Evaluating Auto Scalable Application on Cloud

Takashi Okamoto

Abstract—Cloud computing enables dynamic scaling out of system resources, depending on workloads and data volume. In addition to the conventional scaling-out mechanisms for Web servers, KVS-based scaling-out mechanisms for database servers are drawing attention.

However, there are not many discussions about approaches for automatically scaling out Web servers and database servers, and scaling in database servers that have less workloads and data volume.

In this paper, we present our system and its test results for scaling out and scaling in Web servers and database servers on a cloud-computing platform. Our test results show that workloads on a cloud-computing platform affect applications. We also successfully completed testing of live scaling in of database servers.

Index Terms—Cloud, auto scaleout, web application

I. INTRODUCTION

In March 14, 2011, the Tohoku earthquake devastated Japan. Following the earthquake, many emergency information service systems for checking the safety of people in affected areas were overloaded due to huge volume of access to the systems [1]. As a result, many people were unable to gain access to the systems to check the safety of their family members and friends when they really needed to do so.

Emergency information service systems do not require a lot of servers during normal operations. Scaling out of databases is needed due to increasing access. Once the situation is normalized, access volume decreases and additional servers are no longer required. Databases can also be reduced after deleting messages of safety inquiry.

II. ISSUES

In a Web application, the Web application server generally becomes a bottleneck when the system is overloaded. When a bottleneck occurs, scaling out is kicked in. However, once scaling out of the Web application server is complete, the DB server becomes a bottleneck [2].

When the workload of a system decreases, we need to reduce Web servers and DB servers to eliminate redundant resources. There are several common mechanisms for scaling down of Web servers. One is to replicate session information on Web servers to enable users to continue operations on different servers [3]. Another mechanism is to configure stateless Web servers so as to not retain session information when scaling in. However, there is no established method for scaling in of database servers that are configured as multi servers for load sharing.

In our study, we tested and evaluated scaling out and scaling in of Web servers and database servers on a cloud-computing platform.

III. SYSTEM CONFIGURATION AND APPLICATIONS

The table below describes the system configuration used in our testing environment.

<table>
<thead>
<tr>
<th>Component</th>
<th>Product</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load balancer</td>
<td>HAProxy</td>
<td>1.4.18</td>
</tr>
<tr>
<td>Web Server</td>
<td>Tomcat</td>
<td>7.0.16</td>
</tr>
<tr>
<td>Web Framework</td>
<td>PlayFramework</td>
<td>1.2.3</td>
</tr>
<tr>
<td>Database Server</td>
<td>MongoDB</td>
<td>2.0.1</td>
</tr>
<tr>
<td>Cloud Platform</td>
<td>OpenStack Compute</td>
<td>Cactus</td>
</tr>
</tbody>
</table>

The details of each product are as follows:

A. OpenStack Compute

OpenStack Compute is an open-source middleware component for cloud platforms similar to Amazon EC2, which is used to provide an IaaS environment. OpenStack supports multi-hypervisors, such as Xen/KVM, and can increase and decrease virtual machines through programmable APIs [4]. In our testing environment, we use OpenStack for the cloud platform and KVM for the hypervisor.

B. Load Balancer / Web Server

Because OpenStack does not support a hardware load balancer, we need to provide a software load balancer to build scalable application. We use HAProxy, a very popular and high-performance software load balancer [5].

For a Web server, we use Tomcat, a standard Java application server.

C. Web Framework

There are two major Web application architectures for dynamic scale-out processing. In session-replication architecture, session information is duplicated and retained on each Web server to enable users to continue operations even if their transacting servers are changed due to dynamic scaling down of servers. On the other hand, in stateless-session architecture, session information is not retained on Web servers. Necessary information is kept on client machines.

In a session-replication environment, session information needs to be transferred for duplication to all or a part of the Web servers whenever session information is stored on a single Web server. On the other hand, in a stateless-session environment, session information is not transferred between...
Web servers and state information is retained on users’ Web browsers. Based on our assumption that a stateless architecture is more suitable for scalable applications, we user PlayFramework, which can support a stateless session [6].

D. Database Server
It is not easy for most of the existing RDBMS to scale out databases based on changes in system workloads. Although KVS can scale out, it is difficult to reduce the number of servers once scaling out of servers is complete [7]-[8]. To solve this issue, we use MongoDB on our platform. MongoDB can handle dynamic scaling out. It distributes data to multiple servers when scaling out to enhance throughput with distributed processing. It also scales in to reduce number of database servers that are added by scaling out during high workloads. When scaling in, data on redundant servers are transferred to other databases [9].

Because scaling out and scaling in are processed without interrupting system operations, the current user transactions are not affected.

IV. SYSTEM OVERVIEW
In our testing environment, we installed these middleware components and deployed an application for emergency information service.

![System overview](image1)

Our system provides an emergency information service for corporate users. Using the service, the user can enter employee names and departments on Web browser to inquire the whereabouts of his/her employees when a disaster occurs.

We also deployed monitoring daemons to monitor the workloads of the system. The monitoring daemons periodically access the Web servers and DB servers for monitoring.

If a resource workload exceeds the threshold, scaling out of system resources starts. When scaling out, the system:
- Invokes a virtual machine
- Invokes services
- Configures the load balancer/proxy server
- If resources become redundant, the system runs the following scaling-in processes:
  - Configures the load balancer/proxy server
  - Waits for connection termination
  - Closes server operations

V. BASIC PERFORMANCE EVALUATION
First, we tested the application’s scaling-out performance.

A. Monitoring the Bottleneck on A Single-Server Configuration
We started with a single Web server and DB server configuration. We increased workloads on the servers and monitored the status of each server. Fig. 2 shows the test results.

