International Journal of Innovative Computing, Information and Control
Volume **15**, Number **1**, February **2019**

ICIC International ©2019 ISSN 1349-4198
pp. 337–349

ENERGY-EFFICIENT VIRTUAL MACHINE SCHEDULING STRATEGY WITH SEMI-SLEEP MODE ON THE CLOUD PLATFORM

Bing Qin[1,2], Shunfu Jin[1,2,*] and Dongmei Zhao[3]

[1]School of Information Science and Engineering
[2]The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province
[3]Science and Technology on Communication Networks Laboratory
Liren College
Yanshan University
No. 438, Hebei Street, Haigang District, Qinhuangdao 066004, P. R. China
qinbing94@163.com; *Corresponding author: jsf@ysu.edu.cn; dongmeiz@foxmail.com

Received April 2018; revised August 2018

Abstract. *With the rapid increases in service types of cloud computing and the number of cloud resources, energy consumption is also growing. In order to achieve the goal of environmental protection and provide timely service for cloud users, we propose a virtual machine (VM) scheduling strategy with semi-sleep mode on the cloud platform. Based on the working principle of the proposed strategy, we establish a continuous-time queueing model with multiple asynchronous working vacation. By using the method of quasi birth and death (QBD) process and matrix geometric solution, we evaluate the VM scheduling strategy in terms of the average latency of requests as well as the energy-saving rate of system, and investigate the impact of the semi-sleep mode of the strategy on the system performance. Statistical experiments with analysis and simulation show that, by setting reasonable system parameters, the energy consumption could be reduced effectively with the constraint of response performance of users. In addition, we establish a cost function to optimize the semi-sleep parameter.*

Keywords: Cloud platform, Virtual machine scheduling, Semi-sleep, Asynchronous working vacation, Matrix geometric solution

1. Introduction.

With the development of cloud computing, users could obtain the required resources and process large application data through the cloud. However, more energy in cloud computing will be consumed than that in traditional computing mode. How to effectively save energy in cloud computing while satisfying the quality of service (QoS) requirements has become one of the hot topics.

One of the major reasons of energy inefficiency on the cloud platform is the energy waste when physical machines (PMs) run at low utilization. Researchers try to switch the idle or low-load PMs to sleep state to save energy. In [1], based on the predicted frequency of requests, the idle PMs were moved to sleep state. The operating frequency of PM being in a running state was adjusted dynamically by dynamic voltage frequency scaling. In [2], an energy-aware operation mode for load balancing and application scaling on the cloud was introduced. An energy-optimal operation regime was defined and the number of operating servers was maximized in this regime. Idle and lightly-loaded servers were switched to sleep state to save energy. In [3], the energy-efficient cloud orchestrator (e-eco) was proposed. E-eco acted as the cloud load balancer to calculate the amount of sleeping hosts for the purpose of achieving maximum energy conservation. In the literature mentioned above, the cloud resources could be adjusted as required by employing the

DOI: 10.24507/ijicic.15.01.337

sleeping mechanism of PMs. The sleeping mechanism is energy-efficient to some extent; however, quite often it can affect the response performance of users on the cloud platform. Semi-sleep mode means that server provides service at a lower service rate rather than completely stopping service during the semi-sleep period. In this way, semi-sleep mode makes the server run alternately at two different service rates, which could effectively solve the work delay caused by the sleeping mechanism and maintain the energy-efficient effect of the sleeping mechanism. So it is of great significance to introduce semi-sleep mode to alleviate the potential impact of the sleeping mechanism on the response performance.

Naturally speaking, the queueing model with working vacation is appropriate for describing the semi-sleep mode. The original research of working vacation was carried out by Servi and Finn in [4]. There have been many academic achievements on the analysis of queueing model with working vacation since then. In [5], the authors developed a single server queue with working vacation and server breakdowns. In this model, the server worked at a different rate. In [6], a discrete-time multi-server queue with single synchronization working vacation was studied. The model could be regarded as a packet polling system, in which users synchronously extracted part of service nodes while meeting energy efficiency. In [7], a queueing model with multiple synchronization working vacation was discussed to provide the theoretical basis and analytical method for the optimal design and control of communication network. From the literature mentioned above, we find that all the researches on the multi-server working vacation queueing model are based on synchronous working vacation. For a synchronous working vacation, all the servers start and end a working vacation simultaneously, i.e., all the servers are always in the same state. On the contrary, for an asynchronous working vacation, each server starts a working vacation independently, i.e., different servers could have different states at the same time. Obviously, analysis of the queueing model with asynchronous working vacation is more complicated than that with synchronous working vacation. So there is few research on the queueing model with asynchronous working vacation.

To the best of our knowledge, the present paper is the first attempt to propose an energy-efficient virtual machine (VM) scheduling strategy with semi-sleep mode, and to establish a continuous-time queueing model with multiple asynchronous working vacation. The established queueing model quantifies the effects of system parameters, such as the semi-sleep parameter and the lower service rate, on the system performance of the proposed strategy. These effects are measured by the average latency of requests and the energy-saving rate of system.

The rest of this paper is organized as follows. In Section 2, we propose an energy-efficient VM scheduling strategy and establish the system model accordingly. In Section 3, we establish an analytical framework based on Markov chain to obtain the steady-state distribution. In Section 4, we derive performance measures and provide statistical experiments. In Section 5, we establish a cost function of system to optimize the semi-sleep parameter. In Section 6, we conclude the whole paper.

2. Energy-Efficient VM Scheduling Strategy and System Model.

In this section, we propose an energy-efficient VM scheduling strategy with semi-sleep mode on the cloud platform. Accordingly, we develop a continuous-time queueing model with multiple asynchronous working vacation.

2.1. Energy-efficient VM scheduling strategy.

On conventional cloud platform, large amounts of energy are wasted because the VMs are always running normally even though there are no requests of cloud users to be processed. In order to improve the energy-efficiency, sleeping mechanism is introduced to the cloud platform. However, a general sleeping mechanism potentially affects the response performance of cloud users. How to improve the sleeping mechanism to get a balance between the energy-efficiency on the cloud platform and the response performance of cloud users is a critical problem to be solved.

