ERP: An elastic resource provisioning approach for cloud applications

Danqing Feng1,2, Zhibo Wu1, DeCheng Zuo1, Zhan Zhang1*

1 Computer science and Technology, Harbin Institute of Technology, Harbin, China, 2 Computer science and Technology, Air Force Communication NCO Academy, DaLian, China

Abstract

Elasticity is the key technique to provisioning resources dynamically in order to flexibly meet the users’ demand. Namely, the elasticity is aimed at meeting the demand at any time. However, the aforementioned approaches usually provision virtual machines (VMs) in a coarse-grained manner just by the CPU utilization. Actually, two or more elements are needed for the performance metric, including the CPU and the memory. It is challenging to determine a suitable threshold to efficiently scale the resources up or down. In this paper we present an elastic scaling framework that is implemented by the cloud layer model. First we propose the elastic resource provisioning (ERP) approach on the performance threshold. The proposed threshold is based on the Grey relational analysis (GRA) policy, including the CPU and the memory. Secondly, according to the fixed threshold, we scale up the resources from different granularities, such as in the physical machine level (PM-level) or virtual machine level (VM-level). In contrast, we scale down the resources and shut down the spare machines. Finally, we evaluate the effectiveness of the proposed approach in real workloads. The extensive experiments show that the ERP algorithm performs the elastic strategy efficiently by reducing the overhead and response time.

Introduction

Cloud computing is popular in industry due to its ability to deliver on-demand resources according to a pay-as-you-go model [1]. Usually, three basic service models are included in cloud computing: Infrastructure as a Service (IaaS) [2], Platform as a Service (PaaS) [3] and Software as a Service (SaaS) [4]. Namely, SaaS provides access to complete applications as a service. PaaS provides a platform for developing other applications on top of it, such as the Google App Engine (GAE) and Azure. IaaS provides an environment to deploy the managed virtual machines. Technically, when the users submit the requests, the providers would provide the resources depending on the users’ demand [5–6]. As a key technique in cloud computing, the elasticity [7] has the ability to acquire and release the resources according to the users’ demand.

Generally, the providers implement an automatic provisioning approach via the virtualization technique [8]. Virtualization makes it possible to rapidly scale the resources up or down. The aforementioned approaches [9] present a reactive method, which is triggered by a certain
threshold, such as CPU utilization or memory utilization. Actually, two or more thresholds [10] should be used as a performance metric. In addition, it is important to provision the correct amount of the resources efficiently using a suitable threshold. In fact, the fluctuating workload would lead to an overprovisioning state or an underprovisioning state. To avoid these problems, researchers usually use a predictive technique, such as the proactive method. These feasible predictive approaches, such as machine learning [11–12], Moving Average [13], and Auto-Regression [14], would track the dynamic resource requirement and effectively minimize the energy consumption. This predictive policy would quantify the requirement in advance in order to flexibly scale the resources up or down. However, it is a challenging issue to improve the accuracy of the predictive technique. Additionally, an estimation error would lead to an overprovisioning or underprovisioning state. When in the sudden workload, this predictive method is especially inaccurate. Thus, combining this with an automatic method and a proactive method would be more agile for provisioning the resources. For example, the Elastic VM architecture [15] provisions the resources dynamically to reduce the SLA violation. However, the elasticity is necessary to meet the users’ demand from different perspectives. Some researchers would take the performance metrics into consideration, such as the SLA [16] and the profit of the providers [17]. However, more metrics are used in the elasticity to evaluate the performance [18]. For example, from the purpose of the providers, they might consider more related elements. That is, they would seek to minimize the renting cost, the energy consumption and the Service Level Agreement (SLA) violation. In summary, the elasticity would be implemented for one or two purposes, such as saving energy [19–20] or reducing the cost [21–22]. However, it is difficult to make an elasticity solution by considering multiple objectives. To solve the mentioned issues, we propose the ERP approach to provision the resources by the performance threshold, including the CPU and the memory. According to the threshold, we would flexibly scale the resources up or down by considering multiple perspectives. From the perspective of the provider, the goal is aimed at minimizing the amount of the resources to reduce the energy consumption. From the perspective of the users, the goal is aimed at rapidly scaling the resources up or down. In brief, the ERP approach is aimed at maximizing the utilization and minimizing the SLA violation. Then, the main contributions would be summarized in the following.

First, this approach solves the suitable threshold to determine the users’ demand. We present the performance threshold by using the GRA method, which considers such multiobjectives as the CPU utilization and memory utilization. Meanwhile, it is instructed on the cloud layer model using the MAPE loop. Usually the MAPE loop includes four phases, such as Monitoring (M), Analysis (A), Planning (P) and Execution (E).

Second, this approach solves the issue of scaling the resources flexibly. According to the proposed threshold, we could efficiently scale the resources up or down. That is, we propose a fine-grained algorithm, which means to scale up the resources from the PM-level or the VM-level in order to flexibly meet the users’ demand.

Third, this approach solves the issue of reducing the overheads. When it is overprovisioned, we would shut down the extra machines to reduce the energy consumption via a simple predictive technique, such as the weighted moving average (WMA).

The remainder of the paper is described as listed below. Section 2 analyses the related literature on the elastic techniques in cloud computing. Section 3 presents the ERP framework based on the layer model. Section 4 provides the performance threshold via the cloud layer model. Section 5 presents the effective ERP algorithm, which would scale the resources up or down from different granularities. Section 6 proves the results by comparing them with the aforementioned approaches. Finally, section 7 draws conclusions and describes future development.
Related work

Usually the elastic solution is implemented by scaling the resources in or out. By analyzing some related works, we would divide the elastic resource provisioning approaches into two major aspects, including automatic scaling methods [23] and elastic mechanisms on the predictive technique [24].

