Large-scale Certificate Management on Multi-tenant Web Servers

Ryosuke Matsumoto
GMO Pepabo, Inc.
Email: matumotory@pepabo.com

Kenji Rikitake
GMO Pepabo, Inc. / KRPEO
Email: kenji.rikitake@acm.org

Kentaro Kuribayashi
GMO Pepabo, Inc.
Email: antipop@pepabo.com

Abstract—**In large-scale certificate management on multi-tenant web servers, preloading a large number of certificates for managing a large number of hosts under the single server process results in increasing the required memory usage due to the respective page table entry manipulation, which may be poor resource efficiency and reduced capacity. To solve this issue, we propose a method to dynamically load the certificates bound to the hostnames found during the SSL/TLS handshake sequences without preloading, provided the Server Name Indication (SNI) extension is available. We implement the function of choosing the respective certificates with the ngx_mruby module which extend Web server functions using mruby with small memory footprint while maintaining the execution speed. We also evaluated the proposed method on a Web hosting service of authors' place of an employer.**

1. Introduction

Free Domain-validated (DV) certificates such as Let's Encrypt [3] are beginning to be provided, and supporting HTTPS becomes relatively low cost. Supporting HTTPS has become an urgent task by Web hosting companies.

In the Web hosting service based on the multi-tenant architecture [7], single server process group need to manage a large number of hosts [16] to provide at a low price by reducing the hardware cost and operation cost by accommodating hosts with high integration. The term single server process group means that a server process was shared by a large number of hosts, not activated a process for each host on the highly integrated multi-tenant architecture. The number of server processes does not depend on the number of hosts, but rather depends on the Web server implementation, which may invoke hundreds of server processes at the startup.

To communicate with HTTPS, existing Web server software needs to load the secret key paired with the server certificate for each host at the server process startup [5]. However, the highly integrated multi-tenant architecture don't take advantage of reducing the hardware cost and operation cost with the existing mechanism. The reason for this is that if hosts are accommodated in a high degree of integration, it takes a lot of time to start the server process by reading a large number of server certificates and secret keys, and the memory usage of the server process increases depending on the number of hosts.

In this paper, we propose large-scale certificate management mehtod which effectively and efficiently reduces the memory consumption of the Web server process by dynamically acquiring corresponding server certificate and secret key data each request in a highly integrated multi-tenant Web server.

The proposed method does not preload a server certificate and a secret key at the Web server process startup, but rather dynamically loads the server certificate and the secret key from a database each request based on the requested hostname or IP address/port during SSL/TLS handshake.

We implement the new feature of ngx_mruby [15]that can handle the loading phase of certificates. ngx_mruby [15] is a fast and memory-efficient Web server extension mechanism scripting with mruby [9] for nginx [10]. Server certificates and secret keys are stored in Redis [17], which is a kind of KVS [4], and the certificate and secret key corresponding to a hostname are acquired by the mruby code.

The rest of the paper is organized as follows. First, we summarize the tasks for constantly using HTTPS in a highly integrated multi-tenant Web server in section 2. We describe the architecture and implementation of the proposed method in section 3. In section 4, we quantitatively verify the problems of the existing method and evaluate the effectiveness of the proposed method. In section 5, we evaluate the proposed method in production of hosting service of our employer for one month. Section 6 concludes the paper.

2. Related works

Web hosting service [12] is a typical application service of highly integrated multi-tenant architecture. The Web hosting service shares server resources among multiple hosts and provides an HTTP server function for each hostname. In the Web hosting service, the function that is identified by a Fully Qualified Domain Name (FQDN) and serves the corresponding content is called a host. In this paper, we call a multi-tenant architecture which can accommodate tens of thousands of hosts as a highly integrated multi-tenant architecture [13].

The highly integrated multi-tenant architecture adopts the virtual host method [19] which processes multiple hosts by a single server process group. Popular Web server software such as Apache httpd [18] and nginx can handle multiple hosts by a single server process group using the virtual host method like VirtualHost configuration of Apache httpd.

