2017 IEEE 10th International Conference on Cloud Computing

Virtual Machine Scaling Method Considering Performance Fluctuation of Public Cloud

Yu Kaneko, Toshio Ito, Masashi Ito, Hiroshi Kawazoe
Corporate Research and Development Center, Toshiba Corporation
Email: {yu1.kaneko, toshio9.ito, masashi8.ito, hiroshi.kawazoe}@toshiba.co.jp

***Abstract*—Cloud computing has been adapted for various application areas because it can reduce the time required for system development and the cost of hardware. One of the factors that degrades performance stability of applications running in the cloud is "unexpected loads", caused by interference between Virtual Machines (VMs) coexisting on the same physical machine. In this paper, we propose a VM scaling method that forecasts performance fluctuation caused by unexpected loads to adjust the number of VMs appropriately. We evaluate the proposed method with simulation to verify that the proposed method can improve performance stability of MQTT brokers.**

I. Introduction

Cloud computing has been adapted for various application areas.Users of public cloud can use Virtual Machines (VMs) as a runtime environment for applications, which leads to reduction of development time and hardware cost because users do not need to procure any hardware. Some research projects are ongoing to migrate Industrial Control Systems (ICS) systems to a cloud environment [1]. ICS require high stability of performance to monitor and control devices accurately. Public cloud providers typically guarantee a certain level of availability of VMs, but normally do not guarantee performance stability of VMs. Achieving high stability of performance in the public cloud is users' responsibility and is a challenge when migrating ICS to a cloud environment.

One of the factors that degrades performance stability of applications or systems running in the cloud is "unexpected loads", for example, interference between VMs coexisting in the same physical machine. We refer to the performance fluctuation caused by unexpected loads as "unexpected performance fluctuation". Another factor that affects performance stability of applications or systems is "expected loads", for example, HTTP requests to web servers. Response times of web servers increase according to increases in the number of concurrent HTTP requests. We refer to this kind of performance change caused by expected loads as "expected performance change".

VM scaling methods are used in the public cloud to minimize the usage fee for the cloud while achieving targeted performance of systems by adjusting the number of VMs appropriately [2][3]. Conventional VM scaling methods forecast performance variation of systems, but those methods do not distinguish expected performance change and unexpected performance fluctuation. System performance can be regarded as time series data, and forecast accuracy of time series data is normally improved by extracting and separating factors of variation [4]. It is because proper forecast algorithms can be applied to separated time series data or noise of time series data can be removed. Therefore, performance stability of systems could be improved by distinguishing expected performance change and unexpected performance fluctuation.

In this paper, we propose a VM scaling method that forecasts expected performance change and unexpected performance fluctuation seperately. The rest of the paper is organized as follows. Section II summarizes related work. Section III explains the proposed method. Section IV describes experimental measurement of performance fluctuation in AWS EC2 [5]. The evaluation of the proposed method is described and discussed in Section V. We conclude our proposal in Section VI.

II. Related Work

The fluctuation of VM performance in the public cloud has been studied thoroughly. For example, VM creation time, VM bootstrap time, and Hadoop job completion time, are affected by unexpected loads [6][7]. Interference between VMs is a factor of unexpected loads. In the public cloud, a VM created by a user is normally deployed on a shared host where VMs created by other users may coexist. It is well known that performance interference occurs between VMs coexisting on the same physical machine [8][9].

VM scaling methods control performance of systems by adjusting the number of VMs based on parameters such as system performance, amount of loads, usage of VM resources [2][10]. Some of the methods consider forecast values of the parameters [3][11]. The methods using forecast values could predict and avoid degradation of system performance if forecast accuracy was sufficient. Avoiding performance degradation is important for achieving high performance stability. However, conventional methods based on VM performance do not distinguish expected performance changes and unexpected performance fluctuation. Performance stability of systems could be improved by distinguishing expected performance changes and unexpected performance fluctuation.