Automatic scaling methods

In the automatic policy, the resources would be provisioned and released automatically according to the demand. Generally, the action is triggered by the fixed thresholds, such as the utilization. The common techniques are provided by Amazon and Scalr. However, they provision the resources only based on the utilization, when in fact more elements have taken effect. Additionally, its advantage is a kind of coarse-grained provisioning strategy to scale the virtual machines. When considering the fine-grained provisioning strategy, some researchers focus on the reactive methods by resizing the resources dynamically and minimizing the response time and executing cost in cloud computing. However, they focus more on the fine-grained scaling strategy, and less on multiple perspectives. Kingfisher [25] proposed an elastic mechanism to reduce the transition of time and cost. This approach exploits the available resources on the virtual machines to scale in or out, and uses an integer linear program formulation to optimize the cost. Leitner et al. [26] proposed the SLA-aware scheduling algorithm, which would reduce the request execution time. It presents a cost-efficient method to scale up from the perspective of the providers. In contrast, our approach considers more factors to formulate the threshold by the cloud layer model, such as CPU utilization, memory utilization, etc. Additionally, we aim to scale the resources by minimizing the renting cost and response time. This would shut down the spare machines from the perspective of saving the consumption. By analyzing the mentioned works, we determined that most recent elastic strategies focus on the horizontal elasticity. Therefore, it is important to scale the resources from different granularities, including horizontal elasticity [27] and vertical elasticity [28]. By considering the fine-grained elasticity, we present the ERP algorithm to scale up the resources in the PM-level or VM-level by the performance threshold. Moreover, when it is in overprovisioning, it would scale down the resources in the VM-level.

Elastic mechanisms based on the prediction

In fact, elasticity is essential to meet a fluctuating workload, and it is necessary to determine the suitable amount of the resources in order to scale the resources. Actually, the proactive approaches are used to determine the next demand, such as the Autoregressive moving average model (ARMA) [29] and Holt winter [30]. These predictive techniques have the advantage of giving an accurate prediction value in the stable workload. However, these predictive techniques focus more on the accuracy, but ignore the complexity. Moreover, when a sudden workload appears it might be in estimation error. To reduce the complexity of the prediction algorithm, some techniques are used to determine the repetitive patterns and predict the next values. PRESS [31] is a predictive elasticity system that analyzes and extracts the workload patterns and provisions the resources automatically. The advantage of this policy is that improves the prediction accuracy, and it reduces the resource waste efficiently. However, it only makes emphasis on the overhead. CloudScale [32] is a system that automates the fine-grained resources in cloud computing infrastructures, determining the adaptive resources by the prediction. In addition, it integrates the dynamic CPU voltage scaling to saving the consumption by migration. This technique puts more emphasis on the proactive method based on the prediction, which would minimize the energy consumption and avoid the Service Level Object
An elastic resource provisioning approach

(SLO) violation. In fact, more elements should be taken into consideration. Hence, in our approach, we consider more elements, such as reducing the renting cost, energy consumption and SLA violation. Additionally, we increase or decrease the resources automatically from different granularities to meet the demand, including fine-grained scaling and coarse-grained scaling. Namely, when it is underprovisioning, our approach scales up the resources from different granularities by the performance threshold, such as in the PM-level or VM-level. In contrast, we scale down the VMs by the WMA predictive technique efficiently.

Proposed approach

In this section, we present our proposed approach for the detailed description. Our approach is designed on the cloud layer model. That is, this policy is implemented to determine the performance threshold to flexibly scale the resources up or down. Additionally, the formulation of the performance threshold is presented in detail in the next section. Then the ERP framework is explained in detail in the following.

Cloud layer model

In this section, our approach describes a cloud layer model to scale the resources rapidly. The cloud layer model focuses more on the quantitative analysis, whereas the Delphi method [33] depends more on the subjective assessment. The ERP approach is implemented on the cloud layer model. The layer model is composed of three parts: SaaS, PaaS and IaaS. The SaaS determines a series of requests offered by the users. In the PaaS the broker is responsible for provisioning the infrastructure resources according to the users’ demand which is presented by the MAPE loop. In IaaS, the datacenter is composed of some PMs and VMs. The provider would provision the resources according to the requests. As depicted in Fig 1, the key components of the MAPE are described in detail as follows.

Monitor (M). The monitoring component collects some metrics, such as the CPU utilization, memory utilization and some available resources. It monitors the information every five seconds. The key information is collected, aggregated and calculated by the performance model, which is described in detail in the next section.

Analyze (A). The analyzing phase is responsible for analyzing the collected information. The obtained data is aggregated and calculated by the performance model, and we achieve the performance value to decide whether the scaling action is triggered. Moreover, we use the WMA predictive technique to determine the correct number of the servers and shut down the spare machines.

Plan (P). This component is the core of the cloud layer model. According to the users’ demand, it implements the scaling strategy by minimizing the renting cost and reducing the energy consumption. Additionally, it would increase or decrease the resources by the performance threshold.

Execute (E). In the executing phase, the Nginx load balancing server balances the web requests by provisioning the servers in the infrastructure. Since the VMs are hosted in the PMs, the provider would provision the resources according to the demand by using the proposed plan.

Proposed framework

In our approach we propose a novel framework to flexibly increase or decrease the resources aiming at minimizing the renting cost, energy consumption and response time, as illustrated in Fig 2. The ERP algorithm is mainly composed of two phases. In the first phase, the performance model constructs a baseline threshold, which is aggregated and calculated by the
gathered data. From this the resources would be rapidly scaled up or down. In the second phase, the ERP algorithm is used to scale the resources by the performance threshold for the purpose of minimizing the renting cost and saving power consumption.

Then, we explain these two phases in detail. In the first step, the monitoring component monitors the CPU utilization, memory utilization, CPU clock speed and some available resources. We aggregate the gathered data to make a performance evaluation by the proposed cloud layer model. In the second step, we make a further description on the ERP approach. In the analyzing component, we scale the resources by the performance threshold. Actually, the planning phase may lead into two states, including an underprovisioning state or overprovisioning state. When it is in an underprovisioning state, we execute the action on increasing the resources at the PM-level. If it continues, we go on scaling up the resources at the VM-level. The PM-level scaling depends on the available resources in the same host. The VM-level scaling is based on the VMs hosted on the PMs. Additionally, the VM could come from the same PM or another PM. Otherwise, when it is in overprovisioning we scale down the resources by the prediction. Then the extra spared machines would be shut down by saving the energy consumption. Moreover, our approach implements the elastic scaling from different granularities with the consideration of minimizing the cost and the SLA violation.

**Performance threshold**

In this section, we present a performance threshold on multiple elements. From this we would rapidly scale the resources up or down in cloud computing.
This policy presents a multicriteria threshold that takes five related criterion into account, as shown in Table 1. The criteria on the TOPSIS and GRA policy would include the cost type and benefit type. After the matrix is normalized, the TOPSIS method evaluates them by the positive ideal solution and negative ideal solution. Then, the GRA method makes the decision from less information and explores the system behavior by analyzing the related degree.

