Noname manuscript No.
(will be inserted by the editor)

Enhancing Service Capability with Multiple Finite Capacity Server Queues in Cloud Data Centers

Binh Minh Nguyen · Dang Tran · Giang Nguyen

Received: date / Accepted: date

Abstract Cloud computing took a step forward in the efficient use of hardware through virtualization technology. And as a result, cloud brings evident benefits for both users and providers. While users can acquire computational resources on-demand elastically, cloud vendors can also utilize maximally the investment costs for data centers infrastructure. In the Internet era, the number of appliances and services migrated to cloud environment increases exponentially. This leads to the expansion of data centers, which become bigger and bigger. Not just that these data centers must have the architecture with a high elasticity in order to serve the huge upsurge of tasks and balance the energy consumption. Although in recent times, many research works have dealt with finite capacity for single job queue in data centers, the multiple finite-capacity queues architecture receives less attention. In reality, the multiple queues architecture is widely used in large data centers. In this paper, we propose a novel three-state model for cloud servers. The model is deployed in both single and multiple finite capacity queues. We also bring forward several strategies to control multiple queues at the same time. This approach allows to reduce service waiting time for jobs and managing elastically the service capability for the whole system. We use CloudSim to simulate the cloud environment and to carry out the experiments in order to demonstrate the operability and effectiveness of the proposed method and strategies. The power consumption is also evaluated to provide insights into the system performance in respect of performance-energy trade-off.

Keywords Cloud computing · multiple server queues · finite capacity · queuing theory · Markov chain · performance analysis

Binh Minh Nguyen
School of Information and Communication Technology
Hanoi University of Science and Technology
E-mail: minhnb@soict.hust.edu.vn

Dang Tran
School of Information and Communication Technology
Hanoi University of Science and Technology
E-mail: dangtv18@gmail.com

Giang Nguyen
Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, 845 07 Bratislava, Slovakia
E-mail: giang.ui@savba.sk

1 Introduction

Nowadays, billions of people are using Internet and its services. As a consequence, supercomputers and data centers must provide high-performance computing services to huge numbers of Internet users concurrently. Within this trend, cloud technologies emerge as an effective solution for data centers in order to be able to meet all user requirements. The key distinction point between cloud and other computing paradigms is the adoption of virtualization, which enables the capability to provide resources on demand quickly and dynamically [25]. At present, clouds offer many services for customers including hardware, software and data sets, and a promising technology for a significant costs reduction for both sides - for the vendors and also for the users.

Cloud computing is in the era of vigorous development with data centers becoming larger and larger. In cloud model, computational jobs are allocated into data centers, where a huge number of physical or virtual servers is available. Thus, data centers must ensure the accessibility, offer the transparent infinite resource capacity and provide a large-scale architecture in order to serve efficiently the incoming jobs. At the same time, it is

required that the well-designed scheduling and management algorithms for cloud datacenters would centrally manage physical or virtual servers; provide flexible and resilient resources that help enterprises to accommodate business growth and to achieve their goals in the cost reduction. In specific, one of the service quality (QoS) aspects must be ensured under the control of scheduling and management algorithms: if a large number of jobs comes into data centers with varied intensities, the jobs must be processed as quickly as possible. The system also has to keep the number of refused jobs at the lowest possible level. Companies together with infrastructure vendors concern about the power consumption to guarantee a reasonable trade-off between energy-saving and system performance.

Recently, many research works have been carried out in the field of energy-saving for cloud data centers. In the case of capacity of job queue treated as infinite, the authors of works [5] and [26] propose strategies to control servers, which have only two states ON and OFF. Their goal is to calculate exactly the waiting time of jobs inside the cloud systems using the queueing theory and thus optimize the energy consumption. According to the intensity of arrival of jobs, the work described in [21] presents an algorithm to turn the cloud storage servers on and off. In contrast to others, this work defines active servers with three states: working, standby and idle. However, the authors only develop the algorithm, which enables cloud system to automatically switch from idle to standby mode. If the number of idle and standby servers is zero, the system naturally has to turn on some servers to serve the arriving jobs. This process also takes a long time until the system is ready to run the jobs. The authors still do not determine in specific when and how many storage servers are in standby, idle or power off modes. In our paper [24], we develop a model involving three states (OFF, MIDDLE and ON) for servers and propose the management algorithms to control dynamically the number of MIDDLE state based on the intensity of arriving jobs. In this way, the work decreases the waiting time of jobs for the turn-on process of server in data centers and improves QoS for the whole system.