In the existing server certificate management of web servers, the Web server loads the certificate associated with the hostname into a memory at a Web server process startup [5]. The Web server reads out the certificate corresponding to an IP address/port or a hostname from memory at each SSL/TLS handshake and starts the HTTPS session. This method can process at high speed during the SSL/TLS handshake since the certificate is loaded in advance to the memory.

A highly integrated multi-tenant architecture needs to make the configuration and adopts the process model independent from the number of hosts with the virtual host method, since the architecture manages a large number of hosts. In operation in a production environment, a single server process may accommodate more than several tens of thousands of hosts.

The existing method needs to load a large number of certificates and secret keys in the memory at the server process startup. In the system configuration of a reverse proxy for the TLS termination, the system needs to first perform TLS communication on the reverse proxy to the hostnames of all the hosts accommodated in a large number of hosting servers. The reverse proxy must manage configurations and certificates on hundreds of thousands of hostnames. In that case, as the number of server certificate increase, the loading time of configurations and certificates data at the server process startup greatly increases. Also, the memory usage of the server process greatly increases. These increasing resources may cause a serious problem.

3. Proposed method

3.1. Large-scale Certificate Management

Our proposed method meets the following three requirements, which are essential in the highly integrated multi-tenant architecture which requires maximization of balance between computer resource, performance efficiency and efficiency of the system operation cost while solving the problem described in section 2:

1) To support the Server Name Indication (SNI) extension to accommodate hosts;
2) To avoid loading all Web server certificates for faster startup of the server processes; and
3) To ensure that the memory usage of the Web server process is independent of the number of hosts, by dynamically loading the associated server certificates during each SSL/TLS handshake.

SNI [1] in the requirement (1) is an extended specification of SSL/TLS. Serving multiple HTTPS servers by a small number of IP address is critical to provide at a low price under the cost constraint of the highly integrated multi-tenant architecture. SNI allows selective use the server certificate in the hostname, since SNI tells the unencrypted hostname to the server during SSL/TLS handshake. SNI is commonly used to accomodate a large number of hosts virtually with a single server process group and a single IP address in the highly integrated multi-tenant architecture.

We propose the method that the server certificate and secret key of the request are dynamically loaded from data-store like database, file system or API, based on the requested hostname during the SSL/TLS handshake when an HTTPS request comes to the Web server, with SNI in the requirement (1). The dynamic certificate loading meets the requirement (3).

In the proposed method, the startup time of the web server process does not depend on the number of hosts and memory usage, and does not increase, which meets the requirement (2). The dynamic certificate loading does not need to load a large number of server certificates beforehand at the startup. Even if the number of certificates increases, the proposed method does not cause the problem of taking a long time to reload the server process when a configuration change occurs, since the startup speed of the server process is not slowed down by dynamic certificate loading each request. Also, adding more hosts by changing the configuration does not require the server process reloading, since the proposed method can dynamically analyze the certificate location from the hostname.

By aggregating data in databases and caches that can communicate via TCP, Our proposed method ensures availability and performance of the service system by increasing the number of Web servers using a scale-out model as the number of HTTPS requests increases. Our system can easily prepare HTTPS environment by linking the databases that store the user data like the hostname, certificates, and secret keys data, under TLS option contacts with customers in a production environment.

3.2. Implementation

In our proposed method implementation, we used ngx_mruby which can extend nginx scripting with mruby and process at high speed with less memory usage. Also, the OpenSSL [11] version 1.0.2 or later have a function that calls back an extension function of SSL/TLS handshake behavior such as custom loading server certificates and secret keys during SSL/TLS handshake, `SSL_CTX_set_cert_cb()`. By making this function executable from ngx_mruby, the callback function using mruby during SSL/TLS handshake on nginx [14] can be written by ngx_mruby, which enables the server administrator to easily implement the dynamic certificate loading algorithm for the various systems.