Usually the information on the PMs is gathered every 5 seconds to form the decision matrix, as shown in Eq 1. The gathered data is described as depicted in Table 1. Then we
Table 1. The parameters of the ERP approach.

| Parameter       | Description                                      | Type   |
|-----------------|--------------------------------------------------|--------|
| CPU Cycle       | The CPU cycle refers to the CPU Clock Speed.     | Cost   |
| RAM Capability  | RAM Capability is the rest memory capability.    | Cost   |
| CPU%            | The utilization of CPU is the CPU usage of the PM.| Benefit|
| RAM%            | The utilization of RAM is the RAM usage of the PM.| Benefit|
| Availability    | Supposing that the resource availability is higher than 90%. | Cost |

https://doi.org/10.1371/journal.pone.0216067.t001

construct and implement the performance threshold in detail as follows.

$$ R = \begin{bmatrix} r_{cycle_1} & r_{restMem_1} & r_{cpu_1} & r_{mem_1} & r_{avail_1} \\ r_{cycle_2} & r_{restMem_2} & r_{cpu_2} & r_{mem_2} & r_{avail_2} \\ r_{cycle_3} & r_{restMem_3} & r_{cpu_3} & r_{mem_3} & r_{avail_3} \end{bmatrix} \quad (1) $$

**Normalization of the decision matrix.** In the first step we normalize the decision matrix. Namely, the decision matrix is normalized by achieving the average value of every column as listed in Eq 2.

$$ R = \begin{bmatrix} \frac{r_{cycle_1}}{SUM_{cycle}} & \frac{r_{restMem_1}}{SUM_{restMem}} & \frac{r_{cpu_1}}{SUM_{cpu}} & \frac{r_{mem_1}}{SUM_{mem}} & \frac{r_{avail_1}}{SUM_{avail}} \\ \frac{r_{cycle_2}}{SUM_{cycle}} & \frac{r_{restMem_2}}{SUM_{restMem}} & \frac{r_{cpu_2}}{SUM_{cpu}} & \frac{r_{mem_2}}{SUM_{mem}} & \frac{r_{avail_2}}{SUM_{avail}} \\ \frac{r_{cycle_3}}{SUM_{cycle}} & \frac{r_{restMem_3}}{SUM_{restMem}} & \frac{r_{cpu_3}}{SUM_{cpu}} & \frac{r_{mem_3}}{SUM_{mem}} & \frac{r_{avail_3}}{SUM_{avail}} \end{bmatrix} \quad (2) $$

**Improved TOPSIS.** This is the abbreviation of the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS). The traditional TOPSIS method depends more on subjective weights, while the improved TOPSIS solutions depend more on key factors. In the second step, the ideal solution would be determined by Eq 3 and Eq 4. That is, for the cost type the ideal solutions are the smaller ones, and the negative solutions are the larger ones. It is the opposite situation for the benefit type. Then, we achieve the positive ideal solution and the negative ideal solution, respectively.

$$ P_j^+ = \{ (\max(\tilde{r}_j)|i \in I), (\min(\tilde{r}_j)|i \in J) \} \quad (3) $$

$$ P_j^- = \{ (\min(\tilde{r}_j)|i \in I), (\max(\tilde{r}_j)|i \in J) \} \quad (4) $$

**Grey relational analysis.** Grey theory is an effective way to solve multiobjective decision problems in the engineering areas [34–35]. In the following step, we determine the difference between the comparative series $r_{jk}$ and the standard series $P_j^+$ or $P_j^-$. Additionally, the distinguish coefficient $\rho$ is usually 0.5, and is generally between [0, 1]. Then, the Grey relational coefficients $\zeta^+$ and $\zeta^-$ are constructed by Eq 5 and Eq 6, respectively.

$$ \zeta^+(k) = \frac{\min_j \min_k |r_{jk} - P_j^+| + \rho \max_j \max_k |r_{jk} - P_j^+|}{|r_{jk} - P_j^+| + \rho \max_j \max_k |r_{jk} - P_j^+|} \quad (5) $$

$$ \zeta^-(k) = \frac{\min_j \min_k |r_{jk} - P_j^-| + \rho \max_j \max_k |r_{jk} - P_j^-|}{|r_{jk} - P_j^-| + \rho \max_j \max_k |r_{jk} - P_j^-|} \quad (6) $$
Actually, the weight coefficients are determined by the analytic hierarchy process (AHP) method [36]. Then we determine the degree of relation \( r \) on the weight coefficients \( \omega \) by multiplying them by Grey relational coefficient \( \xi(k) \). Additionally, the degree of relations \( r^+ \) and \( r^- \) are formulated by Eqs 7 and 8, respectively.

\[
r^+ = \sum_{k=1}^{m} \omega_k \xi^+(k) \tag{7}
\]

\[
r^- = \sum_{k=1}^{m} \omega_k \xi^-(k) \tag{8}
\]

Then we formulate the relative closeness coefficient \( u^+ \) to the ideal solution by Eq 9, which is implemented on the ideal relational coefficient \( r^+ \) divided by the sum of the positive relational coefficient \( r^+ \) and negative relational coefficient \( r^- \).

\[
u^+ = \frac{r^+}{r^+ + r^-} \tag{9}
\]

**Performance model**

The performance threshold is constructed by the entropy method [37], which is an effective method to calculate the deviation degree. The smaller the entropy value is, the better the performance is. Similarly, the larger entropy value is, the worse the performance is. Therefore, we determine the performance threshold by the entropy method, which is listed as in Eq 10.

\[
\Delta P = -\log_2 \left( \frac{P_2}{P_1} \right) = -\log_2 \left( \frac{u^+}{u^-} \right) = -\log_2 \left( \frac{u_{\max}^+}{u_{\max}^-} \right) \tag{10}
\]

Where \( \Delta P \) is the performance threshold by the entropy method, \( P_1 \) is the probability before the demand varies, and \( P_2 \) is the probability after the demand varies. Additionally, the probability is constructed on the ideal relational coefficient \( u^+ \) divided by the max relational coefficient \( u_{\max} \).

In the scheduling, a current performance value below 0.1 denotes a better performance environment. When it is above 3, it denotes a poor performance environment [38–39]. In fact, a normal value is between 0.1 and 3, which is described in Table 2. In our experiments, when the performance value is lower than 0.1, we would scale down the servers. Then, we set 0.1 as the lower threshold \( P_d \). When the value is greater than 0.2, we would scale up the servers for the purpose of reducing the response time by reserving slightly more resources. Then, we set 0.2 as the upper threshold \( P_u \).