In the case of capacity of job queue treated as finite, Qi Zhang and et al. in [30] propose a control-theoretic solution for the dynamic resource provisioning problem to minimize the total energy cost while meeting the performance objective. Through extensive analysis and simulation using real workload traces from Google's compute clusters, the authors show that their proposed framework can achieve a significant reduction in energy cost, while maintaining an acceptable average scheduling delay for individual tasks. Hamzeh Khazaei and et al. [14] analyze and evaluate cloud computing centers using M/G/m/m + r queue system with single job queue and a job buffer of finite capacity. However, the approach above does not use any intermediation state to decrease service time. Until now, for the model of single finite-capacity server queue, no research has been made that would develop a transition process with an intermediate server state (similar to standby mode in [21] and MIDDLE state in [24]).

While models of single job queue have been extensively studied, models of multiple job queues have received less attention. The authors of [7] bring forward a model with multiple finite queues for computer networks. Based on Powell algorithm [28], the achieved results of the work mainly contribute to the routing optimization and its performance evaluation. The researches described in [11] and [10] refer to multiple queues for server systems, in which jobs are classified according to their sizes and allocated into available queues. In this way, this research turns towards minimizing response time during task processing. However, the ability of elasticizing queue numbers still has not been touched. As analyzed above, this is an important factor that helps to build a large-scale architecture for data centers, and the dynamic control problem of multiple queues still has not received a relevant interest.

In practice, most of job queues have a certain capacity, even though server quantity of a data center can be considered as unlimited. Besides, in order to ensure the capability to quickly serve the massive scale of incoming jobs, servers are often divided into many different queues to process tasks simultaneously. Inspired by this fact, we firstly propose a single finite-capacity server queue model with an intermediation state, called MIDDLE, for servers. With this state, a reasonable number of servers is powered and kept available to wait and serve incoming jobs. Due to the servers in the MIDDLE state, the booting process can be eliminated and thus it can decrease the waiting time while the system is performing jobs. Like other researchers, we also mainly rely on Poisson distribution and Markov chain to design algorithm which enables to control whether the number of servers in MIDDLE state is increased or decreased accordingly. Next, we develop a large-scale architecture for data centers with many finite-capacity queues built above. The core of the architecture is a scheduling algorithm that enables to manage dynamically the queue number according to two factors: the queue capacity and diverse intensity of incoming jobs. We bring out several mechanisms for the elastic management including the transition states for server queues and their policies, low and high thresholds for queues, and job scheduling for the entire system. Our approach thus will

enhance the job service capability of cloud data centers. Finally, our three states of servers in single and multiple finite-capacity queues are successfully simulated and examined to show their effectiveness. We also evaluate energy consumption of each proposed model.

The rest of the paper is organized as follows. Section 2 gives a detailed analysis of existing works related to job allocation and power optimization techniques. We focused on the survey of single and multiple finite-capacity server queues as studied in the past. Section 3 introduces the novel three-state model for servers. The model is demonstrated by using the Markov chain. An algorithm to control transition processes for servers in single finite-capacity queue is also proposed here. In Section 4 we focused on the issue how to dynamically manage multiple server queues by bringing forward an elastic architecture for data centers in order to handle simultaneously a lot of jobs. Next, the simulation experiments include results to prove the operability of our models as described in Section 5. In this section we analyzed the power consumption of both innovations to evaluate the effectiveness of our approach. Our contributions are summarized in Section 6, where we also outline directions for the future work.

2 Related Works

The effective job allocation and optimizing power consumption are the main challenges in large cloud data centers. A lot of researches in the field on energy-saving and job scheduling has been carried out in recent times. According to techniques of the approaches, the research activities can be roughly divided into the following groups.

The first group of techniques has been focused on building data centers using low-energy hardware components. Hamilton [8] optimizes electricity consumption by putting forward an architecture for server racks that use a low-power equipment like AMD Athlon processors. In the similar way, the authors of [15] also design cloud storage system applying saving-power components for Hadoop platforms. The authors proved by experiments that with their approach the system performance decreased only negligibly, but many times less energy consumption was identified. However, for the existing data centers, due to cost, the mass hardware replacement is not feasible. In addition, the issue of the job scheduling optimization had not been mentioned in these research works.

The second group of techniques exploits migration technologies in the cloud environment to minimize the number of physical servers in the running state. Some of works as [18], [23] and [29] are classified into this group. The evident goal of the studies is to save the power for data centers. Technically, most approaches aim at migrating virtual machines (VMs) and data from low-load to high-load physical servers as much as possible. In this way, the number of idle physical servers can be turned off immediately to reduce the electricity consumption. The authors in [3] and [9] develop an adaptive heuristic algorithm for dynamic consolidation of virtual machines and containers. The methods are based on the resource utilization history of the VMs/containers on physical servers. Lin and et al. [17] present VM provision strategies for cloud systems where two algorithms are proposed including dynamic round-robin (DRR) and hybrid (combination among DRR and First-Fit) for energy aware VM scheduling and consolidation.