In consideration of the trade-off between the energy-saving of cloud platform and the response performance of cloud users, we propose a novel energy-efficient VM scheduling strategy with semi-sleep mode. Based on the semi-sleep mode, the VMs will be switched among semi-sleep period and awake period.

(1) *Awake period*: During the awake period, all the requests of cloud users are processed at a normal service rate. When a request is processed completely on a VM, if there is at least one request waiting in the buffer, the first request waiting in the buffer will occupy the just evacuated VM and receive service at a normal service rate. Otherwise, the evacuated VM will enter its own a semi-sleep period independently, while the other VMs will remain their own state.

(2) *Semi-sleep period*: Each VM has its own semi-sleep timer and each semi-sleep timer works independently. Once a VM enters a semi-sleep period, its semi-sleep timer will be activated with a random time length in order to control the time length of the semi-sleep period. During the semi-sleep period, if there are any requests to be processed, the VMs will operate at a lower service rate, rather than completely stopping service. When a request arrives at the cloud platform, if there is at least one spare VM operating at a lower service rate, the newly arriving request will receive service on one of those spare VMs. When a request is being processed on a VM at a lower service rate, if the semi-sleep timer of the VM expires, the VM will suspend current service for the request and convert to the awake period immediately from the semi-sleep period, and then the VM will resume the service for the suspended request at a normal service rate. When a VM finishes the service for a request at a lower service rate before the semi-sleep timer expires, if there are no requests waiting in the buffer, the just evacuated VM will be spare; otherwise, the just evacuated VM will provide service for the first request waiting in the buffer at a lower service rate. When the semi-sleep timer of a VM expires, if the VM is spare, the VM will start a new semi-sleep period and the semi-sleep timer of the VM will be reactivated, while the other VMs will remain their own state.

From the descriptions above, we find that different VMs on the cloud platform may be in different periods at the same time.

The workflow of the VM scheduling strategy with semi-sleep mode is illustrated in Figure 1.

In Figure 1, we observe that the workflow of the VM scheduling strategy with semi-sleep mode is different from that with traditional sleep mode. In traditional sleep mode, the VM could either be spare during the sleep period and completely not provide service at all, or provides the service during the awake period with high energy consumption. In semi-sleep mode, in order to guarantee the energy-efficiency on the cloud platform, the VM during the semi-sleep period keeps running at a lower service rate with low energy consumption, i.e., spare VM could directly provide service to the newly arriving request at a lower service rate. When a semi-sleep period of the VM ends, the VM which provides service at a lower service rate will convert to the awake period, and then continue to provide service at a normal service rate. With semi-sleep mode, the impact on the response performance of cloud users caused by sleep mode will be reduced effectively.

2.2. System model.

Based on the working principle of the proposed energy-efficient VM scheduling strategy, we establish a continuous-time queueing model with multiple asynchronous working vacation.

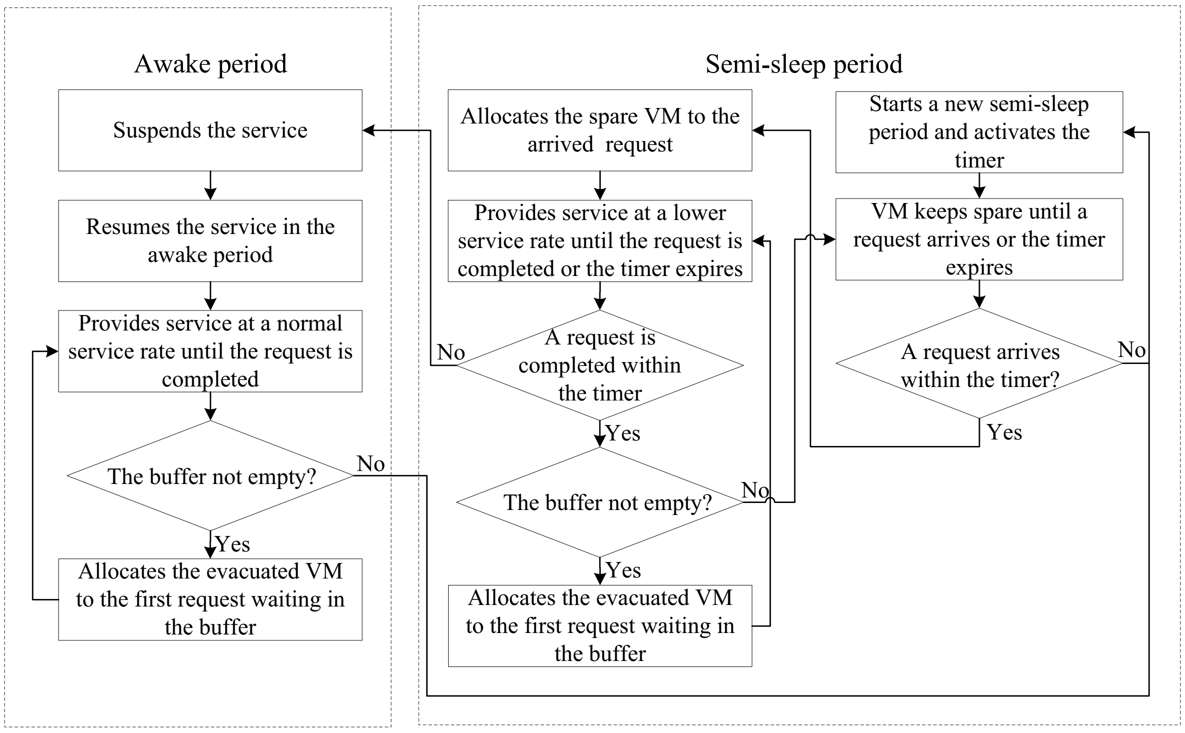
Awake period
Suspends the service
Resumes the service in the awake period
Provides service at a normal service rate until the request is completed
The buffer not empty?
No
Yes
Allocates the evacuated VM to the first request waiting in the buffer
Semi-sleep period
Allocates the spare VM to the arrived request
Provides service at a lower service rate until the request is completed or the timer expires
A request is completed within the timer
No
Yes
The buffer not empty?
No
Yes
Allocates the evacuated VM to the first request waiting in the buffer
Starts a new semi-sleep period and activates the timer
VM keeps spare until a request arrives or the timer expires
A request arrives within the timer?
No
Yes