Figure 1 shows an implementation example of dynamically loading server certificates and secret keys at SSL/TLS handshake using ngx_mruby. The server certificates are stored in Key-Value Store (KVS) such as Redis for the requested hostname of each request. The certificate_data and the certificate_key_data methods pass data of a certificate and a private key themselves, not a file.

```
server {
  listen             443 ssl;
  server_name        _;
  ssl_protocols      TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers        HIGH:!aNULL:!MD5;
  ssl_certificate    /path/to/dummy.crt;
  ssl_certificate_key /path/to/dummy.key;

  mruby_ssl_handshake_handler_code '
    ssl = Nginx::SSL.new
    host = ssl.servername
    redis = Redis.new "127.0.0.1", 6379
    ssl.certificate_data = redis["#{host}.crt"]
    ssl.certificate_key_data = redis["#{host}.key"]
  ';
}
```

Figure 1. KVS-based Configuration Example of Dynamic Server Certificate Management.

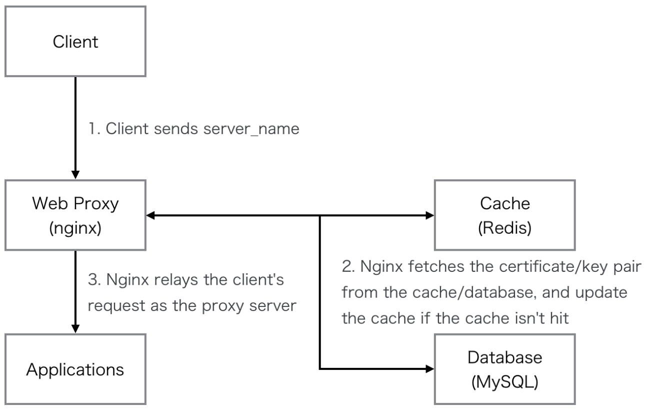
Client
1. Client sends server_name
Web Proxy (nginx)
Cache (Redis)
3. Nginx relays the client's request as the proxy server
2. Nginx fetches the certificate/key pair from the cache/database, and update the cache if the cache isn't hit
Applications
Database (MySQL)

Figure 2. System Example of Dynamic Server Certificate Management.

In Figure 2, a design example in operation of production is described. The administrator saves the server certificate and secret key data in the database. When nginx receives an HTTPS request, it loads the server certificate and secret key from the database via ngx_mruby scripts and establishes the SSL/TLS session. The proposed method implementation temporarily stores cache data using KVS such as Redis which enables high-speed access to the data to reduce the connection cost to the database for each request.

Even if the number of servers is increased for availability or performance, our proposed method implementation can easily share server certificate data via TCP connection with database or cache server. If the network latency from the Web server to the cache server becomes a performance problem, the in-memory cache can also be used for mitigating performance degradation. The implementation of ngx_mruby version 1.16.0 has already been published as OSS as of February 2016.

4. Evaluation and consideration in the production environment

To confirm the effectiveness of the proposed method, in the existing problem described in section 2, we clarify the problem of the time of startup (preloading) of the existing method from the experiment. Next, we compare the existing method (preloading) with the proposed method (dynamic loading) that saves the server certificate and secret key data in Redis and acquires data from the file and Redis for each SSL/TLS handshake. Table 1 shows the experiment environment.

TABLE 1. EXPERIMENTAL ENVIRONMENT.

	Specification
CPU	Intel Xeon E5-2620 v3 2.40GHz 24thread
Memory	32GBytes
Server	NEC Express5800/R120f-2E

TABLE 2. RESULT OF STARTUP TIME BY EXISTING METHOD.

item	value
real time	42.662 sec
system CPU usage time	37.280 sec
user CPU usage time	5.387 sec
virtual memory size (VSZ)	3207592 KBytes
physical memory size (RSS)	3175912 KBytes

4.1. Verification of memory usage and startup time of existing methods

In the table 1 environment, we verify the problem described in section 2 by using nginx version 1.11.13. We generated server certificates and secret keys of the key length of 4096 bits for one hundred thousand hosts by the `openssl` command for each host configuration of nginx and measured the memory usage and startup time of nginx server processes.

The master process of nginx initially loads all the server certificate data and copies the worker process for request processing by the fork() system call after the initial processing of master process is completed. In this experimental environment, the number of worker process is 24 processes, which is the number of logical core of the CPU using configuration setting parameter `worker_processes auto`.

