Implementation and Evaluation of Simultaneous Session Limitation Mechanism

Ryo SHIBAHARA, Hiroki TONE and Keizo SAISHO
Kagawa University
2217-20 Hayashi-cho, Takamatsu 761-0396, Japan
s18g470@stu.kagawa-u.ac.jp, s17g472@stu.kagawa-u.ac.jp, sai@eng.kagawa-u.ac.jp

ABSTRACT

Responsiveness of Web servers is lowered when they are overloaded caused by a lot of requests from clients. Moreover, Web servers are required to be not only available but also stable responsiveness especially for interactive Web application. In this paper, a mechanism which limits the number of simultaneous sessions using firewall is proposed in order to provide stable Web services. The mechanism consists of authentication server, firewall and user identification server. Authentication server authenticates user and registers IP address of his machine with firewall when the number of current simultaneous sessions is less than the specified number. After this, authenticated users can access Web server. By using firewall, it is possible to not only limit the num- ber of simultaneous sessions but also block malicious attacks such as DoS attack. However, users can access the Web server without authentication when they use same NAT environment or proxy server as authenticated users. User identification server detects access from unauthenticated users and blocks them. This paper describes implementation of user authentication server and user identification server, and evaluation of them. From results of evaluations, we confirm that user authentication server can authenticate and user identification server has tolerance of attack with unauthenticated users.

KEYWORD

Access Limitation, User Authentication, Firewall, User Identification, Web Service

1 INTRODUCTION

In recent years, various Web services are provided. Moreover, a lot of users can use Web services anywhere and anytime with the spread of smart phones. As the number of users increases, requests to Web servers also increase. When excessive requests issue to Web server, it is overloaded. As a result, responsiveness of services is lowered and these services may not available at the worst case. Thus, we decided to develop simultaneous session limitation mechanism that limits the number of simultaneous sessions from authenticated users to solve such situation. The mechanism consists of authentication server, firewall and user identification server. By using firewall, it is possible to not only limit the number of simultaneous sessions but also block malicious attacks such as DoS attack. Authentication server authenticates user and registers IP address of his machine with firewall when the number of current simultaneous sessions is less than the specified number. After this, authenticated users can access Web server. However, users can access the Web server without authentication when they use same NAT environment or proxy server as authenticated users. User identification server detects access from unauthenticated users and blocks them. We use cookie to identify sessions. By checking cookie, user identification server can detect whether session is formally authenticated. This paper describes implementation of user authentication server and user identification server and evaluation of them.

2 RELATED WORKS

In [1], a system is aiming to reduce load of the Web server is proposed. This system is achieved

ISBN: 978-1-941968-58-1 ©2019 SDIWC

that all requests to origin Web server are redirected to the CAPTCHA nodes. The results of experiments show that the latency has increased but the load of the Web server has been reduced and the average response time has been significantly improved, compared to the case of accessing the Web server directly. This research is the same as our research in reducing the load on the web server. It cannot cope with many users accessing simultaneously, but our research can.

There are researches that relaxes user frustration even when Web server is overloaded[2][3]. In [2], WebQ, a system to improve user experience when accessing overloaded Web servers, is proposed. Users accessing a server protected by WebQ receive a HTTP redirect response specifying a wait time in the virtual queue, and are automatically redirected to the web server upon expiration of the wait time. The results of experiments show that requests can be processed normally and response time is also improved even if the load of Web server protected by the proposed system exceeds the capacity of the Web server. In [3], the mechanism which continues important services by lowering the quality of unimportant services rather than restricting users access when Web server is overloaded is proposed. From the results of experiments, it has been confirmed that the proposed approach is effective when Web server is overloaded. Our research can block the malicious attacks, but they cannot block.

In [4], a study is aiming to improve the https server performance. This study proposes cipher suite selection algorithm to meet the different demands for security and response time and strategy which reduces the response time for higher priority requests and guarantees the response time for lower priority requests, while reducing the average system response time. From the results of experiments, it has been confirmed that this study proves the efficiency of the method, and when the server load is high, the advantage of the strategy is more obvious. Our research restricts the number of sessions for improving responsiveness to allowed users, but they use strategy for improving performance.