Figure 1. Workflow of the VM scheduling strategy with semi-sleep mode

We focus on one of the PMs on the cloud platform, and assume that there are several identical VMs on that PM, such as c ($c > 1$) VMs. The semi-sleep mode is described as working vacation in the queueing model, and the requests and the VMs are regarded as customers and servers, respectively. Suppose that requests arrive at the system according to a Poisson process with the positive finite parameter λ. If a request enters into the system and finds that all the VMs are occupied, the request has to wait in the buffer until a VM is available. It is the turn of this request to be processed. The arriving requests form a single queueing line based on the order of their arrivals and are processed according to the first-come, first-served (FCFS) discipline. Suppose that each VM can process only one request at a time. If the service for a request is provided during the awake period, the service time of the request is assumed to be exponentially distributed with the positive parameter μ_1. If a request is processed completely during a working vacation period, the service time of the request is assumed to be exponentially distributed with the positive parameter μ_0. In our proposed energy-efficient VM scheduling strategy, the service rate μ_1 during the awake period is called as normal rate, the service rate μ_0 during the working vacation period is called as lower rate, i.e., $\mu_1 > \mu_0$. If one of the VMs processes a request completely during the awake period and the buffer is empty at the service completion instant, the VM will begin a working vacation period individually. When a working vacation period of one spare VM terminates, the VM will begin another working vacation period independently. We assume that the time length of a working vacation period is exponentially distributed with the positive parameter θ. Here, the positive parameter θ is called as semi-sleep parameter.

We consider the system model with an infinite buffer capacity. Let random variable $L(t) = k$, $k \in \{0, 1, 2, \ldots\}$ be the number of requests in the system at time t, where $k = 0$ means that there are no requests in the system. Let random variable $J(t) = j$, $j \in \{0, 1, 2, \ldots, c\}$ be the number of busy VMs in the awake period at time t, where $j = 0$ means that all the VMs are in the working vacation period. $L(t)$ is called as system level and $J(t)$ is called as system stage. The state of the queueing system at time t is described by $\{L(t), J(t)\}$. $\{L(t), J(t), t \geq 0\}$ is a continuous-time two-dimensional Markov chain with the state space $\mathbf{\Omega} = \{(k, j) | k \in \{0, 1, 2, \ldots\}, j \in \{0, 1, 2, \ldots, c\}\}$.

Assume that the two-dimensional Markov chain $\{L(t), J(t), t \geq 0\}$ is positive recurrent. We define the steady-state distribution π_{kj} of the Markov chain as follows:

$$\pi_{kj} = \lim_{t \to \infty} P\{L(t) = k, J(t) = j\}, \quad (k, j) \in \mathbf{\Omega} \tag{1}$$

The steady-state distribution π_{kj} is written as a partitioned vector. The partitioned vector can be given as follows:

$$\boldsymbol{\pi}_k = (\pi_{k0}, \pi_{k1}, \pi_{k2}, \ldots, \pi_{kc}), \quad k \geq 0 \tag{2}$$

$$\mathbf{\Pi} = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots) \tag{3}$$

3. Model Analysis.

In this section, we firstly give the state transition of the two-dimensional Markov chain $\{L(t), J(t), t \geq 0\}$, and then derive the steady-state distribution of the system model.

3.1. State transition.

The state transition indicates the movement from one state to another. Via one step state transition, the system level (the number of requests in the system) and the system stage (the number of VMs providing service at a normal rate) will change as follows.

(1) The system level changes from k to $(k + 1)$ means that a new request arrives at the system.

For the case of $0 \leq k < c$, there is at least one spare VM in the system, the newly arriving request will occupy one of the spare VMs and get service with a lower service rate. The state (k, j) will transfer to state $(k + 1, j)$ with transition rate λ.

For the case of $k \geq c$, all the VMs in the system are occupied, the newly arriving request has to wait in the buffer for future service. The state (k, j) will transfer to state $(k + 1, j)$ with transition rate λ.

(2) The system level changes from k to $(k-1)$ means that a request departs the system.

For the case of $0 < k \leq c$ and $j = 0$, all the occupied VMs are operating at a lower service rate. If the departed request is processed completely within the current semi-sleep period, the evacuated VM will be spare. The state (k, j) will transfer to state $(k - 1, j)$ with transition rate $k\mu_0$.

For the case of $1 < k \leq c$ and $0 < j < k$, the departed request can be processed completely either with a lower service rate or with a normal service rate. If the departed request is processed completely with a lower service rate, the evacuated VM will be spare. The state (k, j) will transfer to state $(k - 1, j)$ with transition rate $(k - j)\mu_0$. Otherwise, the evacuated VM will enter semi-sleep mode. The state (k, j) will transfer to state $(k - 1, j - 1)$ with transition rate $j\mu_1$.

For the case of $0 < k \leq c$ and $j = k$, all the occupied VMs are operating at a normal service rate. The departed request is processed completely with a normal service rate, the evacuated VM will enter semi-sleep mode. The state (k, j) will transfer to state $(k - 1, j - 1)$ with transition rate $j\mu_1$.

For the case of $k > c$, all the VMs in the system are occupied, and the evacuated VM will continue to provide service for the first request waiting in the buffer at the same service rate as before. The state (k, j) will transfer to state $(k - 1, j)$ with transition rate $(c - j)\mu_0 + j\mu_1$.

(3) The system level k remains unchanged which means that neither arrival nor departure occurs.

For the case of $0 < k \le c$ and $j \ne k$, there is (are) $(k - j)$ VM(s) operating at a lower service rate. If one of the $(k - j)$ VMs terminates a semi-sleep period, it will suspend its current service, and will enter into the awake period immediately and resume the service at a normal service rate. The state (k, j) will transfer to state $(k, j + 1)$ with transition rate $(k - j)\theta$.

For the case of $k > c$ and $j \ne c$, there is (are) $(c - j)$ VM(s) operating at a lower service rate. If one of the $(c - j)$ VMs terminates a semi-sleep period, it will suspend its current service, and will enter into the awake period immediately and resume the service at a normal service rate. The state (k, j) will transfer to state $(k, j + 1)$ with transition rate $(c - j)\theta$.

