Prototyping and Evaluation of Virtual Cache Server Management Function for Distributed Web System

Akihiko Horiuchi and Keizo Saisho
Kagawa University
2217-20 Hayashi-cho, Takamatsu 761-0396, JAPAN
Email: s14g481@stmail.eng.kagawa-u.ac.jp, sai@eng.kagawa-u.ac.jp

***Abstract*—We develop a distributed Web system that adjusts the number of virtual cache servers in the Cloud according to load of them to keep responsiveness and reduce running costs. A method of monitoring load of Web servers, and a algorithm to adjust the number of virtual cache servers are discussed in previous researches. This paper describes examination of time for bootup and shutdown virtual cache server, and prototyping a virtual cache server management function that boots up and shuts down virtual cache server using libvirt which provides common API for managing virtual platforms. From results of experiments, we confirm that this function is possible to boot up and shut down virtual cache servers according to load.**

***Index Terms*—Distribyted System, Load Balancing, Web Server, Cache Server, Auto-scaling, Cloud**

I. Introduction

This research aims to realize the distributed Web system that adjusts the number of cache servers in the Cloud according to load of them to keep responsiveness and reduce running costs. It is called auto-scaling that general feature of cloud computing services, and scalability by auto-scaling is big advantage of the Cloud [1]. Fig. 1 shows our distributed Web system, we develop a extended load balancer that monitors load of a original Web server (origin server) and virtual cache servers (VC-Server) provided by the Cloud, boots up and shuts down a VC-Server according load, and distributes requests to these servers (working servers). The extended load balancer boots up a new VC-Server (scale-out) when average load of working servers exceeds upper threshold and shuts down any VC-Server (scale-in) when average load falls lower threshold.

In previous research [2], effectiveness of the load monitoring function and the destination setting function was confirmed. Transition of the number of working servers and response time on clients were evaluated when load was changed. In this paper, prototyping and evaluation of the "VC-Server Management Function" that boots up and shuts down VC-Server for auto-scaling are described. As a result, we confirm that this function is possible to boot up and shut down VC-Servers according to load.

II. Related Works

There have been many studies of distributed Web system for the Cloud. Chieu T.C. et al. proposed a scaling scenario to address the dynamic scalability of Web applications on the system constructed with a front-end load balancer and Web servers in virtual machine instances in the Cloud [3]. Instances are created from "golden" virtual image template, and added and removed according to the number of active sessions using static upper and lower thresholds. Our distributed Web system is different in that it aims to use origin server and cache servers. For example, origin server provides all contents including secure contents such as accounts and payment, and cache servers only provide static contents such as images and style stheets. Our method uses dynamic lower threshold described in section IV.

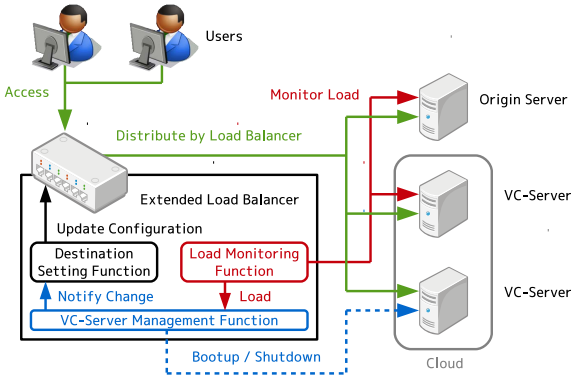

Fig. 1. Structure of the distributed Web system

Usual load balancers can distribute requests to Web servers previously designate and have no function that dynamically changes the number of Web servers according to load and allocates requests to them. Elastic Load Balancing [4] by Amazon is one of Load Balancer as a Service (LBaaS) [5]. Although it changes the number of instances according to amount of requests, it can use only on Amazon Web Service. Xu G. et al. introduce a load balance model for the public cloud based on geographical regions [6]. In this study, the top-level main controller selects idle region and allocates job to load balancer in that region. The target of this study is a single public cloud service. In contrast, our distributed Web system aims to use any virtual machines provided by multiple cloud services and their regions.

III. Outline of Distributed Web System

The extended load balancer shown in Fig. 1 consists of the following functions.