**The ERP algorithm**

In this section we describe the ERP algorithm to scale the resources from different granularities according to the users’ demand.

| Service Level | Rule and Action            |
|---------------|---------------------------|
| \(<0.1\)      | Better-Level              |
| \([0.1,0.9]\) | Scaling down the servers  |
| \([0.9,3]\)   | Normal-Level              |
| \(\geq3\)     | In the normal statement   |
| \(\geq3\)     | Worse-Level               |
| \(\geq3\)     | In the worse statement    |

Table 2. The description of the service-level.Scope.
**ERP algorithm**

To provision the resources flexibly, we first discuss some related definitions on the elasticity, such as the resilience and scalability. Next, we define them and clarify the difference between them. **Scalability** means to the ability of the system to deal with an increasing amount of the servers in a capable manner. However, it focuses more on the increasing ability, and less on the response time. **Resilience** means to provision the resources rapidly in a flexible way. Elastic scheduling refers to two core conditions, including the time and speed [40]. In this paper, we define an elastic scheme $S$, which is represented as $S = (\text{clock}, U_{cpu\%}, U_{mem\%}, P_u, P_d)$, where $\text{clock}$ is the CPU cycle, $U_{cpu\%}$ and $U_{mem\%}$ are the CPU utilization and the memory utilization, respectively, which are gathered by monitoring the system, and $P_u$ and $P_d$ are the upper and lower thresholds, respectively. In brief, the main algorithm (refer to Algorithm 1) provisions the resources rapidly via the MAPE loop. In the monitoring and analyzing components, some key elements are collected to determine the performance threshold. Then in the planning and executing components, the elastic scheme would scale the resources by the performance threshold. To make the ERP algorithm understood for the further step, Table 3 lists the main parameters of the ERP algorithm as below.

Next, the ERP algorithm is described in detail. It implements an elastic resource provisioning approach in the datacenter. This algorithm takes the performance threshold as the baseline to scale the resources up or down. At first, the monitoring component would collect and gather the information as listed in Table 1 (lines 1–2) every few minutes. In fact, the ERP algorithm would increase or decrease the resources to meet the users’ demand. When the performance value $P$ is larger than the upper threshold $P_u$, the algorithm would be triggered to scale up the servers (SUS) (lines 4–5). In contrast, once the current performance value is below the threshold $P_d$, the scaling down the servers (SDS) algorithm is triggered (lines 6–7).

**Algorithm 1. ERP (Elastic resource provisioning)**

1: Initialization: Server, $P$
2: while (the allocation is deploying)
3: monitor the performance value $P$
4: if $P > P_u$
  5: Scaling up the servers (SUS)
6: else if $P < P_d$
  7: Scaling down the servers (SDS)
7: End

The proposed ERP algorithm has included two aspects. First, the scaling up the servers (SUS) algorithm proposes a scaling method that is based on different granularities. That is, we scale up the VMs in the same available PMs or from some different PMs. Second, the scaling down the servers (SDS) algorithm presents the approach to shut down the extra machines.

Table 3. The summary of the notations in the ERP algorithm.

| Symbol | Description               | Relation               |
|--------|---------------------------|------------------------|
| PM     | Physical machine          | Related resources      |
| VM     | Virtual machine           | Related resource       |
| $P$    | The current performance value | Calculated by the monitoring system |
| $P_u$  | The upper performance threshold | Determined by the experiments |
| $P_d$  | The lower performance threshold | Determined by the experiments |
| $u_{cpu\%}$ | The CPU utilization of the VM | Monitoring system |
| $u_{mem\%}$ | The memory utilization of the VM | Monitoring system |
| $c(p_i)$ | The renting cost of the PM | Cloud providers |
| $c(v)$  | The renting cost of the VM | Cloud providers |

https://doi.org/10.1371/journal.pone.0216067.t003
The SUS algorithm

The SUS algorithm is intended to scale up the resources in a flexible way, including from the PM-level or VM-level. The SUS algorithm is described by Algorithm 2. The monitoring component collects some metrics related to the resources (lines 1–4). If the performance evaluation reaches the upper threshold $P_u$, it scales up more available resources on the PM (lines 5–7). When the updated performance value continues past on the upper threshold $P_u$, we would provision slightly more resources (lines 8–10). Additionally, the VMs might come from different PMs.

**Algorithm 2: SUS (Scaling up the servers)**

1: Begin  
2: Initialization: Server, $P$  
3: while (the allocation is deploying)  
4: monitor the performance value $P$  
5: if $P > P_u$  
6: Scaling up the PMs  
7: update the performance value $P$  
8: while ($P > P_u$)  
9: Scaling up the VMs  
10: update the performance value $P$  
11: End

The PLI algorithm

The purpose of the PM-Level increasing (PLI) algorithm is to increase the VMs on the available PMs (refer to Algorithm 3). Then we explore the PLI algorithm in detail. The monitoring component aggregates the information and calculates the performance value (lines 1–3). Once the triggered action appears we scale up the residual resources on the available PMs. Then we would choose the PMs aimed at minimizing the renting cost (lines 4–7). Additionally, the cost function is described by Eqs 11 and 12. Finally, it updates the performance value (line 8).

**Algorithm 3: PLI (PM-level increasing)**

1: Begin  
2: Initialization: Server, $P$  
3: Calculating the performance value  
4: while ($P > P_u$)  
5: if PM is available  
6: select the min cost PM to increase  
7: update the performance value  
8: End

In this phase, Eq 11 is aimed at minimizing the renting cost, where $u_{cpu\%}$ presents the CPU utilization of the VM. The binary variable $v_j$ indicates whether or not the VM is selected, and the binary variable $p_i$ indicates whether or not the PM is selected. The parameter $m$ is responsible for the amount of VMs hosted on the current host, and $c(p_i)$ is the expending cost of the current host.

$$\text{Minimize } p_i \cdot \sum_{j=1}^{m} v_j \cdot (1 - u_{cpu\%}) \cdot c(p_i)/\sum_{j=1}^{m} v_j$$

subject to:

$$\sum_{j=1}^{m} v_j \leq m, v_j \in \{0, 1\}, P_i \in \{0, 1\}$$

$$11$$

$$12$$
The VLI algorithm
In this section, we propose the VM-level increasing (VLI) algorithm (refer to Algorithm 4) to continue increasing the resources to meet the fluctuating demand. It consists of three parts: monitoring the component, increasing the resources and updating the state. Then we describe the algorithm 4 in detail. First, the monitoring component gathers the information to calculate the performance value (lines 1–3). Second, we choose the suitable VMs to increase (lines 4–5), which would determine minimizing the expending cost by Eq 13. That is, we calculate the remaining utilization by the CPU and the memory. We implement the cost in Eq 13 by multiplying the remaining utilization by the single VM renting cost. The purpose of the function is to achieve the VM with a minimum cost, where the binary variable \( v_j \) indicates whether or not the VM is selected in Eq 14. Then we make a global search to find a suitable VM to increase. Finally, we update the state and calculate the performance value (lines 6–7).