We measured CPU metrics using the Linux time command until all the worker processes have completed the initialization. We also determined memory usage size of a certain worker process, arbitrarily picked up from whole worker processes, using the Linux `ps` command; we adopted both fields on virtual memory (VSZ) and physical memory (RSS). We later discuss whole memory usage size comparison of the existing method and the proposed method at section 4.2.

Table 2 shows the result. Loading of server certificates depends on the performance per core since a single master process uses only one CPU at first. In the era when increasing the CPU usage efficiency by the number of cores, shortening this processing time is difficult. There were no notable points about the usage time of the user CPU and system CPU.

Memory usage size of a worker process acquired from both virtual memory (VSZ) and physical memory (RSS) fields from `ps` command are about 3 GBytes as well.

TABLE 3. EXPERIMENTAL RESULT OF PROPOSED METHOD.

Nsc	proposed method dynamic loading(req/sec)	existing method preloading(req/sec)
10	171456.60	171914.98
100	172383.84	172758.28
500	172714.81	173631.06
1000	171872.24	173272.53

Physical machines in recent days often have a large size of physical memories; the size is usually over tens or thousands of gigabytes. Therefore, it is no problem if a process occupies even 3 GBytes or so memory space.

4.2. Performance evaluation of the proposed method

We evaluated the performance of the proposed method. We used nginx which is linked ngx_mruby module as web server software for evaluation.

We set the nginx configuration of both the existing method and the proposed method to different ports. The configuration of the proposed method is a configuration to load the server certificate and secret key with the requested hostname as the key during SSL/TLS handshake. The configuration of the existing method is a configuration of preloading server certificates at the startup time of the Web server process. Both configurations fixed the cipher suites on the server side to prevent the SSL/TLS session cache to maximize the impact during SSL/TLS handshake.

In the case of configuration to load multiple certificates, the calculation of loading certificates amount required for the existing method is O(1) since the method treats correspondence between hostname and certificate as a hash algorithm in nginx. Also, the calculation of loading certificates amount required for the proposed method is O(1) since the method also acquires the certificate from the KVS using the requested hostname as the key. From the above, we have determined that it is necessary and sufficient for the evaluation in this experiment to set one certificate to be read in the configuration of the existing method and the proposed method.

We used HTTPS benchmark software wrk [20] which supports multi threading to compare performance. While changing the number of simultaneous connections, we sent 5 million requests as the total and measured the number of requests per second.

We adopted TLSv1.2 of nginx configuration that enables TLS version 1.2. We also adopted ECDHE-RSA-AES128-GCM-SHA256 [6] as the cipher suites from which Mozilla recommends. The content requested uses index.html of 612 bytes enclosed with nginx by default.

Table 3 shows the result. Nsc in the table is an abbreviation for Number of simultaneous connections. In the experimental results, we observed that there is almost no performance difference between preloading and dynamic loading method.

TABLE 4. PRODUCTION SERVER SPECIFICATIONS.

	Specifications
CPU	Intel Xeon CPU E5-2430 v2 2.50GHz 12thread
Memory	32GBytes
Server	NEC Express5800/E120e-M

We considered that the process of dynamically loading a certificate is almost negligible because encryption and compound processing in SSL/TLS handshake is very large. When the number of simultaneous connections is thousand, both the existing method and the proposed method are somewhat degraded in performance, but this result is within the error range since the difference is also less than 1%.

In the preloading method, there is a problem that the amount of memory usage increases as the number of hosts increases in the highly integrated multi-tenant architecture. On the other hand, the memory usage at startup is very small since the dynamic loading method does not require the initializing process to store all SSL/TLS configuration data such as server certificate in memory at startup.

4.3. Evaluation in production

We applied the proposed method to a hosting service production of our employer and evaluated it on the operation in the production.

The hosting service adopted the existing method (preloading) that loads the certificate at the time of startup using Apache httpd before applying the proposed method.