- **Load Monitoring Function**
  The load monitoring function monitors load of the origin server and VC-Servers. We use Apache 2.4 [7] as a Web

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.26

server software. This functions periodically measures the ratio of the current number of Web server processes againt the maximun number of processes (Operating Ratio, OR), and calculates average of OR of working servers requests (Average OR, AVGOR). The extended load balancer uses AVGOR as load value.

- **VC-Server Management Function**
  The VC-Server management function decides the number of required VC-Servers according to load measured by load monitoring function, and boots up / shuts down VC-Server. We plan to use API provided by virtualization platforms such as OpenStack [8] and cloud services such as Amazon EC2 [9] to boot up and shut down VC-Server.
- **Destination Setting Function**
  The destination setting function updates a configuration of load balancer to change destination of requests to start and stop allocation to booted up and shutted down VC-Server, respectively. We use IPVS [10] as a load balancer which implemented inside the Linux kernel. Web servers are assigned to weight and IPVS allocates requests to them according to their weight. When weight of allocation is set 0, IPVS allocates no requests. This function changes weight of allocation using IPVS configuration command.

IV. Design and Prototyping of VC-Server Management Function

This section describes design and prototyping of the VC-Server management function. The VC-Server management function boots up and shuts down VC-Server at the following cases.

- **Scale-out**
  When load exceeds threshold of scale-out (Th_{high}), the VC-Server management function boots up a new VC-Server. After waiting the VC-Server starts service, this function notifies the destination setting function of bootup it to start allocation to it.
- **Scale-in**
  When load falls threshold of scale-in (Th_{low}), the VC-Server management function shuts down a latest booted up VC-Server. After notifing the destination setting function of shutdown that VC-Server to stop allocation to it, this function shuts down it.

Th_{high} is given by system manager and Th_{low} is decided according to the number of VC-Servers. If the number of VC-Servers decreases from n to $n-1$, average load becomes $n/(n-1)$ times. If this value is lower than Th_{high} , average load is still below Th_{high} theoretically when one VC-Server is shuted down. Therefore, Th_{low} can be calculated by formula (1). m is margin to suppress fluctuation of load.

$$Th_{low} = Th_{high} \times \frac{n-1}{n} - m \quad (1)$$

The current our distributed Web system uses KVM (Kernel-based Virtual Machine) hypervisor that standardly equipped with the Linux kernel. Since KVM has no API, we prototype the VC-Server management function using libvirt [11] which provides common API for managing virtualization platforms. libvirt is usually used to manage virtual machines in several hypervisors and build cloud environments [12]. It consists of API library for several programming languages and the daemon (libvirtd) that receives operations from API on hypervisors.

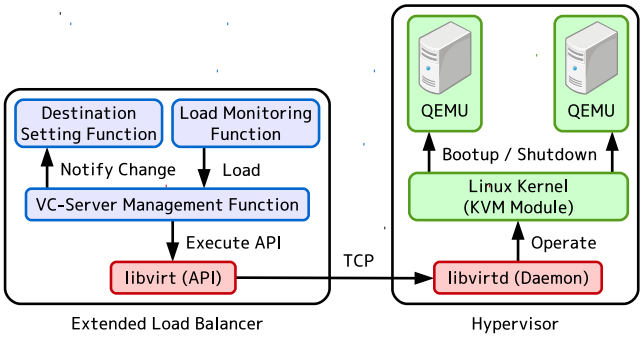

Fig. 2. Structure of prototype of the VC-Server management function

Fig. 2 shows the structure of prototype of the VC-Server management function. This function uses libvirt to boot up and shut down VC-Server one-by-one by the following steps.

1) Open TCP connection to libvirtd using "NewVirConnection" function with hypervisor's IP address.
2) Specify a virtual machine that operated using "LookupDomainByName" function with VC-Server's domain name.
3) Execute "Create" and "Shutdown" function when bootup and shutdown VC-Server, respectively.
4) Close TCP connection to libvirtd using "CloseConnection" function.

Fig. 3 shows a sample configuration of the extended load balancer. Hypervisor's IP addresses and VC-Server's domain names are set. Prototype of the VC-Server management function is able to manage multiple hypervisors.

V. Examination of Bootup and Shutdown Time

In this section, bootup and shutdown time are examined. Bootup / shutdown time is the time between the VC-Server management function starting bootup / shutdown VC-Server and VC-Server starting / stopping service. We use VC-Server that allocated 1 CPU core and 1GB memory, and its operating system is Ubuntu Server 14.04 [13]. Table I shows results of 10 times measured.