Algorithm 4: VLI (VM-level increasing)

1: Begin
2: Initialization: Server, \( P \)
3: Calculating the performance value
4: while (\( P > P_u \))
5: select the min cost VM to increase
6: update the performance value
7: End

\[
C_{vm} = \arg\min_{j} (1 - u_{cpu}) \cdot (1 - u_{mem}) \cdot c(v_j)
\] (13)

\[
j \in \{1, 2, ..., m\}, v_j \in \{0, 1\}
\] (14)

The SDS algorithm

The aforementioned algorithms (refer to Algorithms 2–4) implement increasing the resources from a different granularity according to the users’ demand. In this section, the Scaling-down servers (SDS) algorithm is described for the detailed steps. In the first step we monitor the component and gather some information to achieve the performance threshold (lines 1–3). Once the SDS algorithm is triggered we would scale down the resources. Then we select the extra machines to shut down for the purpose of minimizing the cost (lines 4–5). Hence, we shut down the machines that occupy the maximum expending cost. Eq 15 is as listed below. Finally, we update the state and determine the current performance threshold (lines 6–7).

\[
C'_{vm} = \arg\max_{j} (1 - u_{cpu}) \cdot (1 - u_{mem}) \cdot c(v_j)
\] (15)

Algorithm 5: SDS (Scaling-down servers)

1. Begin
2. Initialization: Server, \( P \)
3. Calculating the performance value
4. while (\( P < P_d \))
5. select the max cost VM to decrease
6. update the performance value
7. End

Experiments

In this section, we implement the elastic resource allocation strategy based on the performance criterion. Meanwhile, the proposed approach proves that it is appropriate for meeting the demand in different kinds of workloads. In addition, this approach considers both reducing the renting cost and improving the utilization.
Environment setup

We use the CloudStack platform and simulated real-world workloads to evaluate the ERP approach. We deploy a cluster composed of ten PMs. One installs the CloudStack platform. The other nine PMs use Xenserver as the management nodes (2.20GHz Intel(R) Xeon(R) 8 CPU, 8 G memory, running CentOS 6.9). We create 27 VMs (1 VCPU, 1 G memory, running CentOS 6.9) in the cluster. Then, the database is run off MySQL. When the workload is fluctuating, the Nginx has the function off balancing the servers. All the configuration information is listed in Table 4.

To evaluate the proposed approach, we design two kinds of workloads: synthetic workloads and real-world workloads. We use the Jmeter to generate the requests based on the TPC benchmark. First, the synthetic workload would vary from the users’ demand. The fluctuating process of the workload is described as below. The load generator would implement 600, 900, 600, 1200, 600, and 1800 users, which is shown in Fig 3, which lasts for over 30 minutes. Second, the simulated real-world workload is extracted from the EPA and NASA traces [41]. The two kinds of real-world workload traces are generated as shown in Fig 4. Additionally, the monitoring service is implemented by the Jmeter plugins, such as monitoring the response time, CPU utilization or memory utilization. The experiment would last for over 40 minutes.

Evaluation metric

In the experiments, we consider some performance indicators as the metrics, such as the renting cost, energy consumption, resource utilization and SLA violation.
The cost. This metric might be measured by the reserved and on-demand VMs. For example, the basic unit of the CPU is set at 1 GB in Aliyu. It is charged \(0.059\) ¥/hour in the reserved plan and \(0.28\) ¥/hour for the on-demand plan. The renting cost is defined in Eq 16, where \(C_r\) and \(C_o\) are responsible for the renting cost in the reserved or on-demand plan, respectively. Then in the scheduling the average overhead is described in Eq 17, where it is calculated by the sum of the cost divided by the time interval \(T\).

\[
C = C_r x_r + C_o x_o
\]  

\[
C_{avg} = \frac{\sum_{t=1}^{t=T} C}{T}
\]

Energy consumption. This metric might be measured by the average energy consumption, which is defined as the energy consumption ratio as listed in Eq 19, where \(N\) is the total number of the intervals. Additionally, the energy consumption is expressed in Eq 18.

\[
Power = k \times P_{\text{max}} + (1 - k) \times P_{\text{max}} \times u
\]

Where the idle power consumption coefficient [42] \(k\) is equal to 0.7, and the parameter \(P_{\text{max}}\) represents the peak power. Additionally, \(u\) is based on the CPU utilization.

\[
Power_{avg} = \frac{\sum_{t=1}^{n} Power}{N}
\]

The utilization. The utilization is one of key indicators to evaluate the performance in the scheduling. The average utilization is defined as the ratio between the total CPU utilization
and the total number of the intervals, as shown in Eq 20.

\[ U_{\text{avg}} = \frac{\sum_{i=1}^{n} U_{t}}{N} \]  

**SLA violation.** The SLA violation can be calculated by the percentage of the difference between the actual requests and allocated requests divided by the total requests, as described in Eq 21. Generally, the SLA violation might be measured by the CPU utilization [43], just in Eq 22. Then the average SLA violation is defined as the ratio between the total SLA violation and the total number of the intervals, expressed by Eq 23. In fact, the SLAV is expressed by the average SLA multiplied by the average response time, as shown in Eq 24.

\[ SLA = \frac{\text{Req}_{\text{total}} - \text{Req}_{\text{allocate}}}{\text{Req}_{\text{allocate}}} \]  

\[ sla = \frac{1}{1 + e^{U_{\text{cpu}} - 0.8}} \]  

\[ sla_{\text{avg}} = \frac{\sum_{i=1}^{n} sla_{i}}{N} \]  

\[ SLAV = sla_{\text{avg}} \times \text{time} \]

**Algorithms in comparison**

To validate the ERP algorithm, we compare it with other algorithms, such as lightweight resource scaling (LS) algorithm [44], the proactive method [45], and the reactive method [46].