As the evaluation method, We measured the transition of the total number of certificates the number of requests processed per second, the CPU usage rate, and memory usage for one month from March 4 to April 4 of 2017 when the existing preloading method was adopted. We compared the transition with the same kind of measured values during the one month from July 22 to August 22 of the same year after applying the proposed method (dynamic loading).

We have developed a production environment based on Figure 2. Also, the server hardware which was adopted in the preloading method and the dynamic loading method has the same specifications.

Table 4 shows server specifications.

Figure 3 shows the transition of the number of certificates for one month. Figure 4 indicates the transition of the number of requests processed per second. Figure 5 shows the transition of CPU utilization of the server. Figure 6 shows the transition of the memory usage of the server.

Figure 3 indicates that the number of certificates increased by about 400 in one month in the preloading method of the existing method. In that case, as described in the 4.1 section, Figure 6 shows that the usage has increased by about 1 GBytes since the preloading method reads all certificates at startup. On the other hand, the dynamic loading method of the proposed method of Figure 3 shows that the number of certificates has increased by more than 10,000 in one month. The increase in the number of certificates was due to a newly-provided free HTTPS certificate service of our

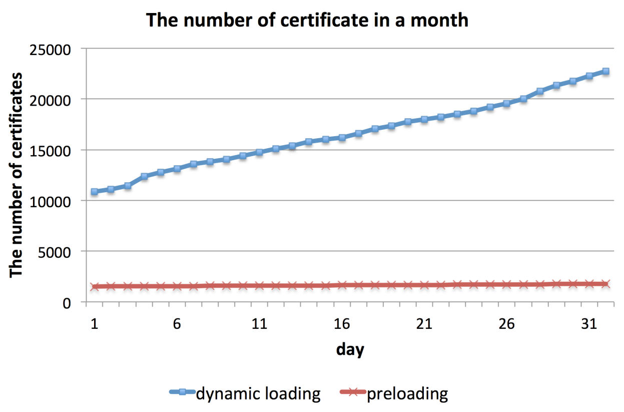

Figure 3. The number of certificates in a month.

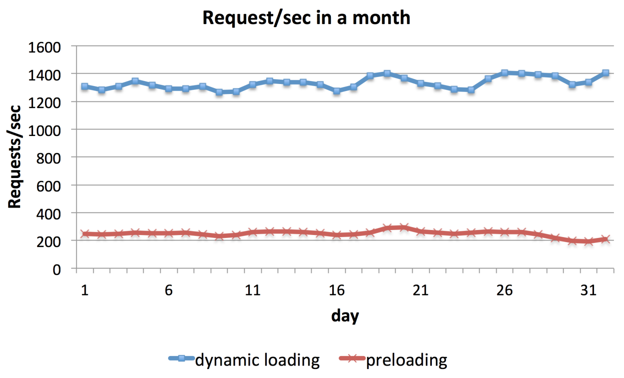

Figure 4. Request per second in a month.

employer. The number of certificates of servers processing by dynamic loading method is 10 times to 15 times that of preloading method, and the number of requests in second is more than six times from Figure 4.

However, as shown by Figure 5 and Figure 6, the transition of CPU usage and memory usage are less than the server which was processing with the existing preloading method.

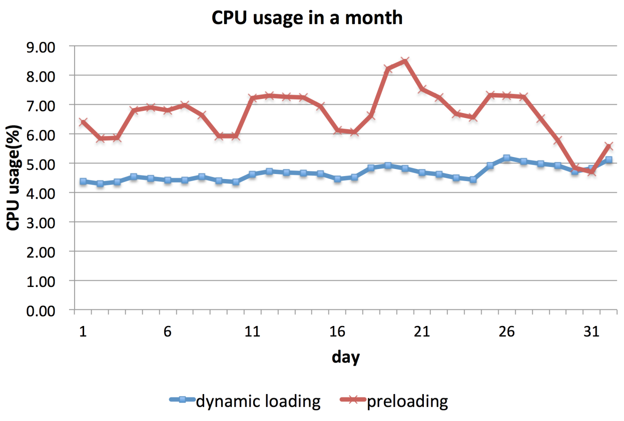

Figure 5. CPU usage in a month.