According to the discussion above, the state transition of the Markov chain can be illustrated in Figure 2.

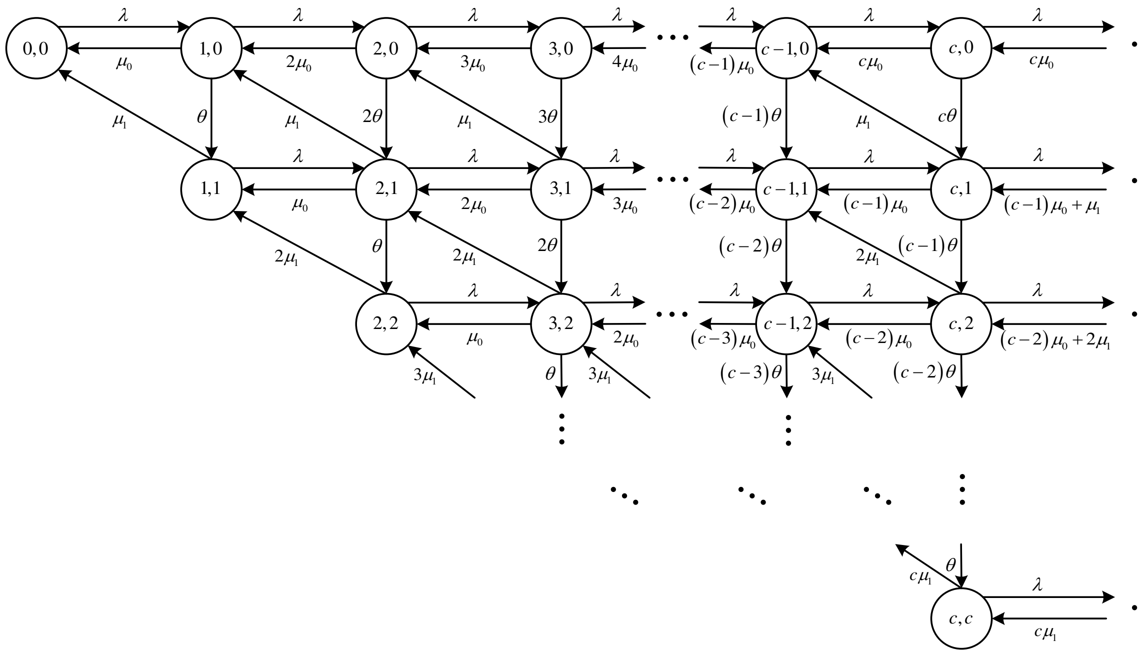

FIGURE 2. State transition of the Markov chain

3.2. **Steady-state distribution.** According to the state transition and the lexicographical sequence, the transition rate matrix $\boldsymbol{Q}$ of the Markov chain $\{L(t), J(t), t \ge 0\}$ can be written as a block-partitioned matrix as follows:

$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{A}_0 & \boldsymbol{C}_0 & & & & & \\ \boldsymbol{B}_1 & \boldsymbol{A}_1 & \boldsymbol{C}_1 & & & & \\ & \boldsymbol{B}_2 & \boldsymbol{A}_2 & \boldsymbol{C}_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \boldsymbol{B}_c & \boldsymbol{A}_c & \boldsymbol{C}_c & \\ & & & & \boldsymbol{B}_{c+1} & \boldsymbol{A}_{c+1} & \boldsymbol{C}_{c+1} \\ & & & & & \ddots & \ddots & \ddots \end{pmatrix} \tag{4}$$

The transition rate matrix $\boldsymbol{Q}$ is composed of three types of sub-matrices $\boldsymbol{B}_k$ $(k > 0)$, $\boldsymbol{A}_k$ and $\boldsymbol{C}_k$ $(k \ge 0)$. The elements of $\boldsymbol{B}_k$ denote the state transition rate for the system level decreasing by one. The elements of $\boldsymbol{A}_k$ denote the state transition rate for the system level remaining unchanged. The elements of $\boldsymbol{C}_k$ denote the state transition rate for the system level increasing by one. Each sub-matrix will be discussed in detail as follows.

For the case of the system level $k = 0$, $\boldsymbol{A}_0 = -\lambda$, $\boldsymbol{C}_0 = \lambda$.

For the case of $1 \leq k \leq c$, $\boldsymbol{B}_k$ is a $(k+1)\times k$ nonnegative matrix; $\boldsymbol{A}_k$ is a $(k+1)\times(k+1)$ square matrix, which has the negative diagonal elements and nonnegative off-diagonal elements; $\boldsymbol{C}_k$ is a $(k+1)\times(k+2)$ nonnegative matrix. $\boldsymbol{B}_k$, $\boldsymbol{A}_k$ and $\boldsymbol{C}_k$ can be represented as follows:

$$\boldsymbol{B}_k = \begin{pmatrix} k\mu_0 & & & & \\ \mu_1 & (k-1)\mu_0 & & & \\ & 2\mu_1 & (k-2)\mu_0 & & \\ & & \ddots & \ddots & \\ & & & (k-1)\mu_1 & \mu_0 \\ & & & & k\mu_1 \end{pmatrix}_{(k+1)\times k} \tag{5}$$

$$\boldsymbol{A}_k = \begin{pmatrix} \alpha_1 & k\theta & & & \\ & \alpha_2 & (k-1)\theta & & \\ & & \ddots & \ddots & \\ & & & \alpha_{t-1} & \theta \\ & & & & \alpha_t \end{pmatrix}_{(k+1)\times(k+1)} \tag{6}$$

where $\alpha_t = -(\lambda + (k-t+1)\mu_0 + (t-1)\mu_1 + (k-t+1)\theta)$, $1 \leq t \leq k+1$.

$$\boldsymbol{C}_k = \begin{pmatrix} \lambda & & & & 0 \\ & \lambda & & & 0 \\ & & \ddots & & \vdots \\ & & & \lambda & 0 \\ & & & & \lambda \; 0 \end{pmatrix}_{(k+1)\times(k+2)} \tag{7}$$