As shown in Table I, bootup time is about 10 seconds and shutdown time is about 3 seconds. We examine whether bootup VC-Server can start service at this bootup time with large amount of requests or not. Shutdown time is also examined. As a result, many time-outs occur. We consider that many other service programs also start up and they disturb processing requests at booting up VC-Sercer. In contrast, VC-Server that stopped allocation still returns responses at shutting down VC-Server. We try to wait 20 and 30 seconds. No time-out occurs

```
{
  "hvs": [                              // Hypervisors
    {
      "host": "192.168.11.20",          // HV's IP Address
      "vms": [                          // Virtual Machines
        {
          "name": "cache-server-1",     // VM's Domain Name
          "host": "192.168.11.21"       // VM's IP Address
        },
        {
          "name": "cache-server-2",     // VM's Domain Name
          "host": "192.168.11.22"       // VM's IP Address
        }
      ]
    },
    {
      "host": "192.168.11.30",          // HV's IP Address
      "vms": [                          // Virtual Machines
        {
          "name": "cache-server-3",     // VM's Domain Name
          "host": "192.168.11.31"       // VM's IP Address
        },
        {
          "name": "cache-server-4",     // VM's Domain Name
          "host": "192.168.11.32"       // VM's IP Address
        }
      ]
    }

  ]
}
```

Fig. 3. A Sample configuration of the extended load balancer

TABLE I
RESULTS OF BOOTUP AND SHUTDOWN TIME

	Bootup time	Shutdown time
1	9.724	3.004
2	9.726	3.031
3	9.512	3.049
4	9.556	3.003
5	9.265	3.029
6	9.681	3.067
7	9.432	3.021
8	9.494	299
9	9.440	296
10	9.767	3.050
Average	9.767	3.025

with 30 seconds. Therefore, time between starting bootup VC-Server and starting allocation (allocation start time, ALLC-START) is set 30 seconds. Time between stopping allocation and starting shutdown VC-Server (allocation stop time, ALLC-STOP) is set 30 seconds too.

VI. EVALUATION OF VC-SERVER MANAGEMENT FUNCTION

We evaluate prototype of the VC-Server management function. Fig. 4 shows experiment environment that includes the extended load balancer, the origin server, 8 VC-Servers and 9 clients. DokuWiki [14] runs on the origin server. We use copy instance of the origin server as VC-Servers bacause cache mechanism suitable for our system is now developing. Each client accesses to the load balancer using Apache Bench [15]. The number of simultaneous accesses is set to 100, so that the maximum number of simultaneous accesses is 900 (100×9).

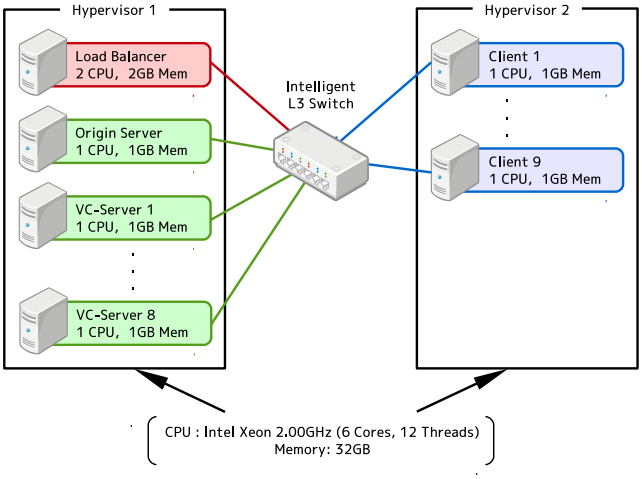

Fig. 4. Experiment environment

In order to examine load and the number of working servers, the number of simultaneous accesses to the load balancer is stepwise changed. The scenario of experiment is shown in below.

1) Start with no accesses.
2) Add 1 client every 30 seconds (add state).
3) After all clients are added, keep all clients accessing for 60 seconds.
4) Remove 1 client every 30 seconds (remove state).
5) End when no accesses.