The third group of techniques concentrates on controlling servers based on active, inactive hardware states and queue algorithms. The techniques closely relate to our work described in this paper. Generally, most studies in this group model resource requirements using the Markov chain, and simulate single job queue except few cases that are referred at the end of this section.

According to the number of jobs in one queue that is treated as infinite or finite, this group can be divided into two classes. The first concerns the infinite capacity of job queues. In [5], [22], [12] and [26], servers have two states: OFF (cannot serve jobs) and ON (are processing jobs). In order to minimize the number of servers in the ON state, most of the authors provide management mechanisms to lead arriving jobs into available ON servers instead of turning on many OFF servers, if it is not really necessary. Thus, the processes of turn on and off are controlled based on the intensity of arriving jobs. Another approach is applied in some works by adding a state for servers. For example, in [21] there are three server states that are defined: working (the server is serving jobs), standby (waiting for arriving jobs) and idle (finished job processing). As mentioned in the Section 1, although the authors offer algorithms to control automatically the switch processes between states, they still have not developed any solution to determine concretely how many servers are sufficient in each state with varied job intensities, and precise moment when servers are switched on to ensure the minimum power consumptions. The work described in [16] introduces a framework for the migration of applications from a mobile to cloud environment. The authors also used a three-state model (power-off, pause and running) for VMs deployed on servers. With the pause state, when smart phone applications are moved to clouds, they will be deployed quickly rather than wait for VM booting from the power-off state. Unfortunately, there is also no control mechanism for VM shift status presented in

this paper. In our previous work [24], we also build a model with three states for cloud servers called OFF, MIDDLE and ON. The main idea in this model is to keep some servers in MIDDLE state which is the intermediate state in the process of turning servers from OFF to ON. Thanks to the state, the waiting time of jobs for OFF servers to startup will be reduce because MIDDLE servers can be switched to ON state faster than OFF servers. Thus, the system response time for incoming jobs decreases and QoS is improved (in the aspect of time service). Although we put forward the solution to calculate exactly the number of MIDDLE servers based on arrival intensity and the reasonable moments to turn on them, but the case of finite-capacity queue has still not been examined in this work.

The other technique class in this category deals with the finite capacity of job queue. The studies related to the class adhere to the situation in data centers where servers are often separated into many clusters for easier and more efficient management. At the same time, each cluster involves a determined number of servers. For instance, the research presented in [6] evaluates the performance of cloud with a queue of VMs. The authors use the finite multi-server queues model, where the deployments of web applications are considered as they were in a queue and the VMs are regarded as service providers. In [31], the authors propose a new way to allocate resources dynamically for the spot market in the cloud environment. The work also gives the finite capacity for each resource type in each data center. The authors model an actual case of a single cloud provider and address the question how to meet the customer demands the best in terms of both supply and price in order to maximize the providers revenue and customer satisfactions while minimizing energy cost. To achieve their goals, the research simulates this problem as a constrained discrete-time optimal control problem and uses Model Predictive Control (MPC) to find the convenient solution. The study presented in [14] describes a novel approximate analytical model for the performance evaluation of cloud server farms with the solution to obtain the accurate estimate of the complete probability distribution of the request response time. In their work, the queue system is called M/G/m/m+r. In [27], Phung-Duc uses M/M/c/K/SETUP model to analyze the performance of the finite-capacity queueing system, in which servers are managed with two states ON and OFF. However, for this technique class, there are no studies that would propose an intermediate state in order to reduce waiting time while processing jobs in a finite-capacity server queue.

It is clearly to see that all the works described above focus on single queue system while studies related to multiple queues have received less attention. In the context of computer network, the authors of [7] used Powell algorithm [28] to optimize the routing processes. Their experimental tests simulate a model consisting of many server queues. The works in [20] and [19] present the ability of stabilizing performance in multiple server queues with time-varying arrivals rates. The work is based on the customer feedback and abandonment to provide a flexible solution that can be readily fit to system data. In contrast to most of techniques referred above, in this work the Markov routing is not assumed. Instead, the model uses a history-dependent Bernoulli routing [2]. The authors of this work also test their solutions with three server queues and gained results are satisfactory. Returning to the Markov chain approach, some others classify the jobs into several types based on their sizes. Theoretically, the number of queues in the system equals to the number of size types which is often not more than five. Typical examples are described in [11] and [10]. Although several works in this class were mentioned to multiple queues using a control mechanism, the capability to dynamically control a large number of different queues is still the imperative task to manage cloud data centers well.