In [5], a system is aiming to defense against DDoS attacks. This system employs cloud-side proactive and reactive defenses to combat DDoS attacks that may target it. In the proactive defese, coordination server attempts to protect against botnet reconnaissance by periodically changing (via DNS) proxy IP addresses. In the reactive defense, all clients are supposed to use proxies that are overloaded could be periodically reassigned to (shuffled among) the attacked proxies at random to detect attackers. From the results of experiments with some cases, it has been confirmed this system's performance. Our research blocks accesses to Web server using firewall when it reaches upper limit, but their system uses proactive and reactive defenses of shuffling client-to-server assignments.

There are researches that do access control for security and performance[6][7][8]. In [6], new fine-grained two-factor authentication (2FA) access control system with the necessity of both a user secret key and a lightweight security device is proposed. From the results of detailed security analysis and simulation, it has been confirmed to achieve the desired security requirements and to demonstrate the practicability of the proposed 2FA system. In [7], a 2-level fuzzy admission control algorithm to improve system throughput is proposed. The first level named load controller judges the load situation of each tier by the resource consumption and requests delay. The second level named admission controller decides the admission rate of the current session by probability statistics. The experiments show that the algorithm can overcome the limitation of general admission control strategies and improve the system throughput. In [8], a design pattern for improving the performances of a distributed access control mechanism has been proposed. This mechanism has a central authorisation service. A client requests permission to access some services form the central authorisation service through LocalController. At this time, the LocalController caches authorisation information. When the client requests permission again, the LocalController permits the request using cached authorisation information. This drastically decreases the number of requests to the central authorisation service. The observations of the runtime behaviour of various occurrences of such a design pattern on real software systems have shown a drastic increase in performance when compared with a straightforward simpler implementation. Our research uses access control with user authentication and firewall.

3 SIMULTANEOUS SESSION LIMITATION MECHANISM

Figure 1 shows proposed the simultaneous session limitation mechanism. It consists of authentication server, firewall, user identification server. The authentication server (Auth server) authenticates restricted users. The firewall (IPF server) has additional functions that automatically update filtering rules. The user identification server (UI server) denies requests from unauthorized users. SS server really provides some kind of Web service. All users are supposed to access SS server via IPF server and UI server. The overview steps to access SS server from user are shown below.

1. User requires access right to Auth server.
2. Auth server authenticates the user by his account and password and check the number of current sessions.
3. If the number is lower than upper limit then IPF server changes filtering rules and the Auth server informs UI server of the session information of the authenticated user.

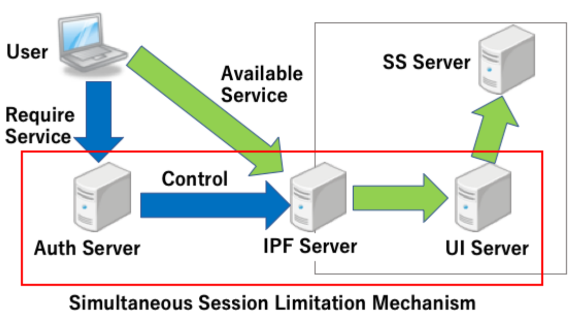

Figure 1: Structure of Simultaneous Session Limitation Mechanism

4. Auth server informs the user of SS server's URL.
5. The user accesses to the URL. The access goes to UI server by DNS setting.
6. UI server checks the session information. If the check is passed then UI server accesses SS server and return result to the user. Otherwise, UI server sends the user the Web page which prompt to be authenticated.

4 DESIGN

We describe design of Auth server and UI server explained in the previous section.

4.1 Auth Server

Auth server has databases to manage user and session. We considered the two following problems at the time of design.

- Protect Auth server from DoS attack
- Protect databases from malicious attack

It is possible to solve DoS attack problem by setting multiple Auth servers. However, this method causes synchronizing problem because each Auth server has databases. Thus, we introduce reverse proxy servers for Auth server. All accesses to Auth server are done via them. By restricting the number of accesses to Auth server at reverse proxy servers, Auth server can avoid overload. It is thinkable that reverse proxy servers deny accesses when the number of requests per second exceeds the specific number. Moreover, all accesses are denied when the number of simultaneous sessions exceeds the upper limit.

It is possible to avoid direct accesses from DoS attackers by setting Auth server inside firewall. This can also solve the second problem. The steps to access SS server by using reverse proxy are shown in Figure 2. The number and title correspond to the number and title in Figure 2.