Fig. 5 and Fig. 6 show results without and with the VC-Server management function, respectively. In without case, VC-Servers are always running and change destination of requests only. In Fig. 5, AVGOR increases according to the number of simultaneous accesses, and VC-Server is added to destination of requests immediately when AVGOR exceeds $Th_{high} = 0.8$. AVGOR never reaches 1.0 by a dissolution of load. In contrast, AVGOR dose not decrease immediately because a new VC-Server can be available at 30 seconds after starting bootup as shown in Fig. 6. Therefore, AVGOR reaches 1.0 in some cases. The number of working servers is 5 at the maximum simultaneous accesses compared with Fig. 5 (the number is 7).

In remove state in with case, the number of VC-Servers is smaller than that in without case in spite of same simultaneous accesses. The maximum number of running virtual servers is 8 (1 load balancer + 7 web servers) in without case, and this number is more than the number of CPU cores (6) of the hypervisor. In contrast, the maximum number of running virtual servers is 6 (1 + 5) in with case, and this number equals the number of CPU cores. Therefore, the performance of VC-Server in with case is thinkable better than that in without case.

In order to investigate influence of ALLC-START on AVGOR, we check ORs of each Web server (the origin server and VC-Servers). Fig. 7 and Fig. 8 shows results of without and

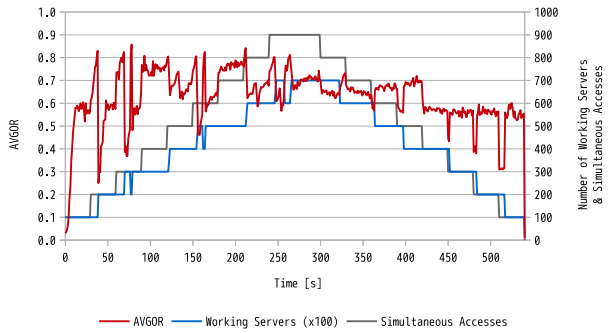

Fig. 5. without the VC-Server management function

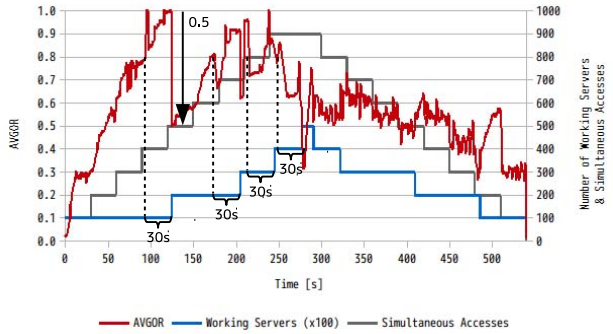

Fig. 6. With the VC-Server management function

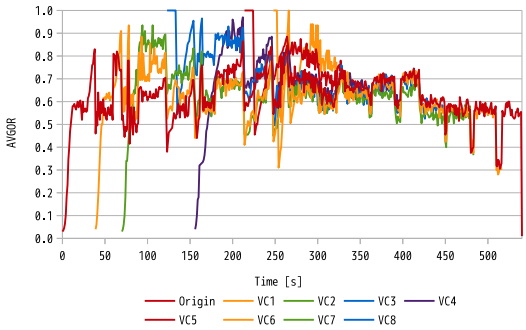

Fig. 7. ORs without the VC-Server management function

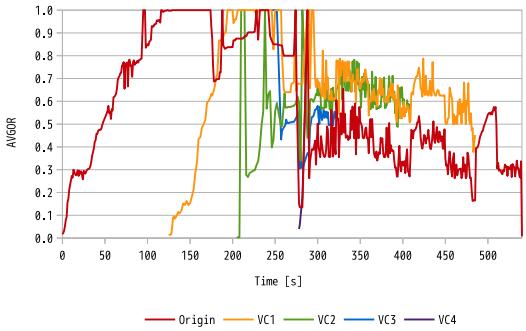

Fig. 8. ORs with the VC-Server management function

with case, respectively. In without case, accesses is distributed to each Web server almost equality. In with case, however, high OR causes by convergence of accesses on current working servers. High OR is kept until allocated requests are processed because it is impossible to reallocate already allocated requests to a new VC-Server. This problem might degrade the quality of service such as increase of response time on clients. In the next section, we discuss a method to relax high OR during ALLC-START.