III. Proposed Method

A. System Architecture

We assume an ICS that monitors status of IoT devices with MQTT. MQTT is expected to be a next-generation communication protocol for ICS because of its feasures: low latency and less power consumption [12]. Fig. 1 shows the system architecture assumed in this paper. The system consists of a load balancer and VMs and each VM runs an MQTT broker. There are many IoT devices (publishers) outside the cloud environment, and they send messages (e.g., sensor data) to the load balancer. The load balancer distributes received messages into the MQTT brokers that form a cluster. The

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.114

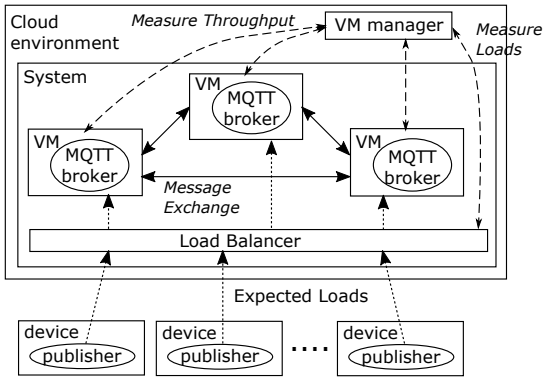

Fig. 1. System architecture assumed in this paper

VM manager measures the number of incoming messages as expected loads and the number of messages processed by each MQTT broker as its performance. Then, it forecasts expected performance changes and unexpected performance fluctuation to determine the appropriate number of VMs (MQTT brokers).

B. Formulation between Expected Loads and Performance

To detect unexpected performance fluctuation, we formulate a relationship between expected loads and performance of a VM in an environment where no unexpected loads exist. The relationship can be figured out based on a simple performance evaluation in a private environment. Multiple pairs of an expected load and performance can be obtained from measurements with various amounts of expected loads. A relationship between an expected load L and performance P can be formulated by applying a fitting method such as regression analysis for the pairs (1).

$$P = f(L) \tag{1}$$

C. Forecast of Unexpected Performance Fluctuation and Expected Loads

Amount of unexpected performance fluctuation of a VMi at time t with an expected load $L_{t,i}$ is calculated by (2). $L_{t,i}$ is an expected load distributed for VMi at t by the load balancer.

$$A_{t,i} = f(L_{t,i}) - P_{t,i} \tag{2}$$

$f(L_{t,i})$ denotes that performance of VMi is affected only by an expected load $L_{t,i}$. $P_{t,i}$ denotes that measured (actual) performance of VMi at t. Therefore, the difference between $f(L_{t,i})$ and $P_{t,i}$ can be considered to be an amount of unexpected performance fluctuation. The VM manager records $A_{t,i}$ as time series data.

Future unexpected performance fluctuation of VMi can be forecast by (3).

$$A_{t+1,i} = forecast(A_{t,i}, A_{t-1,i}, ..., A_{t-K+1,i}) \tag{3}$$

Equation 3 forecasts an amount of unexpected performance fluctuation of VMi at $(t+1)$ from K latest time series data. The proposed method ignores the detail of the $forecast$ method. For example, Exponential Weighted Moving Average (EWMA) or AutoRegressive Integrated Moving Average (ARIMA) can be used as the $forecast$ method.

The VM manager also measures expected loads for the load balancer and records them as time series data to forecast future expected loads. A future expected load at $(t+1)$ is forecast based on J latest measured expected loads by (4).

$$L_{t+1} = forecast(L_t, L_{t-1}, ..., L_{t-J+1}) \tag{4}$$

D. Forecast of VM Performance

The VM manager forecasts performance of VMs based on forecasts of expected loads and unexpected performance fluctuation. Hereafter, we refer to current time as t and future time as $(t+1)$. M denotes the number of VMs at t, and N denotes the number of VMs at $(t+1)$.

First, the VM manager forecasts total amount of expected load L_{t+1} by (4). Then, it calculates future expected loads for each VM $(L_{t+1,1}, ..., L_{t+1,N})$ based on the distribution algorithm of the load balancer. For example, if the load balancer distributes an expected load for VMs equally, $L_{t+1,i} = \frac{L_{t+1}}{N}$. The VM manager also forecasts unexpected performance fluctuation of each VM $(A_{t+1,1}, ..., A_{t+1,N})$ by (3). The VM manager does not have time series data of performance fluctuation of VMi $(i > M)$, and therefore it uses $A_{t+1,M}$ as $A_{t+1,i}$ $(i > M)$. The VM manager finally calculates future performance $P_{t+1,i}$ of VMi, considering unexpected performance fluctuation by (5)