The research presented in this paper is closely related to the third technique category mentioned above. Our main contributions are to propose a finite-capacity server queue model with three states of physical machines. Based on that, we have developed a large-scale architecture for cloud data centers, which contain multiple finite-capacity queues. The number of queues can be changed according to the rate of incoming jobs. The major differences between the existing research and our research are as follows:

1. We define a novel three-state model for servers, namely: OFF, MIDDLE and ON, where with an intermediation state called MIDDLE, a certain number of servers is powered and available to wait for arriving jobs. Based on the state, we can eliminate waiting time of turning on the server process. When jobs arrive at system, MIDDLE servers only go through SETUP mode to become ON servers to process them. Thus, the service waiting time will be reduced significantly.
2. We apply the three-state model to a single finite-capacity queue. Mathematically, we express and demonstrate successfully the model based on the Markov chain and queueing theory. We also bring forward algorithms to reckon and keep a suitable number of servers in the MIDDLE state to guarantee that there are always the MIDDLE servers inside the system.

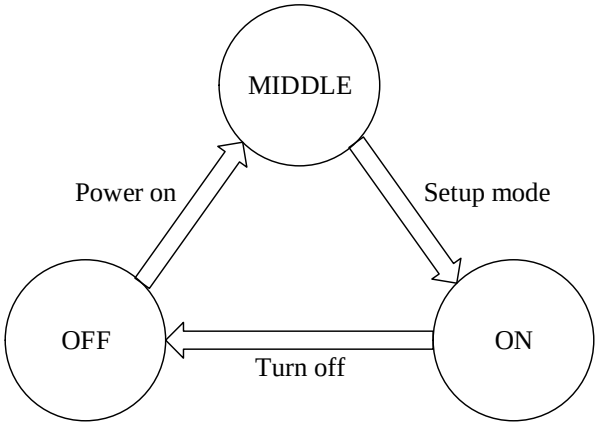

Fig. 1 State transition diagram of the physical servers

3. The proved finite capacity queue model above plays the essential role when we propose a large-scale architecture for cloud data centers, in which we use multiple finite-capacity server queues at the same time to serve arriving jobs. Relying on the architecture, incoming jobs are handled quickly instead of a single queue usage. To reach the goal, we designed several algorithms and strategies for scaling the queue number dynamically and horizontally. Specifically, we used a management server for the central coordination of jobs, we proposed states and their transitions of clusters inside data centers (each cluster corresponds with a single finite-capacity server queue), we designed queue thresholds and job scheduling policies for dynamically increase and decrease of the number of clusters based on different incoming rates.
4. To assess the energy consumption for our proposals from the obtained experimental results, we provide insights into the system performance from the point of view of performance-energy trade-off in these finite-capacity models using three server states.

3 Server farm with single finite-capacity queue

3.1 Three-state Model of Servers (ON/OFF/MIDDLE/ c/K model)

As proposed above, in the finite-capacity model, we set three states for each server, namely ON, MIDDLE and OFF. These states are defined specifically as follows:

Definition 1 ON is a state assigned for servers if they are turned on power supply and performing or ready to immediately perform arriving jobs.

Definition 2 MIDDLE is the intermediate state of servers in the switch process from OFF to ON. This state is reached after hardware and base software (e.g. operating system, required applications) are started up completely. MIDDLE servers cannot perform any jobs immediately. To serve jobs, the servers must be switched to ON by going through SETUP mode and prepare a suitable running environment.

Definition 3 OFF is a state assigned for servers if they are turned off from the power supply. In the OFF state, servers cannot perform any jobs.

The MIDDLE state in our proposed model is totally different from other states in the existing models such as Idle or Standby in [5] and [21]. The Idle is a status of ON servers when they are not serving any job. In our model, because these servers immediately perform a new arriving job or are turned off if there is no job in queue, we consider the lifetime of this status as zero. Standby refers to a low power state of servers that is usually used to save energy consumption by switching idle servers to Standby mode. The main difference between MIDDLE and Standby is that only ON server can be switched to Standby mode directly while MIDDLE s can be directly reached only by OFF servers.

After power-on process, servers must go through SETUP mode to create an environment for running jobs. This is also consistent with the conditions: servers need to be installed and software configured in order to produce platforms for deployment and operation of applications. Furthermore, the SETUP process often takes a diverse time period depending on the deployed software platform types. The SETUP mode is defined as follows:

Definition 4 SETUP mode is a process, in which MIDDLE servers are set up and configured to create a suitable environment for serving jobs. After SETUP process, MIDDLE servers will become ON servers.