**Reactive method.** The traditional algorithm is scaled by the CPU utilization, obeying the simple principle by a rule-condition-action. In the experiments, the threshold is usually fixed at 0.8 or 0.2. Namely, when the utilization is higher than 0.8, the VMs would be increased. In contrast, when the utilization is lower than 0.2, the resources would be decreased.

**Proactive method.** The proactive method means that it would scale the servers up or down by the prediction technique, such as ARMA. That is, it could scale the resources up or down by the ARMA.

**LS.** The LS algorithm focuses more on the response time. When it is higher than the upper threshold the number of the VMs increases. In contrast, the number of the VMs would be scaled down. Additionally, the algorithm would shut down the spare machines by a simple predictive technique.

**Experiment results**

Actually, our proposed algorithm is constructed on the performance value, which is calculated by the GRA and TOPSIS policy. In more experiments we determine that the performance threshold range is between 0.1 and 0.2. Namely, when it is greater than 0.2, we would scale up the servers, and when it is lower than 0.1 we would scale down the servers. Moreover, the performance evaluation considers multiple angles, such as maximizing the utilization, and minimizing the power consumption and the SLA violation. The results prove the effectiveness of the ERP approach.
The number of the servers. In the synthetic load, the reactive algorithm puts a greater emphasis on the scalability of the servers and reacts quickly at first. The proactive algorithm would obtain the suitable number of the servers in the regular load test, and the LS algorithm spends less resources. Our proposed algorithm could occasionally occupy slightly more resources than the LS algorithm to meet the multidimension requirement in the simulated experiment at the beginning, as shown in Fig 5. In the real-world load, including EPA and NASA, our algorithm would occupy slightly more resources at first. Next it would outperform other algorithms in the normal level, as illustrated in Figs 6 and 7. We determine that the LS algorithm is unsuitable for various loads. That is, because the LS algorithm depends more on the response time. When a sudden load appears, it would increase the overhead. However, our approach has the advantage of efficiently avoiding a sudden load efficiently by reserving slightly more resources.

Fig 5. The number of CPUs in the synthetic load.
https://doi.org/10.1371/journal.pone.0216067.g005

Fig 6. The number of CPUs in the EPA load.
https://doi.org/10.1371/journal.pone.0216067.g006
The renting cost. We measure the renting cost using Eq 18. As shown in Fig 8, in the synthetic load the LS algorithm puts a greater emphasis on the time to scale the resources. Namely, in the stable workload it gains the smallest average renting cost. We find that the ERP algorithm spends a slightly higher cost than the LS due to reserving few resources at first. The proactive algorithm would obtain a better result in the regular load test by the prediction. Our proposed algorithm obtains a lower cost than the reactive algorithm. As shown in Fig 9, in the real-world load we find that our proposed algorithm obtains a lower cost than the other algorithms, and the LS algorithm obtains a higher cost depending on the response time. When it appears in the sudden load, the LS algorithm would scale up the resources more quickly, which makes the occupied resources greater than in other algorithms.

Fig 7. The number of CPUs in the NASA load.
https://doi.org/10.1371/journal.pone.0216067.g007

Fig 8. Average renting cost in the synthetic load.
https://doi.org/10.1371/journal.pone.0216067.g008
Resource utilization. We measure the average resource utilization based on Eq 20. Figs 10 and 11 show the CPU utilization during the experiments under different workloads, including the synthetic load and real-world loads. In these experiments, we determine that the proposed approach utilizes the resources more fully, which is depicted in Figs 10 and 11. In the experiments the ERP method consumes slightly more resources at first and simultaneously guarantees a lower SLA violation rate. Additionally, it releases the servers by the WMA prediction by guaranteeing the performance in the varying workloads. We see that no resource utilization is higher than 100%, which proves that our approach efficiently reduces the underprovisioning state.

Response time. The response time is another performance metric that needs to be studied. As depicted in Figs 12 and 13, in the synthetic workload, when considering the maximum response time, we determine that our proposed algorithm obtains a quicker response than the
other algorithms by reserving few resources at first. For the average response time, these algorithms are in the acceptable level at the stable workload. As depicted in Figs 14 and 15, in the real-world loads we find that our algorithm presents a lower maximum or average response time than others by reserving slightly more resources at first, while the LS algorithm obtains slightly higher time due to a longer monitoring time. Additionally, it is unfit for the sudden load. In the NASA load the variable workload leads to inaccurate prediction values, so the proactive algorithm obtains a longer average response time.

**SLA violation.** We measured the SLA violation based on Eq 24. As shown in Figs 16 and 17, in the workloads our algorithm presents a lower SLA violation ratio than the other algorithms. In addition, the error rate is another metric that evaluates the performance. As listed in Table 5, we see that our algorithm produces a slightly lower error ratio and efficiently avoids the sudden load.
Average energy consumption. We measure the average energy consumption based on Eq 19. As shown in Fig 18, in the synthetic load our algorithm achieves a lower power than the LS and proactive algorithms. Since it is in the stable workload, the reactive algorithm obtains a better result than the other algorithms only by the utilization. As shown in Fig 19, in real-world loads the proposed algorithm presents a lower power than the LS and reactive algorithm. The proactive method consumes less energy consumption than others, but it cannot meet the demand due to the inaccurate prediction. This is because that it achieves a higher error rate in Table 5.

Conclusion

Traditional elasticity is often used as a reactive method, which is implemented by the rule-condition-action. However, it would be a better strategy to combine this with the prediction. In
In this paper, we present an elastic strategy that increases or decreases the resources by the performance threshold in a flexible manner. To further elaborate, the ERP approach makes the following contributions. First, we present the performance threshold depending on the CPU and the memory. By this, we could flexibly scale the resources up or down. This solves the issue of deciding a suitable threshold on multiple elements. Second, we propose an SUS algorithm that implements the fine-grained scaling in the PM-Level or VM-Level to increase the resources flexibly. This solves the issue of an elastic scaling strategy from different granularities to reduce the SLA violation and response time. Third, combining this with the WMA prediction we propose the SDS algorithm to scale down the servers. Then we would shut down the spare machines to save energy consumption. This solves the issue of effectively saving the overheads. Finally, we evaluate the proposed ERP approach in the simulated and real-world workloads.
The results show that the ERP method improves the utilization, minimizes the renting cost, saves the energy consumption and gives a quicker response time.