$$P_{t+1,i} = f(L_{t+1,i}) - A_{t+1,i} \tag{5}$$

E. Adjustment of the Number of VMs

The VM manager calculates the appropriate number of VMs according to future performance of VMs forecast by (5). The adjustment algorithm is as follows.

```
1: N ← 1
2: Forecast: L_{t+1}
3: loop
4:    Forecast: L_{t+1,1}, L_{t+1,2}, ..., L_{t+1,N}
5:    Forecast: A_{t+1,1}, A_{t+1,2}, ..., A_{t+1,N}
6:    Forecast: P_{t+1,1}, P_{t+1,2}, ..., P_{t+1,N}
7:    AllGreen ← true
8:    for i = 1 to N do
9:       if P_{t+1,i} does not satisfy TP then
10:         AllGreen ← false
11:      end if
12:   end for
13:   if AllGreen is true then
14:      break
15:   else
16:      N ← N + 1
17:   end if
18: end loop
19: if N > M then
20:    Create (N − M) VMs
21: else if N < M then
22:    Stop (M − N) VMs
23: end if
```

The VM manager forecasts performance of each VMi (line 4–6). The VM manager increases N until all forecast performance $P_{t+1,i}$ satisfy targeted performance TP (line 4–17). After the VM manager determines N, it changes the number of VMs (line 19–23).

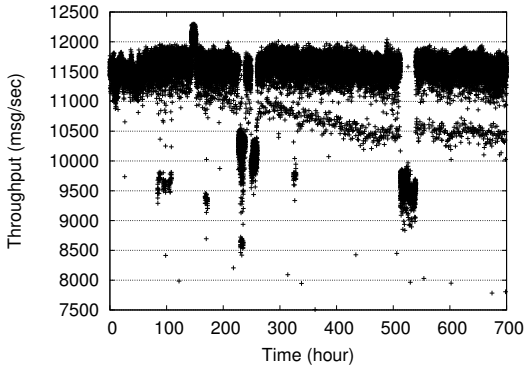

Fig. 2. History of throughput of MQTT broker in AWS EC2

IV. Investigation of Unexpected Performance Fluctuation

We create two VM instances in the Availability Zone (AZ) C of the Tokyo region of AWS EC2. One VM is a Device Under Test (DUT) and the other VM is a tester. They are deployed on different shared hosts, which means that VMs created by other users may interfer with them. The instance type of the VM for the DUT is m3.large and the instance type of the VM for the tester is m4.xlarge. Operating Systems (OS) for those VMs are Ubuntu 14.04.4 LTS. We use VerneMQ (version 0.14.2) [13], an MQTT broker implementation.

We implement a publisher and a subscriber. Hundred publishers and one subscriber are deployed on the tester. Each publisher sends 3000 messages to the MQTT broker running on the DUT. The MQTT broker tranfers all messages to the subscriber. Through this proceduer, performance P, the number of messages processed by the MQTT broker in 1 second, is calculated. This measurement is performed every minute for 30 days.

Fig. 2 shows the measured performance (throughput) of the MQTT broker. The range of throughput is mainly from 11,000 to 12,000 until 250 hour. After 250 hour, another range of throughput, from 10,000 to 11,000, is observed. Standard deviation of measured throughput is 538, which means that throughput of the MQTT broker fluctuates around 5% to the average. This throughput fluctuation may cause message loss. Message loss should be avoided as much as possible in ICS to monitor and control devices accurately.

V. Evaluation

A. Experimental Environment

We evaluate the proposed method by simulation. The duration of the simulation is 30 days. Time unit of the simulation is minute. VM creation time and VM deletion time are also one minute. The initial number of VM is 1. Each publisher sends messages at a regular interval, which means that the number of messages the load balancer receives every minute is constant. The load balancer distributes received messages into the MQTT brokers. Two load balancing algorithms are evaluated in the simulation.