The transition among these states is illustrated in Fig. 1. According to Definition 1, a server is in ON state if it is ready for processing a new job or it is processing a job. We assume that at most one job can be carried out inside ON server at any point of time. If there are no jobs arriving into the system, ON servers are turned off immediately to save energy.

Generally, OFF server definitely consumes certain costs (hardware response delays or power consumption) to switch to ON. In our proposed model, we divide the turn-on process into two stages: switching from OFF to MIDDLE, called power-on, and from MIDDLE to ON, called setup mode. The first stage is the process of powering the OFF server, and the second stage is the process of setting up or configuring the MIDDLE server (via SETUP mode). The MIDDLE servers cannot run

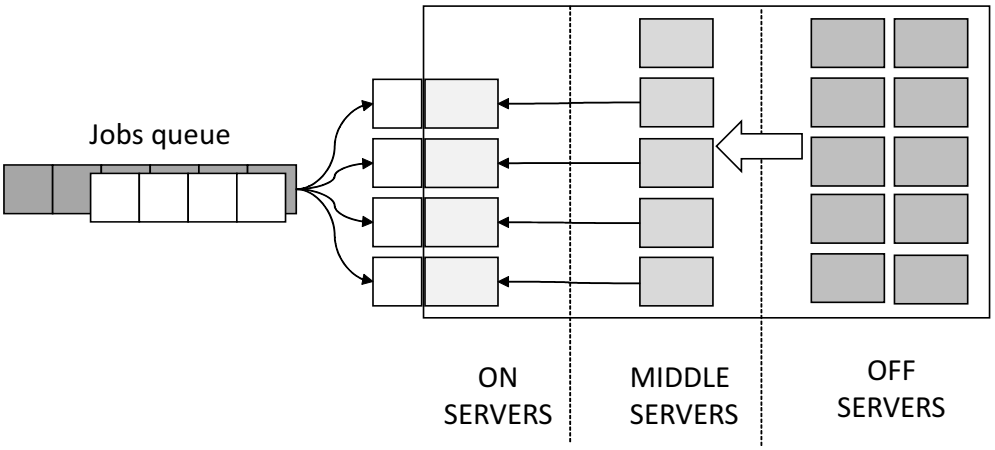

Fig. 2 Model of datacenter with one jobs queue

any jobs until they are switched to ON state successfully through the SETUP mode process. We consider the time for the first stage is invariable (denoted by $t_{OFF \to MIDDLE}$) with identical servers hardware configurations, and the time for the second stage (SETUP time) conforms to an exponential distribution with expectation $1/\alpha$ as the condition discussed above. So in summary, the time for a server is switched from OFF to ON via MIDDLE that is expressed by the following formula:

$$t_{OFF \to ON} = t_{OFF \to MIDDLE} + t_{MIDDLE \to ON} \quad (1)$$

in which $t_{MIDDLE \to ON}$ is time period during which servers perform the SETUP process.

Similarly to the most studies in the third category analyzed in Section 2, we also suppose that jobs arrive at the system according to Poisson process with rate λ, and the time to finish a job (service time) has an exponential distribution with the expectation $1/\mu$.

Because of MIDDLE state, we propose a new power management policy for our three-state model, called ON/OFF/MIDDLE policy, which is a variation of the ON/OFF policy presented in [5]. Under this policy, if a new job comes into the system, it is put into a queue and waits for MIDDLE server switched to ON state or an idle ON server to be served. When a server is switched from MIDDLE to ON successfully or ON server just completes a job (it becomes idle server), the server will execute a job in queue in compliance with FIFO rule. In case the queue is empty, ON servers do not have any tasks to perform, then they are immediately turned off. In our model, in the proportion to the job number in queue, a certain number of MIDDLE servers could be put simultaneously to SETUP process.

At any given time, there are i ON servers ($i \leq c$), and j jobs ($j \geq i$) in the system, where c denotes the total number of all servers in the finite-capacity system. We set three types of jobs as follows:

- *running* jobs, which are running in ON servers;
- *setting up* jobs, which are in the queue and waiting for servers in SETUP mode switched to ON state successfully;
- *waiting* jobs, which are in the queue, but could not find any available MIDDLE server switched to SETUP mode.

After the server is powered successfully (to switch from OFF to MIDDLE state), if there is a *waiting* job in the queue, the server will be put into SETUP mode. At this moment, the *waiting* job becomes a *setting up* job. Like ON servers, to reduce energy consumption, in case there are no jobs in the queue, SETUP servers are also turned off.

For example, when a *running* job J_1 has been completed, its server S_1 will execute job J_2 at the head of queue without going through SETUP process. Although at the time, J_2 is a *setting up* job and it is waiting for another server S_2 in SETUP mode to be served. The S_2 will not be used for job J_2 but it will be transferred to perform *waiting* job J_3 if J_3 waiting in the queue. And then J_3 becomes *setting up* job. If J_3 does not exist, there is no job waiting in the queue, the server S_2 will be turned off.