For the case of the system level $k \geq c+1$, we find that starting from the system level $(c+1)$, all the sub-matrices of $\boldsymbol{Q}$ remain repeated forever with $\boldsymbol{B}_{c+1}$, $\boldsymbol{A}_{c+1}$ and $\boldsymbol{C}_{c+1}$. We denote the repetitive $\boldsymbol{B}_{c+1}$, $\boldsymbol{A}_{c+1}$ and $\boldsymbol{C}_{c+1}$ as $\boldsymbol{B}$, $\boldsymbol{A}$ and $\boldsymbol{C}$, respectively. The sub-matrices $\boldsymbol{B}$, $\boldsymbol{A}$ and $\boldsymbol{C}$ are square matrices with order of $(c+1)\times(c+1)$ as follows:

$$\boldsymbol{B} = \begin{pmatrix} c\mu_0 & & & & \\ & (c-1)\mu_0+\mu_1 & & & \\ & & \ddots & & \\ & & & \mu_0+(c-1)\mu_1 & \\ & & & & c\mu_1 \end{pmatrix}_{(c+1)\times(c+1)} \tag{8}$$

$$\boldsymbol{A} = \begin{pmatrix} \beta_1 & c\theta & & & \\ & \beta_2 & (c-1)\theta & & \\ & & \ddots & \ddots & \\ & & & \beta_{l-1} & \theta \\ & & & & \beta_l \end{pmatrix}_{(c+1)\times(c+1)} \tag{9}$$

where $\beta_l{=}-(\lambda + (c-l+1)\mu_0 + (l-1)\mu_1 + (c-l+1)\theta)$, $1 \leq l \leq c+1$.

$$\boldsymbol{C} = \lambda\boldsymbol{I} \tag{10}$$

where $\boldsymbol{I}$ represents the unit matrix with order of $(c+1)\times(c+1)$.

From the structure of the transition rate matrix $\boldsymbol{Q}$, we find the Markov chain $\{L(t), J(t), t \geq 0\}$ is a specific quasi birth and death (QBD) process [8, 9]. In order to analyze the

QBD process, we need to solve for the minimal non-negative solution $\boldsymbol{R}$ of the matrix quadratic equation:

$$\boldsymbol{R}^2\boldsymbol{B} + \boldsymbol{R}\boldsymbol{A} + \boldsymbol{C} = \boldsymbol{0} \tag{11}$$

and this solution is called as rate matrix.

Based on Equations (8)-(10), we find that the sub-matrices $\boldsymbol{B}$, $\boldsymbol{A}$ and $\boldsymbol{C}$ are upper triangular matrices. So, the rate matrix $\boldsymbol{R}$ is also upper triangular matrix as follows:

$$\boldsymbol{R} = \begin{pmatrix} r_{00} & r_{01} & \cdots & r_{0(c-1)} & r_{0c} \\ & r_{11} & \cdots & r_{1(c-1)} & r_{1c} \\ & & \ddots & \vdots & \vdots \\ & & & r_{(c-1)(c-1)} & r_{(c-1)c} \\ & & & & r_{cc} \end{pmatrix}_{(c+1)\times(c+1)} \tag{12}$$

Substituting $\boldsymbol{R}$, $\boldsymbol{B}$, $\boldsymbol{A}$ and $\boldsymbol{C}$ into Equation (11), we obtain each element in the rate matrix $\boldsymbol{R}$ by the recursive method.

In fact, when the spectral radius $Sp(\boldsymbol{R}) < 1$, $\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_c$ can be determined uniquely by the following set of equations:

$$\begin{cases} (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_c)B[\boldsymbol{R}] = \boldsymbol{0} \\ \displaystyle\sum_{k=0}^{c-1} \boldsymbol{\pi}_k \boldsymbol{e}_1 + \boldsymbol{\pi}_c (\boldsymbol{I} - \boldsymbol{R})^{-1} \boldsymbol{e}_2 = 1 \end{cases} \tag{13}$$

where $\boldsymbol{e}_1$ is a $c(c+1)\times 1$ vector with ones, $\boldsymbol{e}_2$ is a $(c+1)\times 1$ vector with ones, and $B[\boldsymbol{R}]$ is given as follows:

$$B[\boldsymbol{R}] = \begin{pmatrix} \boldsymbol{A}_0 & \boldsymbol{C}_0 & & & \\ \boldsymbol{B}_1 & \boldsymbol{A}_1 & \boldsymbol{C}_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \boldsymbol{B}_{c-1} & \boldsymbol{A}_{c-1} & \boldsymbol{C}_{c-1} \\ & & & \boldsymbol{B}_c & \boldsymbol{R}\boldsymbol{B} + \boldsymbol{A} \end{pmatrix} \tag{14}$$

Furthermore, we obtain $\boldsymbol{\pi}_k$ $(k = c+1, c+2, \ldots)$ by the following equation:

$$\boldsymbol{\pi}_k = \boldsymbol{\pi}_c \boldsymbol{R}^{k-c}, \quad k > c \tag{15}$$

Substituting $\boldsymbol{\pi}_c$ obtained in Equation (13) into Equation (15), we obtain $\boldsymbol{\pi}_k$ $(k > c)$. Thus, the steady-state distribution $\boldsymbol{\Pi} = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots)$ of the Markov chain can be given mathematically.

4. **Performance Measures.**

In this section, we firstly define some performance measures to evaluate the proposed strategy. Then, we provide statistical experiments to investigate the performance trade-off.

4.1. **Performance measures.**

We define the latency of a request as the time duration from the instant the request arrives at the system to the instant the request is processed completely.

Based on the steady-state distribution of the Markov chain obtained in Section 3.2, we calculate the average number of requests in the system as $\sum_{k=0}^{\infty}\sum_{j=0}^{c} k\pi_{kj}$. Using Little's formula [10, 11], the average latency W of requests is given as follows:

$$W = \frac{1}{\lambda}\sum_{k=0}^{\infty}\sum_{j=0}^{c} k\pi_{kj} \tag{16}$$

In our proposed strategy, energy can be saved when the VMs operate at a lower service rate; however, additional energy will be consumed when the VMs switch to the normal service rate from the lower service rate. We define the energy-saving rate of system as the overall energy conservation per unit time.