- **Equal Load Balancing (ELB)** distributes received messages into the MQTT brokers equally.
- **Weighted Load Balancing (WLB)** calculates amount of distributed messages according to forecast throughput of each broker.

$$L_{t,i} = L_t \times \frac{P_{t+1,i}}{\sum_{i=1}^{M} P_{t+1,i}} \tag{6}$$

L_t denotes total received messages at time t and M denotes the number of MQTT brokers. $P_{t+1,i}$ denotes forecast throughput of an MQTT broker i.

The time series data obtained by the evaluation in Sec. IV are used to emulate throughput of an MQTT broker. Basically, each value of the time series data is used as throughput of an MQTT broker in a certain minute. Note that the starting point of the time series data for each broker is determined randomly. It means that performance fluctuation may occur at different times between VMs.

The VM manager adjusts the number of VMs (Sec. III). How to obtain a function f in (1) is described in Sec. V-B. Expected loads L_t can be forecast because we assume the number of incoming messages every minute is constant. EWMA and ARIMA are used to forecast unexpected performance fluctuation. Both of the methods have a parameter K, the number of time series data used for forecast. We examined forecast accuracy of EWMA and ARIMA for the data obtained in Sec. IV with different K values. As a result, we decided to use $K = 10$ for EWMA and $K = 480$ for ARIMA.

In the simulation, two metrics are evaluated. First, **success rate** is a metrics of performance stability. In each simulation time unit, the simulator judges whether each MQTT broker can process all incoming messages or cannot. It is considered a "success" if the number of messages distributed to an MQTT broker is less or equal to the throughput of the MQTT broker. Second, **running time** is a time VMs run. Running time of VMs affects usage fee for the public cloud.

B. Relationship between Incoming Messages and Throughput

We conduct an evaluation to determine a relationship between expected loads and performance. Two VMs are deployed as a DUT and a tester in the AZ C of the Tokyo region of AWS EC2. Note that a dedicated host is used for the DUT, instead of a shared host, to mitigate unexpected loads. Instance types of VMs and OS used in this evaluation are the same as for the evaluation in Sec IV. Throughput of the MQTT broker is measured according to various numbers of incoming messages. Incoming messages are considered as expected loads L and throughput is considered as performance P in (1). We formulated the relationship between L and P as follows.

$$P = 3 \times 10^{-11} \times L^2 - 0.0003 \times L + 10242 \tag{7}$$

Equation 7 is used by the VM manager to extract unexpected performance fluctuation.

C. Simulation Results and Discussion

We evaluated the proposed method with different amount of incoming messages from 3,000,000 (3M) messages to 12,000,000 (12M) messages per minute. We refer to the proposed method with EWMA as Proposed-EWMA and the proposed method with ARIMA as Proposed-ARIMA.

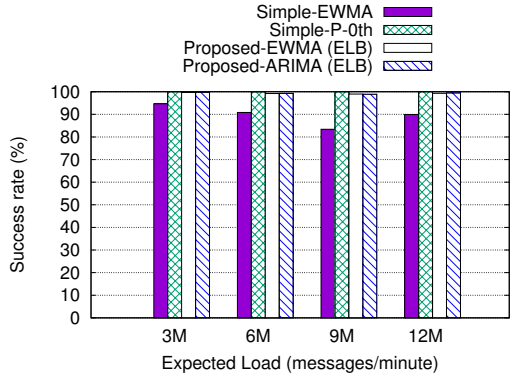

Fig. 3. Success rates of the proposed method and Simple-EWMA

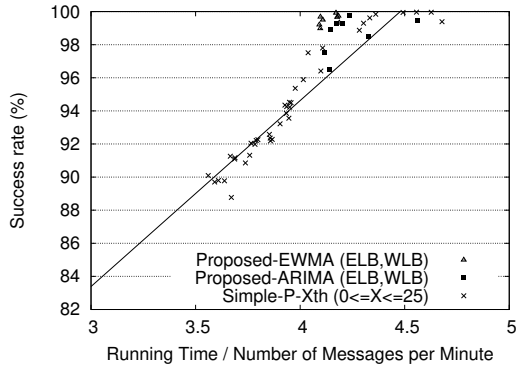

Fig. 4. Comparison of cost-performance between proposed method and simple methods

Two simple methods that do not consider unexpected performance fluctuation are also evaluated for comparison.