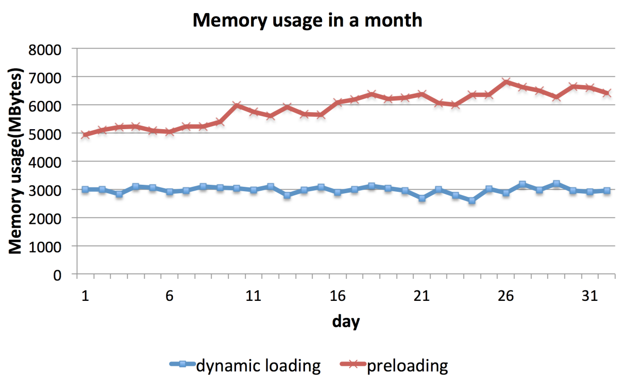

Figure 6. Memory usage in a month.

Regarding the problem of increasing the memory usage depending on the number of certificates in the preloading method, as shown by Figure 6, the proposed method does not significantly increase in memory usage. The memory usage can be greatly reduced by the proposed method. This is because it is not necessary to read a certificate of a domain without access by loading only the certificate corresponding to the domain HTTP requested from clients.

It is possible to reload, graceful restart command by nginx, the server process online without failing the request since the time required for reloading the configuration of the Web server process is greatly shortened. By shortening this time, the proposed method can freely release the memory usage, and the memory usage as a whole can be reduced.

Depending on the implementation of the web server, normally the graceful restart function to reload the configuration without missing the request completes in seconds. In the existing method, when the number of certificates to be load at the time of activation increases, it takes time to reload for several tens of seconds and even several minutes. Even if graceful restarting, the service was stopped due to the request timeout.

4.4. The discussion of the evaluation result

Figure 3 and Figure 6 indicate that the number of certificates increases by about 400 in one month in the preloading method and the memory increases by about 1 GBytes. The breakdown of the memory increase amount per certificate is the configuration of the host, the data of the certificate and the secret key, and the memory usage amount used when the Web server processes HTTPS requests. In other words, when trying to process 20000 server certificates by the preloading method, it is calculated that 50 GBytes of additional memory is required. The proposed method can process 20000 certificates with about 3 GBytes, so that the resource usage can be greatly improved.

When using the server equipped with 32GBytes of memory in table 4, if the number of certificates reaches 200000 or more in the future, the existing method requires memory over 500 GBytes from the viewpoint of the memory usage mentioned above. In other words, in the existing method, more than 15 servers with 32 Gbytes memory installed are required.

On the other hand, in the proposed method, Figure 6 indicates that memory usage hardly depends on the number of certificates. From the viewpoint of memory usage, this result shows that even if the number of certificates reaches 200000, even one server can process it. In the proposed method, in the situation where future HTTPS communication becomes commonplace, the number of servers can be greatly reduced.

In the existing method, the time for reloading the server process becomes longer as the number of certificates increases, so that there was a problem that the service stoppage time due to reloading becomes long at loading a new configuration or registering a new certificate.

However, in the proposed method, the proposed method can reload the process in a short time and shorten the service downtime since the certificate is not loaded at server process startup,

Also, applying a configuration of a new certificate of other host does not need the reloading server process. The proposed method can dynamically analyze where the certificate is located from the hostname like file path including a hostname or database with a hostname as key. When adding a new certificate, if the server administrator registers certificate data in the database, the proposed method can apply HTTPS to the existing site without reloading the server process.

Therefore, the service stoppage time can be shortened as a whole by the proposed method, and it becomes possible to adopt a system configuration easy to operate.

5. Conclusion

As RFC adoption of HTTP/2 protocol is required on HTTPS, the existing sites need to support HTTPS. The existing method takes time to start up, since a highly integrated multi-tenant Web server needs to load a large number of server certificates at a server process startup.

The proposed method dynamically loads the server certificate and secret key corresponding to the requested hostname using SNI during SSL/TLS handshake and then communicates via HTTPS. By using the proposed method, the server can communicate via HTTPS without loading a large number of server certificates at startup. Moreover, the cost of dynamically loading a certificate is low compared with the cost of CPU usage time from the whole SSL/TLS handshake, and the experimental results show that performance does not cause a problem in practical use.