VII. Bootup Multiple VC-Servers

We consider bootup multiple VC-Servers to relax high OR during ALLC-START. In Auto Scaling by Amazon, users can define stepwise thresholds for auto-scaling and set the number of bootup / shutdown instances at each threshold [16]. We use increase ratio of AVGOR to deside the number of bootup VC-Servers.

A. Decision Algorithm for the Number of Bootup VC-Server Based on Increase Ratio

We define IR as increase ratio of AVGOR. The load monitoring function saves latest 10 AVGORs at every second and calculates IR from them when AVGOR exceeds Th_{high}.. IR is calculated by formula (2) that means dividing latest AVGOR ($AVGOR_L$) by 9 seconds before AVGOR ($AVGOR_{L-9}$).

$$IR = \frac{AVGOR_L}{AVGOR_{L-9}} \tag{2}$$

The number of bootup VC-Servers (N) is decided by formula (3). The number of bootup VC-Servers is rounded up IR to the nearest integer.

$$N = \lceil IR \rceil \tag{3}$$

B. Evaluation of Bootup Multiple VC-Servers

We implement decision algorithm in the VC-Server management function and experiment. The environment and scenario of experiment are same as described in section VI. The number of shutdown VC-Server is always 1.

Fig. **??** shows result of experiment using bootup multiple VC-Servers. 2 VC-Servers are booted up around 100 and 200 seconds. In previous experiment, AVGOR decrease about 0.5 at the first time of bootup VC-Server as shown in Fig. 6. In contrast, AVGOR decrease about 0.65 at the first time of bootup VC-Server in this experiment. It is larger than that of previous experiment. ABGOR more decreases when starting allocation by multiple VC-Servers bootup simultaneously. The maximum number of working servers is 7. This number equals that of without case (Fig. 5). At start of remove state, AVGOR in Fig. **??** is less than that in Fig. 6. So, the method is effective.

But the current version has the following problems.

- A VC-Server is shutted down soon after first bootup because AVGOR falls Th_{low}.

- The interval between starting allocation to 4th and 5th VC-Servers is too short. It must be longer than ALLC-START. It is thinkable that exclusive control is missed.
- VC-Servers are not shutted down after around 350 seconds. It is thinkable that error handling of libvirt is insufficient.

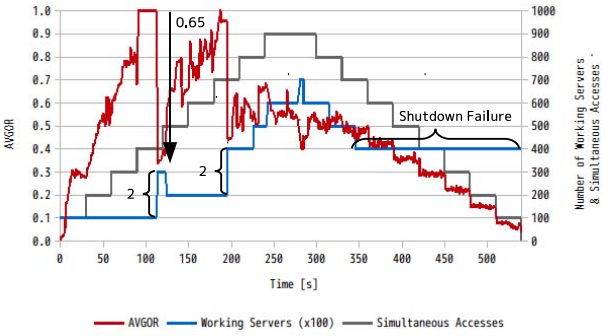

Fig. 9. With multiple VC-Servers bootup

VIII. Evaluation of Response Time

Response time would increase during ALLC-START caused by high OR. Similarly, response time would decrease when starting allocation to the new VC-Server. In order to verify the assumption, we examine response time on clients.

The environment and scenario of experiment are same as described in section VI. New virtual machine (1 CPU core, 1GB memory) for measuring response time is introduced on the hypervisor 2 and a measuring script written in Ruby runs on it. The script creates threads for measuring response time at every second. It accesses to the load balancer and measures response time.

The following 3 cases are examined to compare response time.

A) Without the VC-Server management function.
B) With the VC-Server management function.
C) With the VC-Server management function using bootup multiple VC-Servers.

Fig. 10, Fig. 11 and Fig. 12 show results of case A, B and C, respectively. Response time is moving average calculated from each 30 samples.

In Fig. 10, response time in case A (without the VC-Server management function) is small and stable. In contrast, in Fig. 11, response time in case B (with the VC-Server management function) suddenly rises when AVGOR exceeds Th_{high} = 0.8 (see Fig. 6). In Fig. 12, response time in case C (using bootup multiple VC-Server) is same tendency as that in case B. The peak response time in case C is, however, lower than that in case B. Although, allocation to one VC-Server is stopped soon, it decreases the peak time of response time.