1. Access
   User accesses any reverse proxy servers.

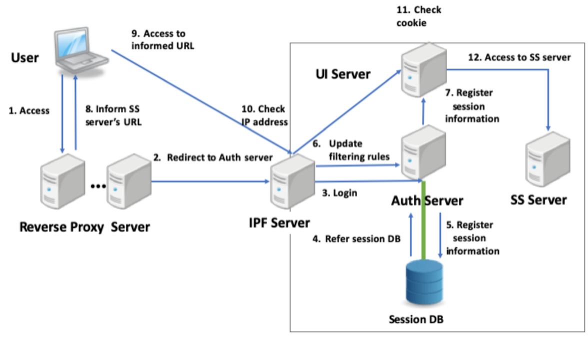

Figure 2: Steps to access SS server

2. Redirect to Auth server
   User accesses Auth server by redirecting.
3. Login
   Auth server authenticates user.
4. Refer session DB
   When the authentication succeeds, Auth server obtains the number of current sessions by referring session DB, and then checks whether it exceeds the upper limit or not.
5. Register session information
   If the check is passed, Auth server registers session information to session DB. Session information has cookie, IP address, expired time and so on.
6. Update filtering rules
   IPF server updates filtering rules according to the session information stored in session DB every 1 second. The operations are done every second. When the new session information is registered session DB, it is applied filtering rules at least after 1 second.
7. Send session information
   Auth server sends UI server session information every 1 second.
8. Inform SS server's URL
   Auth server informs SS server's URL via reverse proxy server.
9. Access to informed URL
   User accesses to Informed URL.
10. Check IP address
    IPF server passes the access to UI server if the access is permitted.
11. Check cookie
    UI server checks cookie included the access.
12. Access to SS server
    If the check is passed, UI server accesses SS server and returns the result to him. Otherwise, UI server sends the user the Web page which prompt to be authenticated.

4.2 UI Server

In the simultaneous session limitation mechanism, UI server identifies authenticated users by checking cookie. We considered two methods. The first method is to manage cookies using database. The database is built on the UI server and stores cookies. The second method is to manage cookies using file. If cookies are managed by a single file, search time increases as the number of cookies increases. By distributing cookies into multiple files using hash method, search time can be reduced. The file name that stores a cookie is the hash value calculated from the value of cookie. We call the former method DB method and latter method HASH method.

5 FUNCTIONAL TEST OF USER AUTHENTICATION METHOD

We performed functional test for authentication method.

5.1 Experiment Environment and Check Points

Figure 3 shows the experiment environment. We use virtual machines for reverse proxy server, Auth server, UI server and SS server. IPF server is physical machine which also hypervisor on which UI server and SS server run. Specifications of hypervisors are also shown in Figure 3. We implemented Auth server and reverse proxy server. We implemented only access deny function which denies when the number of simultaneous sessions reaches upper limit. In future, we will implement access deny function. The following points are tested by actually accessing from browsers.

- User can be authenticated and access SS server.

- User can't access SS server if the number of simultaneous sessions reaches upper limit.
- Reverse proxy server can deny accesses if the number of simultaneous sessions has already reached upper limit.

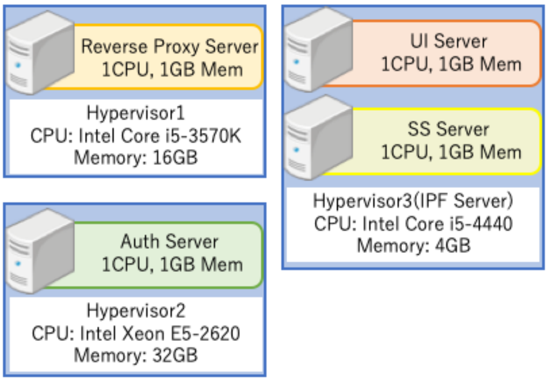

Figure 3: Experiment environment of functional test

5.2 Experiment Procedure

In this experiment, upper limit of the number of simultaneous sessions is set 2. This experiment uses three browsers: Google Chrome(user1), Safari(user2) and Firefox(user3). Since they store cookie independently, their sessions are identified individually. It is possible to use them on the same client. The steps to perform functional test are shown below.

Step-1 Users access to Auth server via reverse proxy server simultaneously with their browsers.

Step-2 user1 sends his account and password to Auth server.

