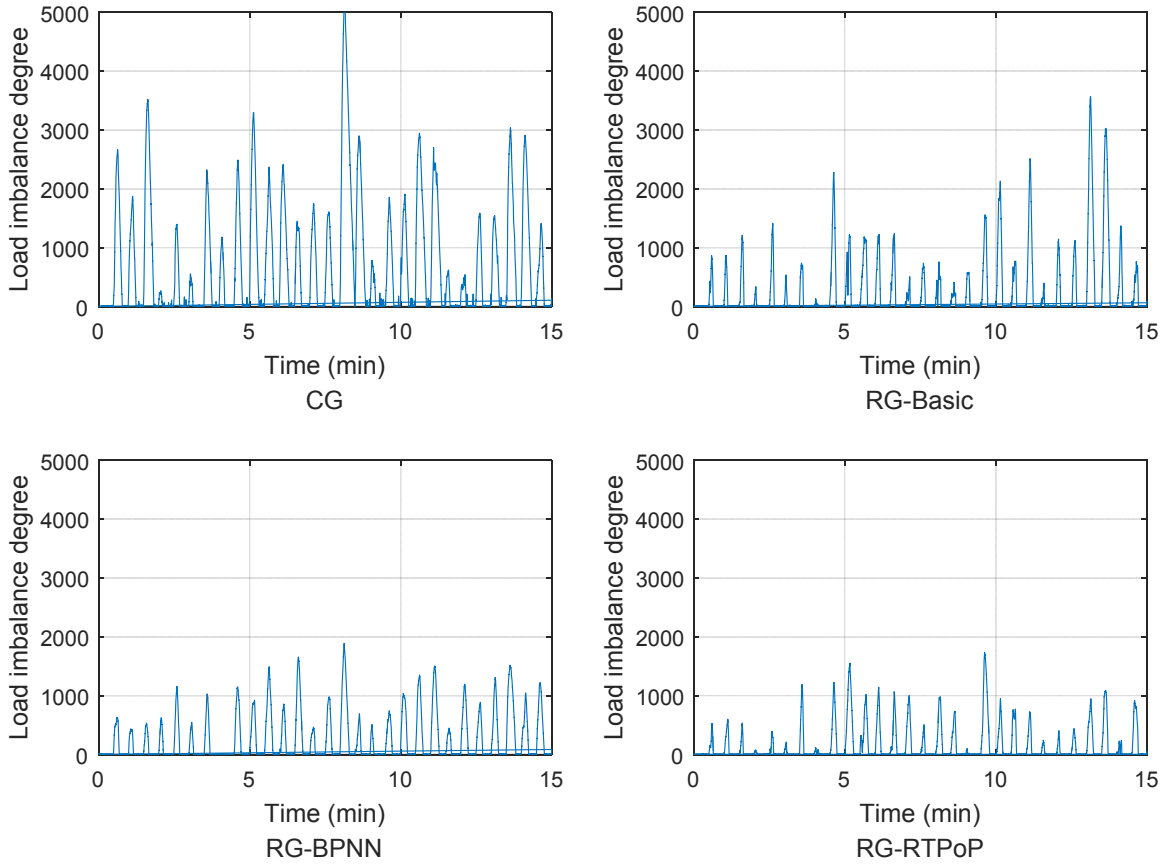

Fig. 6. Load imbalance degree of the SSL reverse proxy system during the first 15 minutes of our simulation, 20 newcome rude clients per second.

REFERENCES

[1] M. E. Kounavis, X. Kang, K. Grewal, M. Eszenyi, S. Gueron, and D. Durham, "Encrypting the internet," ACM SIGCOMM Computer Communication Review, vol. 40, no. 4, pp. 135-146, October 2010.

[2] A. Freier, P. Karlton, and P. Kocher, RFC 6101: The secure sockets layer (SSL) protocol version 3.0, 2011.

[3] T. Dierks, and C. Allen, RFC 2246: The TLS protocol version 1.0, 1999.

[4] T. Dierks, and E. Rescorla, RFC 4346: The transport layer security (TLS) protocol version 1.1, 2006.

[5] T. Dierks. RFC 5246: The transport layer security (TLS) protocol version 1.2, 2008.

[6] J. H. Kim, G. S. Choi, and C. R. Das, "An SSL back-end forwarding scheme in cluster-based web servers," IEEE Trans. on Parallel and Distributed Systems, vol. 18, no. 7, pp. 946-957, July 2007.

[7] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting SSL/TLS implementations: new bleichenbacher side channels and attacks," in Proc. 23rd USENIX Security Symposium, San Diego, 2014, pp. 733-748.

[8] J. Vehent. (November 2014). SSL/TLS analysis of the Internet's top 1,000,000 websites. [Online]. Available: https://jve.linuxwall.info/blog/index.php?post/TLS_Survey

[9] R. Mraz, "Secure blue: an architecture for a scalable, reliable high volume SSL internet server," in: Proc. 17th Annual Computer Security Applications Conference, New Orleans, 2001, pp. 391-398.

[10] R. Hatsugai, and T. Saito, "Load-balancing SSL cluster using session migration," in Proc. 21st International Conference on Advanced Information Networking and Applications, Niagara Falls, 2007, pp. 62-67.