In fact, we implement the scaling approach on the premise of regarding the servers as the available resources. However, no cloud provider offers unlimited resources, except for Google

Table 5. The error analysis in the experiments.

| Type      | ERP (error%) | LS (error%) | Proactive (error%) | Reactive (error%) |
|-----------|--------------|-------------|-------------------|------------------|
| Synthetic | 0            | 0           | 0                 | 0                |
| EPA       | 0            | 0.01%       | 0.02%             | 0                |
| NASA      | 0            | 1.22%       | 22.50%            | 0                |

Fig 17. SLAV in the real-world workloads.

https://doi.org/10.1371/journal.pone.0216067.g017

Fig 18. Energy consumption in the synthetic workload.

https://doi.org/10.1371/journal.pone.0216067.g018

The results show that the ERP method improves the utilization, minimizes the renting cost, saves the energy consumption and gives a quicker response time.

In fact, we implement the scaling approach on the premise of regarding the servers as the available resources. However, no cloud provider offers unlimited resources, except for Google
and Amazon. Thus, a further study should be made on some aspects. First, it is necessary to find an effective way to minimize the renting cost by reserving some available resources in advance. However, more servers would be wasted by reserving too many resources. Therefore, it is necessary to balance the reserved plan and the on-demand plan. Second, from the perspective of minimizing the energy consumption, a reasonable dynamical provisioning approach might efficiently consolidate the available resources by the migration technique. Then in the future it will be necessary to explore the dynamical provisioning approach in the complex workloads. Perhaps some typical types of the workflow would be an interesting extension in the future.

**Supporting information**

**S1 Table. Synthetic load or real-world workloads by Jmeter.** The synthetic workload is generated by Jmeter, as is the simulated real workload, such as the EPA and the NASA.

**S2 Table. The number of CPUs in the synthetic load.** The number of CPU in the synthetic load and real-world load is monitored by Jmeter plugin.

**S3 Table. The logs in the synthetic load.** The logs record the power, the SLA and the utilization.

**S4 Table. The logs in the EPA load.** The logs record the power, the SLA and the utilization.

**S5 Table. The logs in the NASA load.** The logs record the power and the SLA. Additionally, the utilization is evaluated in the workloads.

**S1 Performance Evaluation.** The SLA violation and energy consumption are evaluated in the synthetic load and real-world workloads.
Author Contributions

Conceptualization: Danqing Feng, Zhibo Wu, DeCheng Zuo, Zhan Zhang.

Formal analysis: Danqing Feng, Zhibo Wu.

Methodology: Danqing Feng, Zhibo Wu, DeCheng Zuo, Zhan Zhang.

Validation: Danqing Feng.

Writing – original draft: Danqing Feng.

Writing – review & editing: Danqing Feng.

References

1. Rimal, B. P., Choi, E., & Lumb, I. A taxonomy and survey of cloud computing systems. In INC, IMS and IDC; 2009. NCM’09. Fifth International Joint Conference on Ieee. 2009 Aug; 44–51. https://doi.org/10.1109/NCM.2009.218

2. Manvi S. S., & Shyam G. K. Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. Journal of Network and Computer Applications 2014; 41: 424–440. https://doi.org/10.1016/j.jnca.2013.10.004

3. Zhang Q., Cheng L., & Boutaba R. Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications 2010; 1(1): 7–18. https://doi.org/10.1007/s13174-010-0007-6

4. Armbrust M., Fox A., Griffith R., Joseph A. D., Katz R., Konwinski A., & Zaharia M. A view of cloud computing. Communications of the ACM 2010; 53(4): 50–58. https://doi.org/10.1145/1721654.1721672

5. Jennings B., & Stadler R. Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management 2015; 23(3): 567–619. https://doi.org/10.1007/s10922-014-9307-7

6. Hameed A., Khoshkbarforoush A., Ranjan R., Jayaraman P. P., Kolodziej J., Balaji P., & Khan S. U. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. Computing 2016; 88(7): 751–774. https://doi.org/10.1007/s00607-014-0407-8

7. Galante, G., & Bona, L. C. E. D. A survey on cloud computing elasticity. In Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing. IEEE Computer Society. 2012 Nov; 263–270. https://doi.org/10.1109/UCC.2012.30

8. Dillon, T., Wu, C., & Chang, E. Cloud computing: issues and challenges. In Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on Ieee. 2010 April; 27–33. https://doi.org/10.1109/AINA.2010.187

9. Amazon elastic compute cloud (EC2). http://aws.amazon.com/ec2/ (29.10.2011).

10. Kaur P. D., & Ghana I. A resource elasticity framework for QoS-aware execution of cloud applications. Future Generation Computer Systems 2014; 37: 14–25. https://doi.org/10.1016/j.future.2014.02.018

11. Di Sanzo, P., Rughetti, D., Ciciani, B., & Quaglia, F. Auto-tuning of cloud-based in-memory transactional data grids via machine learning. In Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on IEEE. 2012 Dec; 9–16. https://doi.org/10.1109/NCCA.2012.20

12. Moore, L. R., Bean, K., & Ellahi, T. Transforming reactive auto-scaling into proactive auto-scaling. In Proceedings of the 3rd International Workshop on Cloud Data and Platforms. ACM. 2013 April; 7–12. https://doi.org/10.1145/2460756.2460758

13. Mi, H., Wang, H., Yin, G., Zhou, Y., Shi, D., & Yuan, L. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In Services Computing (SCC), 2010 IEEE International Conference on IEEE. 2010 July; 514–521. https://doi.org/10.1109/SCC.2010.69

14. Kan, C. DoCloud: An elastic cloud platform for Web applications based on Docker. In Advanced Communication Technology (ICACT), 2016 18th International Conference on IEEE. 2016 Jan; 478–483. https://doi.org/10.1109/ICACT.2016.7423440

15. Dawoud, W., Takouna, I., & Meinel, C. Elastic vm for cloud resources provisioning optimization. In International Conference on Advances in Computing and Communications. Springer, Berlin, Heidelberg. 2011 July; 431–445. https://doi.org/10.1007/978-3-642-22709-7_43

16. Serrano, D., Bouchenak, S., Kouki, Y., Ledoux, T., Lejeune, J., Sopena, J., & Sens, P. Towards qos-oriented sla guarantees for online cloud services. In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on IEEE. 2013 May; 50–57. https://doi.org/10.1109/CCGrid.2013.66
17. Chen, J., Wang, C., Zhou, B. B., Sun, L., Lee, Y. C., & Zomaya, A. Y. Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In Proceedings of the 20th international symposium on High performance distributed computing. 2011 June; 229–238. ACM. https://doi.org/10.1145/1996130.1996161