The difference of case B and C shows that bootup multiple VC-Server is effective to reduce response time at high AVGOR, but we think it is insufficient. In order to reduce response time moreover, we plan to introduce mechanism that

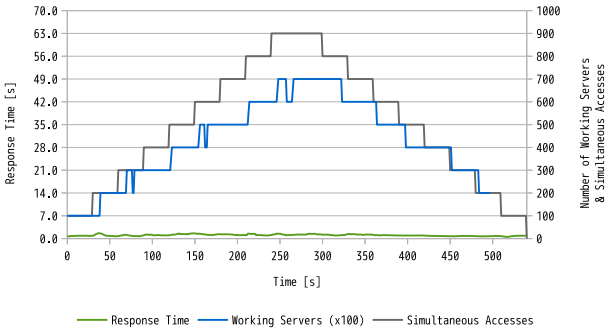

Fig. 10. Without the VC-Server management function

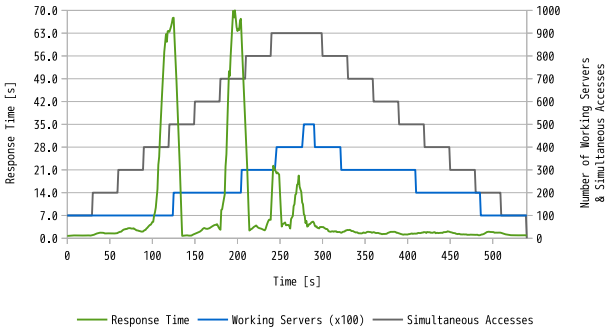

Fig. 11. With the VC-Server management function

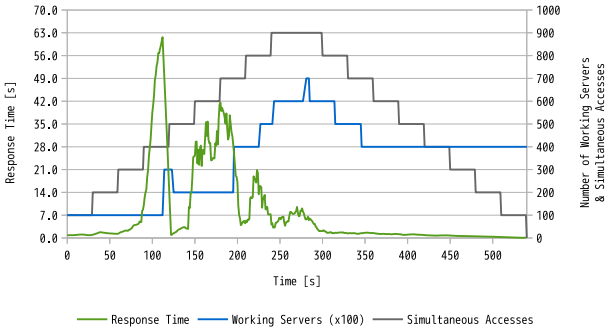

Fig. 12. Using multiple VC-Servers Booting

forecasts when AVGOR exceeds Th_{high} and boots up VC-Server adjusting to ALLC-START in advance.

IX. Conclusion & Future Works

In this paper, the VC-Server management function that boots up and shuts down VC-Server for auto-scaling is described. We prototype this function using libvirt. By the experiment with the virtual environment, it is confirmed that this function is possible to boot up and shut down VC-Server according to load. However, high OR causes by convergence of accesses on current working servers. In order to relax high OR during ALLC-START, bootup multiple VC-Servers is introduced. Results of experiments show that it is possible to join new

VC-Servers faster than bootup single VC-Server. Bootup multiple VC-Servers is effective to reduce response time at high AVGOR, but it is insufficient.

Future works include the following things.

- **Improvement of VC-Server Management Function**
  A mechanism that forecasts when AVGOR exceeds Th_{high} and boots up VC-Server adjusting to ALLC-START in advance will be implemented. Moreover, bugs of error handling of libvirt will be fixed.
- **Evaluation of Threshold of Scale-out**
  Th_{high} will be evaluated to find a optimum value. (Because current Th_{high} is hard-coded to 0.8.)
- **Experiment with More Hypervisors**
  In comparison with Fig. 6, Fig. 5 shows that AVGOR at the beginning of experiment suddenly increases. We consider that decreases of allocated machine power per virtual machine cause because all 9 Web servers run on single 6 cores hypervisor. Therefore, a case of running Web servers in multiple hypervisors will be experimented.
- **Practical Scenarios of Experiment**
  Practical senarios such as using access logs of working Web servers will be experimented.
- **Experiment in Private Cloud**
  Current virtual environment by KVM will be shift to private cloud such as OpenStack. VC-Server management function using API of private cloud will be implemented and evaluated.
- **Improvement of Distribution Algorithm of Load Balancer**
  It is impossible to reallocate already allocated requests when new VC-Servers are booted up. Therefore, an algorithm that waits allocating requests until new VC-Servers booted up when load of working servers is high will be implemented.
- **Forecast of Number of Requests and Optimal Number of VC-Servers**
  J. Jiang et al. proposed auto-scaling scheme that forecasts both the number of requests and resource demands [17]. In this study, request records of Web servers are used as history data, and analyze them to forecast the number of requests for the next time-unit (1 hour). We think it is too long and it is not possible to scale-out when the number of requests increases suddenly. It is necessary to imprement forecasting algorithm to cope with such a case.
- **Approach to Hetero Cloud Environment**
  The performance and cost of instances are different with cloud services. When a distributed Web system uses multiple cloud services, it is necessary to select better instances based on performance and cost, and allocate requests to them according to their performance. M. Mao et al. proposed a mechanism to dynamically scale cloud computing instances based deadline and budget information [18]. Similarly, it is necessary to scale VC-Servers based on performance and cost of instances, and response time in our distributed Web system.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 25330082.