Also, based on the steady-state distribution of the system, we calculate the average number of the spare VMs operating at a lower service rate as $N_{l0} = \sum_{k=0}^{c-1}\sum_{j=0}^{k}(c-k)\pi_{kj}$ and the average number of the busy VMs operating at a lower service rate as $N_{l1} = \sum_{k=1}^{c}\sum_{j=0}^{c-1}(k-j)\pi_{kj} + \sum_{k=c+1}^{\infty}\sum_{j=0}^{c-1}(c-j)\pi_{kj}$. The energy-saving rate S of system is given as follows:

$$S = (\omega_{h0} - \omega_{l0}) * N_{l0} + (\omega_{h1} - \omega_{l1} - \omega_2 * \theta) * N_{l1} \tag{17}$$

where ω_{h0} and ω_{h1} are the energy consumptions per unit time with a normal service rate on a spare VM and on a busy VM, respectively; ω_{l0} and ω_{l1} are the energy consumptions per unit time with a lower service rate on a spare VM and on a busy VM, respectively; ω_2 is the energy consumption for each switching from the lower service rate to the normal service rate.

4.2. **Statistical experiments.** In order to investigate the influence of system parameters on the system performance, we provide statistical experiments with analysis and simulation to show the variation tendency for the average latency of requests and the energy-saving rate of system. We perform the experiments in MATLAB R2010a software to obtain the analysis results. By establishing four classes: job, server, exponent and service in Eclipse, we obtain the simulation results.

Provided that the system is guaranteed in a steady state, the setting of the experiment parameters has no impact on the change trend of the performance measures. In statistical experiments, we fix experiment parameters as follows: the total number of VMs in the system as $c = 20$, the arrival rate of requests as $\lambda = 1.5$, the normal service rate as $\mu_1 = 0.6$, the energy consumption per unit time with a normal service rate on a spare VM as $\omega_{h0} = 0.2$, the energy consumption per unit time with a lower service rate on a spare VM as $\omega_{l0} = 0.1$, the energy consumption per unit time with a normal service rate on a busy VM as $\omega_{h1} = 0.25$, the energy consumption per unit time with a lower service rate on a busy VM as $\omega_{l1} = 0.15$, and the energy consumption for each switching from the lower service rate to the normal service rate as $\omega_2 = 1.2$. We show the influence of the semi-sleep parameter θ on the average latency W of requests and the energy-saving rate S of system for different lower service rates μ_0 in Figure 3 and Figure 4, respectively.

In Figure 3, we observe that as the semi-sleep parameter θ increases from 0.1 to 1.5, the average latency W of requests decreases in terms of the same lower service rate μ_0. When the semi-sleep parameter is bigger, the time length for the VMs being in the semi-sleep period is shorter. For this case, the VMs are more likely to provide service at a normal service rate and the requests will be processed quickly. So, the average latency of requests will decrease.

We also notice that as the lower service rate μ_0 increases from 0.1 to 0.5, the average latency W of requests decreases in terms of the same semi-sleep parameter θ. When the lower service rate is bigger, obviously, the requests will be processed quickly even in the semi-sleep period. So the average latency of requests will decrease.

In Figure 4, we find that as the semi-sleep parameter θ increases from 0.1 to 1.5, the energy-saving rate S of system decreases in terms of the same lower service rate μ_0. The bigger the semi-sleep parameter is, the more earlier the VMs will switch to the awake period from the semi-sleep period. For this case, the energy conservation in the semi-sleep period will be lower, and the energy consumption for the service rate transition will be greater. Therefore, the energy-saving rate of system will decrease.