- **Simple-EWMA** calculates EWMA from K recent throughput data and uses it as forecast throughput.
- **Simple-Percentile Xth (Simple-P-Xth)** calculates X percentile of throughput from recent throughput data and uses it as forecast throughput. Note that Simple-P-0th means using the minimum throughput of recent throughput data as forecast throughput.

Fig. 3 shows the success rate of MQTT brokers. The proposed method with ELB achieved around 99% success rate regardless of the amount of expected load. The success rate of the proposed method is 5%-15% higher than that of Simple-EWMA, which means that considering unexpected performance fluctuation is effective to improve performance stability. Success rate of the proposed method and Simple-P-0th are almost same, therefore, those success rate should be compared with consideration of running time.

Fig. 4 shows a relationship between the success rate and the running time obtained from the simulation. To normalize running time, each running time is divided by the corresponding expected load. The cross line in Fig 4 is a linear approximate formula calculated from simulation results of Simple-P-Xth with various X values from 0 to 25. The success rate of Simple-P-Xth seem to be proportional to its running time. The points of Proposed-EWMA are located upper left to the cross line, which means that Proposed-EWMA can achieve high success rate compared to Simple-P-Xth with the same running time. Therefore, Proposed-EWMA is superior to Simple-P-Xth from cost-performance perspective. On the other hand, Proposed-ARIMA is less effective than Proposed-EWMA because a portion of the points of Proposed-ARIMA are close to the cross line.

VI. Conclusion

We proposed a VM scaling method that forecasts the unexpected performance fluctuation to adjust the number of VMs. We evaluated the proposed method and confirmed that the proposed method improved the success rate of message processing up to 15% compared with simple methods that do not consider the unexpected performance fluctuation. In future work, we intend to evaluate the proposed method with inconstant expected loads, different VM instance types, and more advanced methods for time series forecast. we also intend to implement a prototype of the proposed method and verify its effect on the public cloud.

References

[1] O. Givehchi, H. Trsek, and J. Jasperneite, "Cloud computing for industrial automation systems - A comprehensive overview," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, Sept 2013, pp. 1–4.

[2] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "DejaVu: Accelerating Resource Allocation in Virtualized Environments," *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 423–436, Mar. 2012.

[3] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. USENIX, 2013, pp. 69–82.

[4] Y.-H. L. Jin-Cherng Lin and C.-H. Liu, "Building time series forecasting model by independent component analysis mechanism," in *Proceedings of The World Congress on Engineering 2007*, July 2007, pp. 1010–1015.

[5] Amazon Web Services. Elastic Compute Cloud (EC2) Cloud Server & Hosting. [Online]. Available: https://aws.amazon.com/ec2/

[6] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*, June 2012, pp. 423–430.

[7] K. Aida, O. Abdul-Rahman, E. Sakane, and K. Motoyama, "Evaluation on the performance fluctuation of hadoop jobs in the cloud," in *2013 IEEE 16th International Conference on Computational Science and Engineering*, Dec 2013, pp. 159–166.

[8] G. Somani and S. Chaudhary, "Application Performance Isolation in Virtualization," in *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, Sept 2009, pp. 41–48.

[9] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, and Y. Cao, "Who Is Your Neighbor: Net I/O Performance Interference in Virtualized Clouds," *Services Computing, IEEE Transactions on*, vol. 6, no. 3, pp. 314–329, July 2013.

[10] H. Kang, J. in Koh, Y. Kim, and J. Hahm, "A sla driven vm auto-scaling method in hybrid cloud environment," in *2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Sept 2013, pp. 1–6.

[11] H. Fernandez, G. Pierre, and T. Kielmann, "Autoscaling web applications in heterogeneous cloud infrastructures," in *2014 IEEE International Conference on Cloud Engineering*, March 2014, pp. 195–204.

[12] Y. t. Lee, W. h. Hsiao, C. m. Huang, and S. c. T. Chou, "An integrated cloud-based smart home management system with community hierarchy," *IEEE Transactions on Consumer Electronics*, vol. 62, no. 1, pp. 1–9, February 2016.

[13] Erlio GmbH. VerneMQ. [Online]. Available: https://vernemq.com