The ON/OFF/MIDDLE policy leads to the fact that the number of servers in SETUP mode is always less than or equals to the number of jobs in the queue (including *setting up* jobs and *waiting* jobs). The server number in SETUP is also limited by a total number of servers kept in MIDDLE state because the system cannot put more servers in SETUP mode if it does not have MIDDLE servers. Except for the case when all servers are in ON state or SETUP, if there is at least one available MIDDLE server at any point of time a

new job comes into system, the number of servers in SETUP equals $min(j-i, c-i)$.

In this finite-capacity model, the capacity of the system is K, that means at most K jobs are allowed to stay inside the system consisting of *running* jobs performed in ON servers, *setting up* and *waiting* jobs in the queue. As a result, whenever the system has K jobs, then new arriving jobs will be blocked. Our three-state model for server farm with single finite-capacity queue is denoted by ON/OFF/MIDDLE/c/K model, shown in Fig. 2.

As introduced above, in order to decrease the service time while servers are switched from OFF to ON, we put forward a management strategy for servers based on an intermediate state called MIDDLE. Based on arrival rate of jobs, the strategy determines concretely a number of OFF servers, which are turned to MIDDLE and kept ready in this state. There are two cases which can occur with MIDDLE servers when a new job comes into the system. In the first case, at least one MIDDLE server is available, the MIDDLE server will be switched to ON state via SETUP mode. Hence, the waiting time of this job in the system only equals the SETUP time, not including the server power-on time ($t_{OFF \to MIDDLE}$). In the second case, there is no server in MIDDLE state, so the job must wait for OFF server to be switched to MIDDLE state. Let us assume that the probability of the first case is P_{MIDDLE} ($Max\ P_{MIDDLE} = 1$), then the probability of the second case is $1 - P_{MIDDLE}$. Because waiting time is reduced in the first case, the mean waiting time of all jobs is reduced. It is clear that the larger P_{MIDDLE} is, the smaller mean waiting time of jobs.

In order to reduce the waiting time of jobs, our strategy is to increase the probability P_{MIDDLE}. In other words, our strategy is based on calculating the number of servers which are switched to MIDDLE state. This number must be big enough to maximize the probability P_{MIDDLE}, but not too big because of the significant energy consumption of MIDDLE servers. In this way, we use mathematical model to analyze ON/OFF/MIDDLE/c/K model in the case of $P_{MIDDLE} = 1$, then the result is used to build a control algorithm for the server management to meet the requirements of such a system.

The state of the system managed by ON/OFF/ MIDDLE policy is represented by a triplet (m, i, j) where m, i, j is the number of MIDDLE servers, the number of ON servers and the number of jobs, respectively. Unfortunately, it is complicated to represent the state transition of the system with state (m, i, j). However, if $P_{MIDDLE} = 1$, that means there is at least one MIDDLE server at any point of time if a new job arrives at system, the number of MIDDLE servers (m) does not effect to the probability of the transition of pairs (i, j). So, we can use ON/OFF/MIDDLE/c/K Markov chain with state (i, j)(Fig. 3) to represent the state transition of the server farm with a single finite-capacity queue in the case $P_{MIDDLE} = 1$.

3.2 Mathematical Model and Control Algorithm

3.2.1 Mathematical Model

Theoretically, the ON/OFF/MIDDLE/c/K Markov chain is similar to M/M/c/K/SETUP model proposed by Phung-Duc in [27], in which a simple recursion to calculate the stationary distribution of the system state for the M/M/c/K/SETUP model is presented. In our study, we also inherit the Lemma 1, which was proved in [27]; the Lemma 2 and Lemma 3 were improved in our previous work [24] and then we use these lemmas to improve the Theorem 1.

Lemma 1 *All the limit probabilities $\pi_{i,j}$ are expressed in term of $\pi_{0,0}$ by the following equations:*

- *For $i = 0$:*

$$\pi_{0,j} = b_j^{(0)} \pi_{0,j-1} \tag{2}$$

where $b_j^{(0)} = \frac{\lambda}{\lambda + \min(j,c)\alpha}, (j = 1, 2, \ldots, K-1)$ *and* $b_K^{(0)} = \frac{\lambda}{c\alpha}$.