As a result of introducing the proposed method to the production environment of hosting service, resource usage can be greatly reduced compared with the existing method, and the proposed method is sufficiently effective on the operation of the production environment.

Furthermore, even if the performance becomes insufficient due to the processing of HTTPS, the proposed method can easily scale up servers using the scale-out model by the centralized management of the server certificate data with a reverse proxy put in front of the HTTPS servers.

We conclude that the proposed method is one promising method of a practical system design for supporting HTTPS of highly integrated multi-tenant architecture.

References

[1] Eastlake D, Transport Layer Security (TLS) Extensions: Extension Definitions, RFC 6066, 2011.

[2] Ferdman M, Adileh A, Kocberber O, Volos S, Alisafaee M, Jevdjic D, Falsafi B, Clearing the clouds: a study of emerging scale-out workloads on modern hardware, ACM SIGPLAN Notices, Vol. 47, No. 4, pp. 37-48, March 2012.

[3] Internet Security Research Group (ISRG), Let's Encrypt - Free SSL/TLS Certificates, https://letsencrypt.org/.

[4] Han J, Haihong E, Le G, Du J, Survey on NoSQL database. 2011 6th International Conference on Pervasive computing and applications (ICPCA), pp. 363-366, October 2011.

[5] Let'sEncrypt Community Support, Apache Module mod_vhost_alias & LE, https://community.letsencrypt.org/t/apache-module-mod-vhost-alias-le/9476.

[6] Mozilla Project, mozilla wiki Security/Server Side TLS, https://wiki.mozilla.org/Security/Server_Side_TLS.

[7] Mietzner R, Metzger A, Leymann F, Pohl K, Variability Modeling to Support Customization and Deployment of Multi-tenant-aware Software as a Service Applications, the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, pp. 18-25, May 2009.

[8] Naylor D, Finamore A, Leontiadis I, Grunenberger Y, Mellia M, Munafò M, Steenkiste P, The cost of the S in HTTPS, the 10th ACM International on Conference on emerging Networking Experiments and Technologies (CoNEXT '14), pp. 133-140, ACM, December 2014.

[9] NPO mruby forum, http://forum.mruby.org/.

[10] Nginx, Nginx, http://nginx.org/ja/.

[11] OpenSSL Software Foundation, OpenSSL, https://www.openssl.org/.

[12] Prodan R, Ostermann S, A Survey and Taxonomy of Infrastructure as a Service and Web Hosting Cloud Providers,10th IEEE/ACM International Conference on Grid Computing, pp. 17-25, October 2009.

[13] Ryosuke M, Studies on Highly Integrated Multi-Tenant Architecture for Web Servers, https://dx.doi.org/10.14989/doctor.k20590, Kyoto University, Ph.D. thesis, 2017.

[14] Ryosuke M, ngx_mruby: Support ssl_handshake handler and dynamic certificate change, https://github.com/matsumotory/ngx_mruby/pull/145.

[15] Ryosuke Matsumoto, Yasuo Okabe, mod_mruby: A Fast and Memory-Efficient Web Server Extension Mechanism Using Scripting Language, IPSJ Journal, Vol.55, No.11, pp.2451-2460, Nov 2014.

[16] Ryosuke Matsumoto, Masashi Kawahara, Teruo Matsuoka, Improvement of Security and Operation Technology for a Highly Scalable and Large-scale Shared Web Virtual Hosting System, IPSJ Journal, Vol.54, No.3, pp.1077-1086, Mar 2013.

[17] Sanfilippo S, Noordhuis P, Redis, https://redis.io/.

[18] The Apache Software Foundation, Apache HTTP Server, http://httpd.apache.org/.

[19] The Apache Software Foundation, Apache Virtual Host documentation, http://httpd.apache.org/docs/2.2/en/vhosts/.

[20] Will Glozer, wrk - a HTTP benchmarking tool, https://github.com/wg/wrk.