Step-3 user2 sends his account and password to Auth server.

Step-4 user3 sends his account and password to Auth server.

Step-5 user3 accesses to Auth server via reverse proxy server.

5.3 Result

Figure 4 to 8 show results of Step-1 to Step-5, respectively. Figure 4 shows the user1's screen only because the screen is slightly different, but the same content. At Step-2 and Step-3, user1 and user2 were authenticated successfully and got the response from SS server as shown Figure 5 and 6. On the other hand, user3 got the response from Auth server as shown Figure 7. Access to SS server from user3 was denied even though authentication of user3 succeeded because the number of simultaneous sessions reached upper limit. In this experiment, upper limit is 2. At Step-5, reverse proxy server denied the access from user3 because the number of simultaneous sessions already reached the

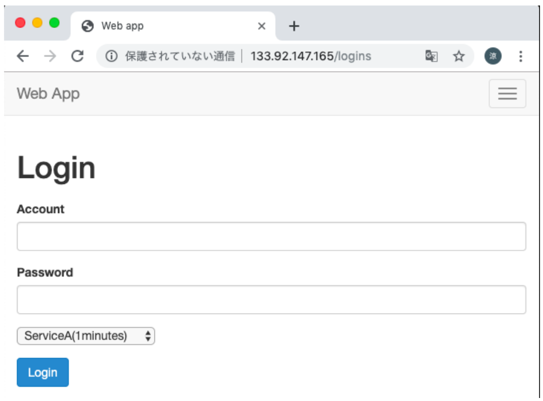

Figure 4: Login screen

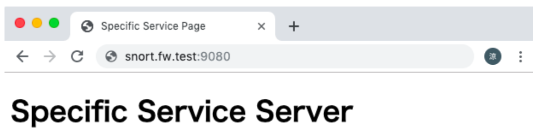

Figure 5: Success to access SS server (user1)

Figure 6: Success to access SS server (user2)

ISBN: 978-1-941968-58-1 ©2019 SDIWC

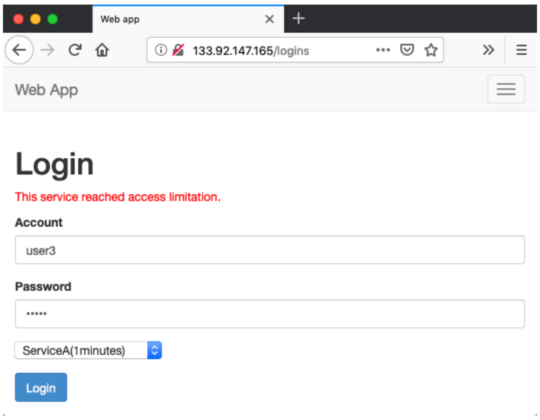

Figure 7: Failure to access SS server (user3)

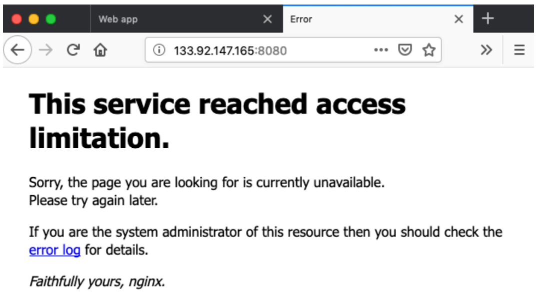

Figure 8: Denies accessing to Auth server by reverse proxy server

upper limit and sent the message showing reason for the denial as shown Figure 8. The results show that our authenticates method works as designed.

6 PERFORMANCE EVALUATION OF UI SERVER

In this section, performance of UI server using two methods are examined. One is DB method and the other is HASH method.

6.1 Experiment Environment

Figure 9 shows experiment environment. All servers and user clients are built as virtual machines on hypervisors which specifications are shown in Table 1. User clients are built on hypervisor1 and hypervisor2, UI server is built on hypervisor3 and SS servers are built on hypervisor4 and hypervisor5. UI server uses Nginx[9], Lua[10] and lua-nginx-module[11]. SS servers use Apache2.4.18[12]. The reason for using multiple SS servers to prevent them from becoming bottleneck.