REFERENCES

[1] L.M. Vaquero, L. Rodero-Merino and R. Buyya, "Dynamically scaling applications in the cloud," in *Newslett. ACM SIGCOMM Comput. Commun. Review*, vol. 41, Jan. 2011, pp 45-52
[2] A. Horiuchi and K. Saisho, "Development of Scaling Mechanism for Distributed Web System," in *Proc. 16th IEEE/ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, Jun. 2015, pp. 283-288.
[3] Chieu T.C., Mohindra A., Karve A.A. and Segal A, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in *Proc. IEEE 6th Int. Conf. on e-Business Engineering*, Oct. 2009, pp. 281-286
[4] Amazon Web Services Inc. (2015, Jul, 8). *Elastic Load Balancing - AWS*. Available: http://aws.amazon.com/elasticloadbalancing/
[5] Rahman M., Iqbal S. and Gao J., "Load Balancer as a Service in Cloud Computing," in *Proc. IEEE 8th Int. Symp. on Service Oriented System Engineering*, Apr. 2014, pp. 204-211.
[6] Xu G., Pang J. and Fu X., "A Load Balancing Model Based on Cloud Partitioning for the Public Cloud," in *Tsinghua Science and Technology*, vol. 18, Feb. 2013, pp. 34-39
[7] The Apache Software Foundation. (2015, Jul, 8). *The Apache HTTP Server Project*. Available: http://httpd.apache.org/
[8] The OpenStack project. (2015, Jul, 8). *OpenStack Open Source Cloud Computing Software*. Available: http://www.openstack.org/
[9] Amazon Web Services Inc. (2015, Jul, 8). *Elastic Compute Cloud (EC2) - AWS*. Available: http://aws.amazon.com/ec2/
[10] The Linux Virtual Server. (2015, Jul, 8). *IPVS Software - Advanced Layer-4 Switching*.
Available: http://www.linuxvirtualserver.org/software/ipvs.html
[11] Red Hat Open Source Community. (2015, Jul, 8). *libvirt: The virtualization API*, Available: http://libvirt.org/
[12] C.T. Yang, C.L. Chuang and W.C. Chu, "Implementation of Cloud Infrastructure Monitor Platform with Power Saving Method," in *Proc. IEEE 29th Int. Conf. on Advanced Information Networking and Applications Workshops*, Mar. 2015, pp. 223-228
[13] Canonical Ltd. (2015, Jul, 8). *Ubuntu Server - for scale-out computing*. Available: http://www.ubuntu.com/server
[14] AOE - the open web company. (2015, Jul, 8). *DokuWiki*. Available: https://www.dokuwiki.org/
[15] The Apache Software Foundation. (2015, Jul, 8). *ab - Apache HTTP server benchmarking tool*.
Available: http://httpd.apache.org/docs/2.4/programs/ab.html
[16] Amazon Web Services Inc. (2015, Jul, 8). *Auto Scaling Update New Scaling Policies for More Responsive Scaling*. Available: https://aws.amazon.com/jp/blogs/aws/auto-scaling-update-new-scaling-policies-for-more-responsive-scaling/
[17] J. Jiang, J. Lu, G. Zhang and G. Long, "Optimal Cloud Resource Auto-Scaling for Web Applications," in *Proc. 13th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing*, May. 2014, pp. 58-65
[18] M. Mao, J. Li and Humphrey M., "Cloud auto-scaling with deadline and budget constraints," in *Proc. 11th IEEE/ACM Int. Conf. on Grid Computing*, Oct. 2010, pp. 41-48