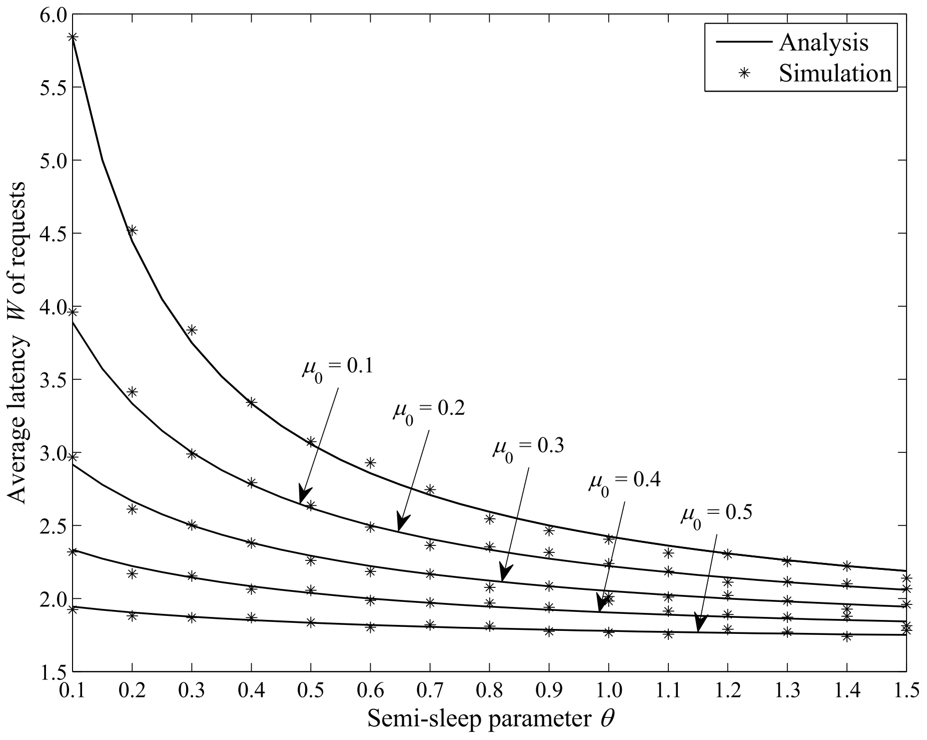

FIGURE 3. Average latency W of request versus semi-sleep parameter θ

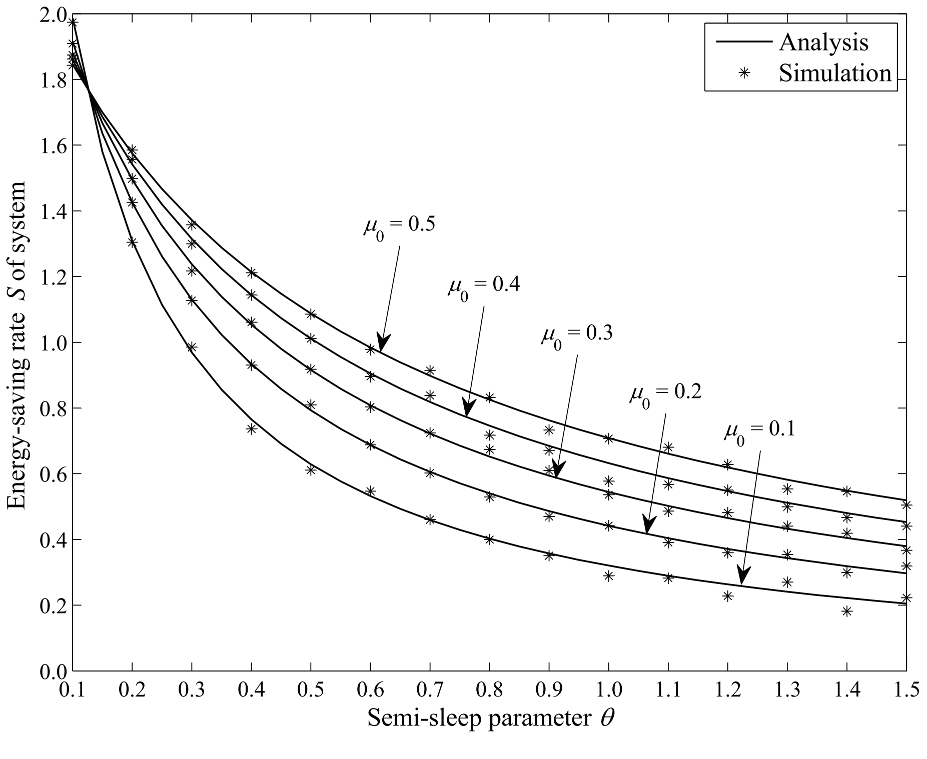

FIGURE 4. Energy-saving rate S of system versus semi-sleep parameter θ

We also see that for a smaller semi-sleep parameter, such as $\theta \leq 0.15$, as the lower service rate μ_0 increases, the energy-saving rate S of system decreases. When the semi-sleep parameter is smaller, the dominant factor influencing the energy-saving rate of system is the lower service rate in the semi-sleep period. The higher the lower service rate is, the greater the energy consumption will be. So, the energy-saving rate of system will decrease. On the other hand, for a bigger semi-sleep parameter, such as $\theta \geq 0.15$, the dominant factor influencing the energy-saving rate of system is the time length for the VMs providing service during the semi-sleep period. The higher the lower service rate is, the more quickly the requests will be processed, the more possible is that VMs will be spare at a lower service rate, moreover, the less frequently the VMs will switch to the normal service rate from the lower service rate. So, the energy-saving rate of system will increase.

Summarizing Figure 3 and Figure 4, we find that the analysis results match well with simulation results, we also find that there is a trade-off between the expense for the average latency of requests and the benefit for the energy-saving rate of system. This trade-off can be achieved by designing an appropriate semi-sleep policy.

5. **System Optimization.**

In this section, we establish a cost function of system to trade off the average latency of requests and the energy-saving rate of system. Then we optimize the semi-sleep parameter with the minimum cost function of system by using ant colony algorithm.

The total cost of system is defined as the difference between the time consumption of requests and the energy-saving benefit of system with the proposed strategy. So, we establish a cost function F of system as follows:

$$F = f_w W^2 - f_s S \tag{18}$$

where f_w is the coefficient for the second power of the average latency of requests; f_s is the coefficient for the energy-saving rate of system. W and S are obtained in Equations (16) and (17), respectively. Employing the parameters used in Subsection 4.2 and setting $f_w = 0.85$, $f_s = 5.0$ as an example, we illustrate how the cost function F of system changes along with the semi-sleep parameter θ for different lower service rate μ_0 in Figure 5.

As shown in Figure 5, for all the lower service rate μ_0, the cost function F of system firstly decreases and then increases as the semi-sleep parameter θ increases. Recall that both of the average latency of requests and the energy-saving rate of system will decrease as the semi-sleep parameter increases. When the semi-sleep parameter is smaller, the declining trend of the average latency of requests is greater than that of the energy-saving rate of system, and the average latency of requests is the dominate impact factor. Hence, there is a decreasing stage in the cost function of system when the semi-sleep parameter is smaller. When the semi-sleep parameter is bigger, the declining trend of the energy-saving rate of system is greater than that of the average latency of requests, and the energy-saving rate of system is the dominate impact factor. So, there is an increasing stage in the cost function of system when the semi-sleep parameter is bigger. Conclusively, there is a minimum cost function F^* of system with an optimal semi-sleep parameter θ^*.

We note that the average latency of requests and the energy-saving rate of system in Equations (16) and (17) are difficult to be given in close-forms. In addition, the strict monotonicity for the cost function of system is difficult to be addressed. In order to obtain the optimal semi-sleep parameter, we turn to an intelligent searching ant colony algorithm (ACA) [12, 13]. In ACA, we regard the semi-sleep parameter as the position of an artificial ant, and the cost function of system as the pheromone information of an artificial ant. After several iterations, we can obtain the optimal semi-sleep parameter (the