18. Zheng, Z., & Zhang, Y. Cloudrank: A qos-driven component ranking framework for cloud computing. In 2010 29th IEEE Symposium on Reliable Distributed Systems. IEEE. 2010 October; 184–193. https://doi.org/10.1109/SRDS.2010.29

19. Beloglazov A., & Buyya R. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In MGC@ Middleware. 2010 Nov; 4. https://doi.org/10.1145/1890799.1890803

20. Hong H. J., Chen D. Y., Huang C. Y., Chen K. T., & Hsu C. H. Placing virtual machines to optimize cloud gaming experience. IEEE Transactions on Cloud Computing. 2015; 3(1): 42–53. https://doi.org/10.1109/TCC.2014.2338295

21. Dutta, S., Gera, S., Verma, A., & Viswanathan, B. Smartscale: Automatic application scaling in enterprise clouds. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on IEEE. 2012 June; 221–228. https://doi.org/10.1109/CLOUD.2012.12

22. Buyya, R., Ranjan, R., & Calheiros, R. N. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In International Conference on Algorithms and Architectures for Parallel Processing, Berlin, Heidelberg, 2010 May; 13–31. Springer.

23. GoGrid. http://www.gogrid.com/(29.10.2011).

24. Caron, E., Desprez, F., & Muresan, A. Forecasting for Cloud computing on-demand resources based on pattern matching, Doctoral dissertation, INRIA. 2010. https://doi.org/10.1109/CloudCom.2010.65

25. Sharma U., Shenoy P., Sahu S., & Shaikh A. Kingfisher: Cost-aware elasticity in the cloud. In INFOCOM, 2011 Proceedings IEEE. IEEE. 2011 April; pp. 206–210. https://doi.org/10.1109/INFCOM.2011.5935016

26. Leitner, P., Hummer, W., Satzger, B., & Stachtiari, E. Forecasting for Cloud computing on-demand resources based on pattern matching, Doctoral dissertation, INRIA. 2010. https://doi.org/10.1109/CloudCom.2010.65

27. Naskos, A., Stachtiari, E., Gounaris, A., Katsaros, P., Tsoumakos, D., Konstantinou, I., & Sioutas, S. Dependable horizontal scaling based on probabilistic model checking. In 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE. 2015 May; 31–40. https://doi.org/10.1109/CCGrid.2015.91

28. Sotiriadis S., Bessis N., Amza C., & Buyya R. Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling. IEEE Transactions on Services Computing. 2016.

29. Fernandez, H., Pierre, G., & Kielmann, T. Autoscaling web applications in heterogeneous cloud infrastructures. In: Cloud Engineering (IC2E), 2014 IEEE International Conference on IEEE. 2014 March; 195–204. https://doi.org/10.1109/IC2E.2014.25

30. da Silva Dias, A., Nakamura, L. H., Estrella, J. C., Santana, R. H., & Santana, M. J. Providing IaaS resources automatically through prediction and monitoring approaches. In Computers and Communications (ISCC), 2014 IEEE Symposium on IEEE. 2014 June; 1–7. https://doi.org/10.1109/ISCC.2014.6912590

31. Gong Z., Gu X., & Wilkes J. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. CNSM. 2010; 10: 9–16.

32. Shen, Z., Subbiah, S., Gu, X., & Wilkes, J. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM. 2011 Oct, 5. https://doi.org/10.1145/2038916.2038921

33. Okoli C., & Pawlowski S. D. The Delphi method as a research tool: an example, design considerations and applications. Information & Management 2004; 42(1): 15–29. https://doi.org/10.1016/j.im.2003.11.002

34. Li Y., Li Y., Li G., Zhao D., & Chen C. Two-stage multi-objective OPF for AC/DC grids with VSC-HVDC: Incorporating decisions analysis into optimization process. Energy. 2018; 147, 286–296. https://doi.org/10.1016/j.energy.2018.01.036

35. Li Y., Wang J., Zhao D., Li G., & Chen C. A two-stage approach for combined heat and power economic emission dispatch: Combining multi-objective optimization with integrated decision making. Energy. 2018; 162, 237–254. https://doi.org/10.1016/j.energy.2018.07.200

36. Zahedi F. The analytic hierarchy process—a survey of the method and its applications. Interfaces. 1986; 16(4): 96–108. https://doi.org/10.1287/inte.16.4.96
37. Dincer I., & Cengel Y. A. Energy, entropy and exergy concepts and their roles in thermal engineering. Entropy. 2001; 3(3): 116–149. https://doi.org/10.3390/e3030116

38. Zhao G. S., Wang H. Q., & Wang J. Study on Situation Evaluation for Network Survivability Based on Grey Relation Analysis. MINIMICRO SYSTEMS-SHENYANG. 2006; 27(10): 1861.

39. Zhao, G., Wang, H., & Wang, J. A novel quantitative analysis method for network survivability. In Computer and Computational Sciences, 2006. IMSCCS’06. First International Multi-Symposiums on IEEE. 2006 June; 2: 30–33. https://doi.org/10.1109/IMSCCS.2006.160

40. Feng D., Wu Z., Zhang Z., & Fu J. On the Conceptualization of Elastic Service Evaluation in Cloud Computing. Journal of Information Technology Research (JITR). 2019; 12(1), 36–48. https://doi.org/10.4018/JITR.2019010103

41. Traces in the Internet Traffic Archive[EB/OL]. http://ita.ee.lbl.gov/html/traces.html.

42. Hu, L., Jin, H., Liao, X., Xiong, X., & Liu, H. Magnet: A novel scheduling policy for power reduction in cluster with virtual machines. In Cluster Computing, 2008 IEEE International Conference on IEEE. 2008 Sep; 13–22. https://doi.org/10.1109/CLUSTER.2008.4663751

43. Ma F., Liu F., & Liu Z. Multi-objective optimization for initial virtual machine placement in cloud data center. Journal of Information & Computational Science. 2012; 9(16): 5029–5038.

44. Han R., Guo L., Ghanem M. M., & Guo Y. Lightweight resource scaling for cloud applications. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE Computer Society. 2012 May; pp. 644–651. https://doi.org/10.1109/CCGrid.2012.52

45. Roy, N., Dubey, A., & Gokhale, A. Efficient autoscaling in the cloud using predictive models for workload forecasting. In Cloud Computing (CLOUD), 2011 IEEE International Conference on IEEE. 2011 July; pp. 500–507. https://doi.org/10.1109/CLOUD.2011.42

46. RightScale. http://www.rightscale.com/ (29.10. 2011).