$$\lambda \pi_{0,0} = \mu \pi_{1,1} \tag{3}$$

- *For $1 \le i \le c-1$:*

$$\pi_{i,j} = a_j^{(i)} + b_j^{(i)} \pi_{i,j-1}, \quad j = i+1, i+2, \ldots, K-1, K \tag{4}$$

$$(i+1)\,\mu \pi_{i+1,i+1} = \sum_{j=i+1}^{K} (j-i)\,\alpha \pi_{i,j} \tag{5}$$

where

$$a_j^{(i)} = \frac{i\mu a_{j+1}^{(i)} + min\,(c-i+1, j-i+1)\,\alpha \pi_{i-1,j}}{\lambda + min\,(c-i, j-i)\,\alpha + i\mu - i\mu b_{j+1}^{(i)}},$$

$$b_j^{(i)} = \frac{\lambda}{\lambda + min\,(c-i, j-i)\,\alpha + i\mu - i\mu b_{j+1}^{(i)}},$$

$j = K-1, K-2, \ldots, i+1$, *and*

$$a_K^{(i)} = \frac{(c-i+1)\,\alpha \pi_{i-1,K}}{(c-i)\,\alpha + i\mu}, \qquad b_K^{(i)} = \frac{\lambda}{(c-i)\,\alpha + i\mu}$$

- *For $i = c$:*

$$\pi_{c,j} = a_j^{(c)} + b_j^{(c)} \pi_{c,j-1}, \qquad j = c+1, c+2, \ldots, K-1, K \tag{6}$$

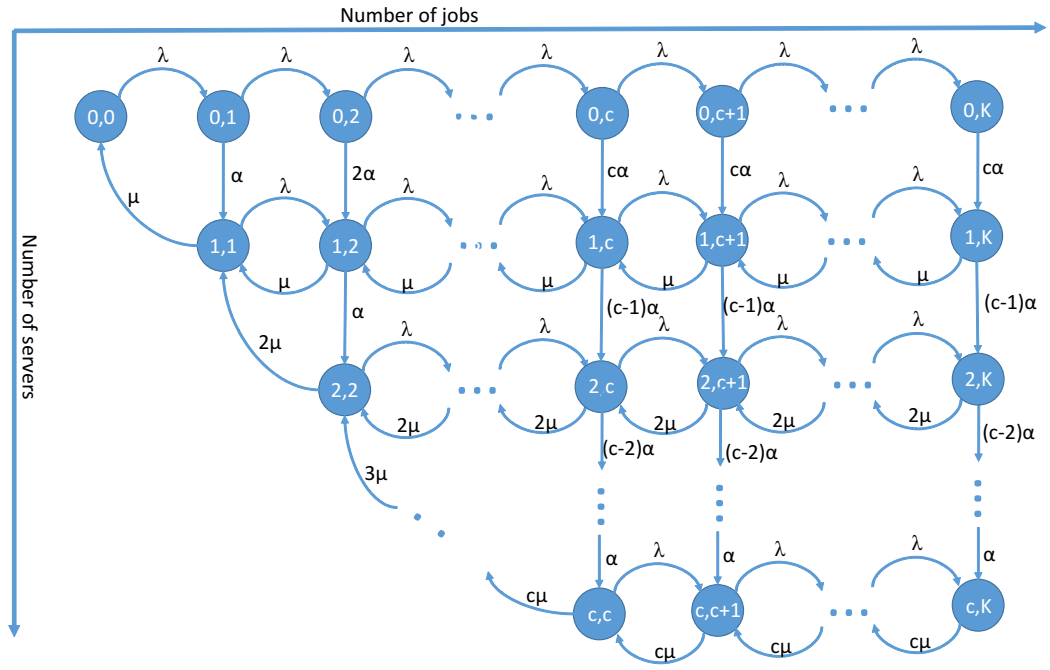

Fig. 3 ON/OFF/MIDDLE/c/K Markov chain

where

$$a_j^{(c)} = \frac{c\mu a_{j+1}^{(c)} + \alpha\pi_{c-1,j}}{\lambda + c\mu - c\mu b_{j+1}^{(c)}},$$

$$b_j^{(c)} = \frac{\lambda}{\lambda + c\mu - c\mu b_{j+1}^{(c)}},$$

$j = K-1, K-2, \ldots, c+1$ *and*

$$a_K^{(c)} = \frac{\alpha\pi_{c-1,K}}{c\mu}, \qquad b_K^{(c)} = \frac{\lambda}{c\mu}$$

From equations (2), (3), (4), (5) and (6), all the probabilities $\pi_{i,j}$ can be expressed in term of $\pi_{0,0}$. With the normalizing condition $\sum \pi_{i,j} = 1$, all $\pi_{i,j}$ can be calculated. According to the [27], the computational complexity order for $\{\pi_{i,j}; i \le K, j \le K\}$ is $O(cK)$.