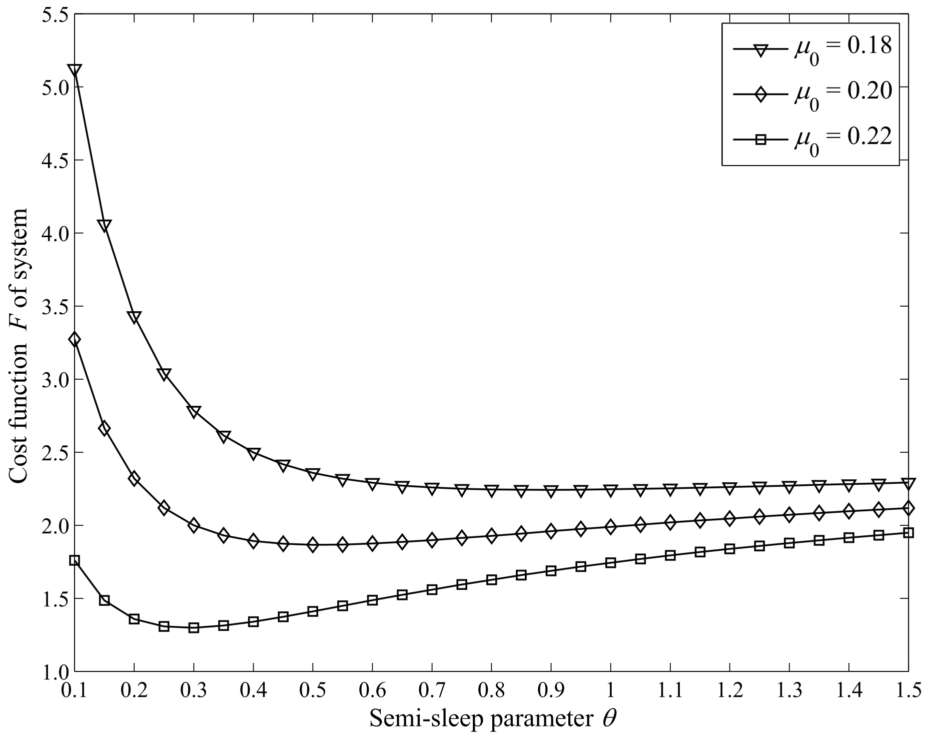

FIGURE 5. Cost function F of system versus semi-sleep parameter θ

TABLE 1. Optimal semi-sleep parameter θ^* and minimum cost function F^* of system

Lower service rate	Optimal semi-sleep parameter	Minimum cost function
μ_0	θ^*	F^*
0.18	0.8820	2.2420
0.20	0.5115	1.8660
0.22	0.2885	1.2983

best position of an artificial ant) with the minimum cost function of system (the smallest pheromone information of an artificial ant). The numerical results of the optimal semi-sleep parameter θ^* and the minimum cost function F^* of system for different lower service rate μ_0 are summarized in Table 1.

In Table 1, we observe that the optimal semi-sleep parameter θ^* gets smaller as the lower service rate μ_0 increases. When the lower service rate μ_0 is bigger, the average latency of W requests becomes smaller, and then the energy-saving rate S of system has more development potential.

6. Conclusions.

In this paper, by introducing semi-sleep mode, we proposed an energy-efficient VM scheduling strategy on the cloud platform. We established a continuous-time queueing model with multiple asynchronous working vacation to capture the stochastic behavior of the proposed strategy. By using the method of QBD process and matrix geometric solution, we analyzed the system model in steady-state and derived some performance measures in terms of the average latency of requests and the energy-saving rate of system. Statistical experiments with analysis and simulation show that there is a trade-off between the average latency of requests and the energy-saving rate of system. For this, we built a cost function of system and optimized the semi-sleep parameter numerically.

As a future research, we will investigate the Nash equilibrium and the social optimal behavior of cloud users with the energy-efficient VM scheduling strategy based on semi-sleep mode.

Acknowledgment. This work was supported in part by National Natural Science Foundation (No. 61872311, No. 61472342), Hebei Province Natural Science Foundation (No. F2017203141) and Open Project of Science and Technology on Communication Networks Laboratory (No. KX17600025), China.

REFERENCES

[1] P. Sasikala and S. Suresh, An adaptive approach for efficient energy saving technique in enterprise cloud data centers, *Advances in Natural and Applied Sciences*, vol.10, no.6, pp.164-169, 2016.

[2] A. Paya and D. Marinescu, Energy-aware load balancing and application scaling for the cloud ecosystem, *IEEE Trans. Cloud Computing*, vol.5, no.1, pp.15-27, 2017.

[3] F. Rossi, M. Xavier, C. Rose et al., E-eco: Performance-aware energy-efficient cloud data center orchestration, *Journal of Network and Computer Applications*, vol.78, pp.83-96, 2017.

[4] L. Servi and S. Finn, M/M/1 queues with working vacations (M/M/1/WV), *Performance Evaluation*, vol.50, no.1, pp.41-52, 2002.

[5] S. Jeyakumar and B. Senthilnathan, Modelling and analysis of a bulk service queueing model with multiple working vacations and server breakdown, *RAIRO – Operations Research*, vol.51, no.2, pp.485-508, 2017.

[6] Y. Wang, L. Chen, Z. Ma et al., Geom/Geom/c queue with multiple-server synchronously working vacations, *Journal of Yanshan University*, vol.36, no.3, pp.259-264, 2012.

[7] C. Goswami and N. Selvaraju, Phase-type arrivals and impatient customers in multiserver queue with multiple working vacations, *Advances in Operations Research*, vol.2016, no.2, pp.1-17, 2016.

[8] S. F. Jin and S. Chen, An energy saving strategy in digital cognitive radio networks and its performance assessment, *International Conference on Intelligent Computing*, vol.10, no.6, pp.1367-1373, 2016.

[9] D. Bini, G. Latouche and B. Meini, Shift techniques for quasi-birth and death processes: Canonical factorizations and matrix equations, *Applied Numerical Mathematics*, vol.116, pp.24-36, 2017.

[10] M. Boon, O. Boxma, O. Kella et al., Queue-length balance equations in multiclass multiserver queues and their generalizations, *Queueing Systems*, vol.86, nos.3-4, pp.277-299, 2017.

[11] H. Zhang and G. Zhou, M/M/1 queue with m kinds of differentiated working vacations, *Journal of Applied Mathematics and Computing*, vol.54, nos.1-2, pp.213-227, 2017.

[12] Y. X. Sun, J. L. Shang, J. X. Liu et al., An improved ant colony optimization algorithm for the detection of SNP-SNP interactions, *International Conference on Intelligent Computing*, pp.21-32, 2016.

[13] K. Salama and A. Abdelbar, Learning cluster-based classification systems with ant colony optimization algorithms, *Swarm Intelligence*, vol.11, nos.3-4, pp.211-242, 2017.