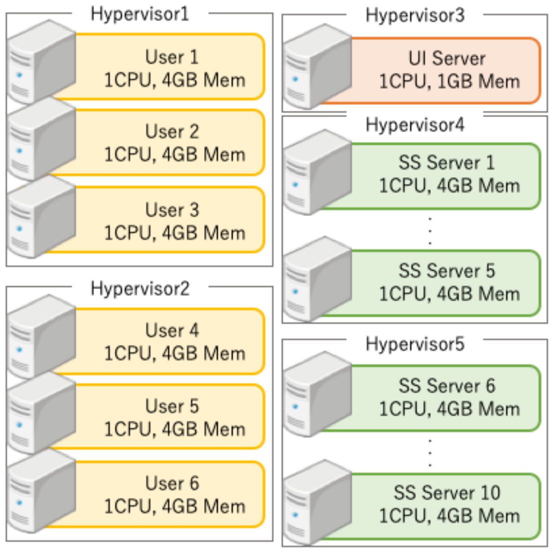

Figure 9: Experiment environment

	CPU	Memory
Hypervisor1	Intel Core i5-3470	16GB
Hypervisor2	Intel Core i5-4460	16GB
Hypervisor3	Intel Xeon E5-2620	32GB
Hypervisor4	Intel Xeon E5-2620	32GB
Hypervisor5	Intel Xeon E5-2620	32GB

Table 1: Specification of hypervisor

6.2 Experiment Procedure

We performed two experiments. One examines throughput of UI server, the other examines capacity of blocking accesses from malicious users.

This first experiment uses 5 user clients and 10 SS servers. The scenario is shown below. The number of accesses per second from each client is increases by N every 30 seconds from 20 accesses per second up to E (maximum number of accesses per second).

The second experiment uses 6 user clients and 10 SS servers. 1 user client plays authenticated users and issues accesses with fixed number of access per second (C). Other user clients play malicious users and issue accesses with same manner as the first experiment. We call accesses from authenticated user and malicious user $Access_A$ and $Access_M$.

6.3 Experiment Result

Figure 10 and 11 show the result of the first experiment with DB method and HASH method, respectively. In this experiment, N is 20 and E is 600. Total N and E are 100 and 3000, since the number of clients is 5. There are some differences in results among clients. These figures are of the client showing the average result. Histogram in figures shows the number of responses per second and the scale is shown by left vertical axis. Green part shows successful responses and red part shows failed responses. Line graph shows the number of requests waiting for response and the scale is shown by right vertical axis. Horizontal axis shows times of day.

First, we examine the result of DB method. In Figure 10, access failure has begun around 300 seconds from the start of experiment (1,100 accesses per second) and the number of requests waiting for response gradually increases. Only about to 250 (= 50×5 clients) accesses per second were successful after 360 seconds (1,300 accesses per second).

Next, we examine the result of HASH method. In Figure 11, access failure has begun around 480 seconds from the start of experiment (1,700 accesses per second) and the number of requests waiting for response also gradually increases. After this, the number of successful accesses per second was decreased gradually and finally reached about 1,750 (= 350×5 clients).

Figure 12 and 13 show the result of the second experiment with DB method and HASH method, respectively. In this experiment, C is 100, N is 20 and E is 1,800. Total N and E are 100 and 9000, since the number of clients playing malicious users is 5. These figures show the number of responses to authenticated user's.

First, we examine the result of DB method. In Figure 12, failure of $Access_A$ per second has begun around 480 seconds from the start of experiment (1,700 $Access_M$s per second). After this, the number of successful $Access_A$ per second is about 30 (30%).

Next, we examine the result of HASH method. In Figure 13, failure of $Access_A$ per second has begun around 2220 seconds from the start of experiment (7,500 $Access_M$s per second). However, the number of $Access_A$ failures per second was less than equal to 3 from log analysis.

From the first experiment, HASH method can process accesses about 1.6 times of DB method when all accesses are issued from authenticated users. From the second experiment, the tolerance of malicious access of HASH method is much higher than that of DB method. We