We prepared an overloading access and run the scenario concurrently to increase workloads on the Web servers. Figure x describes the test results.

Fig. 2 shows that the throughput peaks at 20 concurrent runs and the CPU load of the Web server reaches nearly 100%. Because the CPU load of the DB server remained low, we understand that the performance limit was caused by the Web server bottleneck.

![Web server performance limit](image2)

B. Monitoring the DB Server Bottleneck

![Benchmark with single DB](image3)

![Benchmark with 2 DBs](image4)
Next, we monitored the bottleneck under additional Web servers. We added Web servers one-by-one and checked the peak in the throughput. (Fig. 3)

The throughput reached its peak when the third Web server was added. The CPU load of the DB server exceeded 80%. The figure also shows that the response time of the DB server drastically increased.

These results mean that the DB server became a bottleneck when three Web servers were running concurrently. We added one DB server to the system configuration and monitored the throughput under different Web server configurations. The outcome shows that the throughput increased when six Web servers were running concurrently (Fig. 4).

VI. IMPLEMENT AUTO SCALE AND SCALE IN

Based on the initial test results, scaling out of Web servers and DB servers were tested under the following conditions:

Web server: Scaling out when the Web server workload exceeds 85%.

DB server: Scaling out when the DB server response time exceeds 350ms.

Conditions for scaling in were as follows:

Web server: CPU load is below 10%.

DB server: Response time is below 50ms.

We also prepared for temporary increase and decrease of workloads. For scaling in of Web servers, if the average CPU load measured every 10 s exceeds the threshold twice, we monitored the response time for every 10 s, and if the response time exceeded the threshold four times out of five monitoring intervals, scaling out and scaling in were activated.

The following sections explain the test results of automatic scaling out and scaling in.

A. Scaling out of the Second Web Server

Fig. 5 shows the results of the second Web server scaling out under increased workloads.

Overload was detected at 80 s. The VM was invoked and scaling out started. However, at the beginning of the scaling-out process, the CPU load of web serv1 went down. When the VM was invoked, the VM image was transferred to the host machine for execution. During this image transfer, I/O and network processes were running and the CPU load of web serv1 was decreased.

The CPU load of web serv1 was close to 0 at 320 m. Because the proxy server was reconfigured at this time, requests to web serv1 were stopped temporarily. Then, web serv2 was added to share the system workloads.

While the CPU load reached almost 100% immediately after introducing the second server, the CPU load subsequently became stable. We assume that this spike occurred because Java was running on the Web servers and optimization processing of JavaVM byte codes was executed.

B. Scaling out of the Third Web Server

Fig. 6 describes the results of 2nd web server scaling out. Scaling out of the third Web server started at 900ms. At the beginning of scaling out, the CPU load went down once again. The reason is the same as mentioned above. Before optimization processing of the JVM became stable, the CPU load of web serv2 was nearly 100%. Further, because allocation of web serv2 to the load balancer took longer time, number of allocations to web serv1 was decreased. Thus, the CPU load of web serv1 went down temporarily.
C. Scaling out of the DB Server

Fig. 7 describes the results of DB server scaling out. The load exceeded its threshold at 1400ms and scaling out started at 1410ms. After invoking the VM, the response time went down at 1470ms. The additional database server was activated at 1490ms and added to the database proxy. After adding the database, the CPU load of the first database server reached 100%. This spike occurred due to the data transfer from the first database server to the second database server. After completing data transfer, the workload went down at 1680ms. At the same time, the second database server started servicing and the CPU load increased.

D. Scale in

Fig. 8 shows the results of scaling in. We stopped execution of the overloading scenario at 1900 ms and reduced the load on the servers. The overall status may not be clear on the graph, but one Web server was closed at 1950 ms and the other Web server was closed at 2030 ms. The database server started closing operations at 1980 ms. Around this time, the response of the database became slightly high. The reason of this high response time is that when MongoDB scales in database servers, data on the closing servers are transferred to the remaining servers. After completing data transfer, the response time became low.

VII. CONCLUSION AND FUTURE WORKS

In this study, we tested the system for dynamically scaling out and scaling in Web servers and DB servers on a cloud-computing platform and evaluated the results. We conclude that when scaling out, processes run by cloud platform components, such as copy process for VM image files and system invoking process, affect application performance. In dynamic scale out with MongoDB, the system response goes down until scaling out is complete because data on the current database are transferred to the database added by scaling out. This means that we need to allow some margins when scaling out databases.

On our cloud-computing platform, we used only one system for testing. In the practical operational environment, however, many different systems are running on the single cloud and have complex effect on the system environment. In the future, we would like to test our system on different cloud-computing platforms with multiple systems running. We also would like to study the modeling of system loads affected by different systems.

ACKNOWLEDGMENT

This work was partially supported by Ministry of Internal Affairs and Communications of the Japanese Government.

REFERENCES


Takashi Okamoto received the B.E. and M.E. degrees in Faculty of Engineering from Okayam University, Okayama, Japan in 1994 and 1996 respectively. In 1996, he joined NTT DATA. He is currently working at System Platforms Sector. His research interests include Cloud Platform and Application Architecture for Cloud.