[11] NGINX. (October 2015). SSL-Offloader. [Online]. Available: http://wiki.nginx.org/SSL-Offloader

[12] BlueCoat. (October 2015). Reverse Proxy with SSL - ProxySG Technical Brief. [Online]. Available: https://bto.bluecoat.com/sites/default/files/tech_briefs/Reverse_Proxy_with_SSL.b.pdf

[13] Ericom. (October 2015). Ericom Secure Gateway. [Online]. Available: http://www.ericom.com/securegateway.asp

[14] G. Hunt, E. Nahum, and J. Tracey, "Enabling content-based load distribution for scalable services," Technical report, IBM TJ Watson Research Center, 1997.

[15] J. Jiang, H. Deng, and X. Liu, "A predictive dynamic load balancing algorithm with service differentiation," in Proc. 15th IEEE International Conference on Communication Technology, Guilin, 2013, pp. 372-377.

[16] M. Andreolini, M. Colajanni, and R. Morselli, "Performance study of dispatching algorithms in multi-tier web architectures," ACM SIGMETRICS Performance Evaluation Review, vol. 30, no. 2, pp. 10-20, September 2002.

[17] X. Qin, H. Jiang, Y. Zhu, and D. R. Swanson, "Dynamic load balancing for I/O-and memory-intensive workload in clusters using a feedback control mechanism," in Euro-Par 2003 Parallel Processing, H. Kosch, L. Böszörményi, and H. Hellwagner, Eds. Berlin Heidelberg: Springer, 2003, pp. 224-229.

[18] K. Dutta, A. Datta, D. VanderMeer, H. Thomas, and K. Ramamritham, "ReDAL: an efficient and practical request distribution technique for application server clusters," IEEE Trans. on Parallel and Distributed Systems, vol. 18, no. 11, pp. 1516-1528, November 2007.

[19] V. Ungureanu, B. Melamed, and M. Katehakis, "Effective load balancing for cluster-based servers employing job preemption," Performance Evaluation, vol. 65, no. 8, pp. 606-622, July 2008.

[20] B. Richard, and L. Derek, "Achieving load balance and effective caching in clustered Web servers," in Proc. the Fourth International Web Caching Workshop, San Diego, 1999, pp. 159-169.

[21] E. Casalicchio, and M. Colajanni, "A client-aware dispatching algorithm for web clusters providing multiple services," in Proc. the 10th International Conference on World Wide Web, Hong Kong, 2001, pp. 535-544.

[22] S. Kontogiannis, and A. Karakos, "ALBL: an adaptive load balancing algorithm for distributed web systems," International Journal of Communication Networks and Distributed Systems, vol. 13, no. 2, 2014, pp. 144-168.

[23] H. Dong, L. Song, J. Wang, and J. Yang. "SSLSARD: A request distribution technique for distributed SSL reverse proxies," Journal of Communications. 2016; in press.

[24] L. Borzemski, and K. Zatwarnicki, "A fuzzy adaptive request distribution algorithm for cluster-based web systems," in Proc. Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing, Genova, 2003, pp. 119-126.

[25] K. Zatwarnicki, "Adaptive request distribution in cluster-based web system," in Knowledge-Based and Intelligent Information and Engineering Systems, A. König, A. Dengel, K. Hinkelmann, K. Kise, R. J. Howlett, and L. C. Jain, Eds. Berlin Heidelberg: Springer, 2011, pp. 42-51.

[26] A. Zatwarnicka, and K. Zatwarnicki, "Adaptive HTTP request distribution in time-varying environment of globally distributed cluster-based web system," in Knowledge-Based and Intelligent Information and Engineering Systems, A. König, A. Dengel, K. Hinkelmann, K. Kise, R. J. Howlett, and L. C. Jain, Eds. Berlin Heidelberg: Springer, 2011, pp. 141-150.

[27] M. Tarighi, S. A. Motamedi, S. Sharifian, "Intelligent adaptive multi-parameter migration model for load balancing virtualized cluster of servers," Tehnički vjesnik, vol. 21, no. 4, pp. 763-772, August 2014.

[28] S. Sharifian, S. A. Motamedi, M. K. Akbari. "A predictive and probabilistic load-balancing algorithm for cluster-based web servers," Applied Soft Computing, vol. 11, no. 1, pp. 970-981, January 2011.

[29] M. K. Alsmadi, K. B. Omar, S. A. Noah, I. Almarashdah, "Performance Comparison of Multi-layer Perceptron (Back Propagation, Delta Rule and Perceptron) algorithms in Neural Networks," in Proc. IEEE International Advance Computing Conference, 2009, Patiala, 2009, pp. 296-299.

[30] J. R. Quinlan, "Simplifying decision trees," International Journal of Man-Machine Studies, vol. 27, no. 3, pp. 221-234, September 1987.

[31] F. Esposito, D. Malerba, G. Semeraro, J. Kay, "A comparative analysis of methods for pruning decision trees," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 19, no. 5, pp. 476-491, May 1997.

[32] A. Oke, and R. Bunt. "Hierarchical workload characterization for a busy web server," in Computer Performance Evaluation: Modelling Techniques and Tools, T. Field, P. G. Harrison, J. Bradley, and U. Harder, Eds. Berlin Heidelberg: Springer, 2002, pp. 309-328.

[33] E. Casalicchio, V. Cardellini, and M. Colajanni, "Content-aware dispatching algorithms for cluster-based web servers," Cluster Computing, vol. 5, no. 1, pp. 65-74, January 2002.