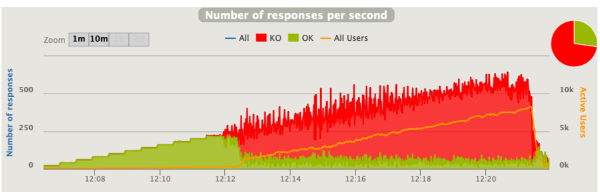

Figure 10: Using database method

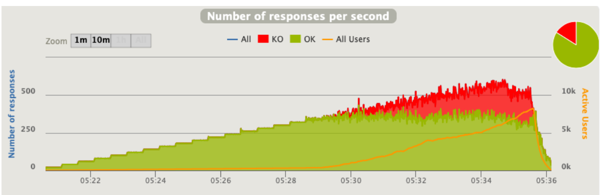

Figure 11: Divided files using hash method

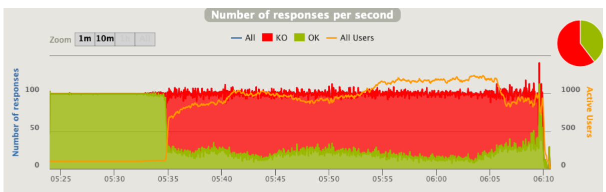

Figure 12: Using database method

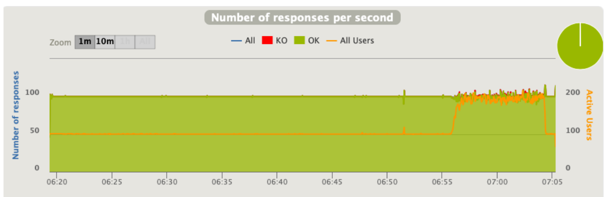

Figure 13: Divided files using hash method

think that HASH method has enough capacity and tolerance of malicious accesses.

7 CONCLUSION

We proposed simultaneous session limitation mechanism and implemented reverse proxy server, authentication server and user identification server that are comprised it. Reverse proxy server and authentication server cooperate to authenticate users and to limit the number of sessions. We confirmed that these functions work correctly. We also implemented two user identification methods, DB method and HASH method, for user identification server. From the results of experiments, HASH method is superior to DB method and has enough capacity and tolerance of malicious accesses.

Future works is shown below.

- Performance evaluation of authentication server
- Implementation of user authentication mechanism using multiple authentication servers to cope with concentration of authentication
- Implementation of access deny function

References

[1] Ahmad T. Al-Hammouri, Zaid Al-Ali, Basheer Al-Duwairi, "ReCAP: A distributed CAPTCHA service at the edge of the network to handle server overload", Transactions on Emerging Telecommunications Technologies 2017, https://doi.org/10.1002/ett.3187

[2] Bhavin Doshi, Chandan Kumar, Pulkit Piyush, Mythili Vutukuru, "WebQ: A virtual queue for improving user experience during web server overload", 2015 IEEE 23rd International Symposium on Quality of Service (IWQoS), pp.135-140, 2015

[3] ZiYou Wang, MingHui Zhou, Hong Mei, "Towards a degradation-based mechanism for adaptive overload control", 19th Asia-Pacic Software Engineering Conference (APSEC 2012), pp.52-60, 2012

[4] Lu Yan, Haojiang Deng, Xiao Chen, Xiaozhou Ye, "Service Differentiation Strategy Based on User Demands for Https Web Servers", 2018 IEEE 16th International Conference on Software Engineering Research, Management and Applications (SERA), pp.189-194, 2018

[5] Yuquan Shan, George Kesidis, Daniel Fleck, "Cloud-Side Shuffling Defenses against DDoS Attacks on Proxied Multiserver Systems", CCSW '17 Proceedings of the 2017 on Cloud Computing Security Workshop, pp.1-10, 2017

[6] Joseph K. Liu, Man Ho Au, Xinyi Huang, Rongxing Lu, Jin Li, "Fine-Grained Two-Factor Access Control for Web-Based Cloud Computing Services", IEEE Transactions on Information Forensics and Security, Vol.11, No.3, pp.484-497, 2016

[7] Pan Dan, Gan Hong, "Building of Multi-Tier Web Server Access Control Design and Research", Proceedings of the 2nd Information Technology and Mechatronics Engineering Conference (ITOEC 2016), pp.422-427, 2016

[8] Emiliano Tramontana, "A Design Pattern for Improving the Performances of a Distributed Access Control Mechanism", Proceedings of 5th Asian Conference on Pattern Languages of Programs (AsianPLoP 2016), pp.80-88, 2016

[9] Nginx, https://nginx.org/en/

[10] lua, https://www.lua.org/

[11] lua-nginx-module, https://github.com/openresty/lua-nginx-module

[12] Apache, https://httpd.apache.org