Lemma 2 *If $X_1, X_2, \ldots, X_n$ are independent exponential random variables with expectations ${\mu_1}^{-1}$, ${\mu_2}^{-1}$, ..., ${\mu_n}^{-1}$ then:*

$$P\{X_i = \min(X_1, X_2, \ldots, X_n)\} = \frac{\mu_i}{\mu_1 + \mu_2 + ... + \mu_n}$$

Proof Let $y = min(X_1, X_2, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$, y has the exponential distribution with expectation $(\mu_1 + \mu_2 + \ldots + \mu_{i-1} + \mu_{i+1} \ldots + \mu_n)^{-1}$. We have:

$$P\{X_i = \min(X_1, X_2, \ldots, X_n)\} = P(X_i < y)$$

If A, B are independent exponential random variables with expectations ${\mu_A}^{-1}$, ${\mu_B}^{-1}$ then $P(A < B) = \frac{\mu_A}{\mu_A + \mu_B}$. Hence,

$$P\{X_i = \min(X_1, X_2, \ldots, X_n)\} = P(X_i < y) = \frac{\mu_i}{\mu_1 + \mu_2 + \ldots + \mu_n}$$

Lemma 3 *In ON/OFF/MIDDLE/c/K Markov chain model, if at any time t, the system is in state $(\theta, i), (i > \theta)$ then the probability that the system is in the state (θ', j) at time $(t + h)$ $(\theta' \neq \theta$ or $i \neq j)$ is:*

$$P\{(\theta, \mathrm{i}) \to (\theta', \mathrm{j})\} = \begin{cases} \min(i-\theta, c-\theta)\alpha h + o(\mathrm{h}), & \theta' = \theta + 1 \text{ and } i = j \\ \lambda h + o(h), & \theta' = \theta \text{ and } j = i + 1 \\ \theta\mu h + o(\mathrm{h}), & \theta' = \theta \text{ and } j = i - 1 \\ o(\mathrm{h}), & \textit{other cases} \end{cases} \quad (7)$$

Proof If $i > \theta$, job queue is not empty. There are three types of events that can happen during $(t, t+h)$, namely a job arrives, a job is completed, and a server is switched to ON state (via SETUP mode). Jobs arrive according to a Poisson process with rate λ. Their inter-arrival times (the time period between two consecutive jobs), execution time and SETUP time are independent and conform to an exponentially distributed random variables with expectations λ^{-1}, μ^{-1}, and α^{-1}, respectively. On the other hand, the time to the next event is the minimum of the following three random variables: inter-arrival time, the θ execution time of θ running servers and the $\min(c-\theta, i-\theta)$ SETUP time of $\min(c-\theta, i-\theta)$ servers which are switching to ON state. Thus, the time from t until the next event has the exponential distribution with expectations $(\lambda + \min(c-\theta, i-\theta)\alpha + \theta\mu)^{-1}$. Thus, the probability of an event will occur in $(t, t+h)$ is:

$$1 - e^{-((\lambda+\min(c-\theta,i-\theta)\alpha+\theta\mu)h} = (\lambda + \min(c-\theta, i-\theta)\alpha + \theta\mu)\, h + \mathrm{o}\,(h)$$

When an event occurs, according to the Lemma 2, the probability will be impacted by the arrival of a job: $\lambda/(\lambda + min(c-\theta, i-\theta)\alpha + \theta\mu)$. The time scope needed for the event to occur and the event type are independent. Therefore, the probability of an occurrence of the transition $(\theta, i) \to (\theta, i+1)$ is:

$$\begin{aligned}\mathrm{P}\,\{(\theta,\mathrm{i}) \to\ (\theta,\mathrm{i}+1)\} = \frac{\lambda}{\lambda + \min(c-\theta, i-\theta)\alpha + \theta\mu} * \\ [(\lambda + \min(c-\theta, i-\theta)\alpha + \theta\mu)\,\mathrm{h} + \mathrm{o}\,(\mathrm{h})] + \mathrm{o}\,(\mathrm{h}) \\ = \lambda\mathrm{h} + \mathrm{o}\,(\mathrm{h})\end{aligned}$$

By a similar argument,

$$\begin{aligned}\mathrm{P}\,\{(\theta,\mathrm{i}) \to\ (\theta,\mathrm{i}-1)\} = \frac{\theta\mu}{\lambda + \min(c-\theta, i-\theta)\alpha + \theta\mu} * \\ [(\lambda + \min(c-\theta, i-\theta)\alpha + \theta\mu)\,\mathrm{h} + \mathrm{o}\,(\mathrm{h})] + \mathrm{o}\,(\mathrm{h}) \\ = \theta\mu\mathrm{h} + \mathrm{o}\,(\mathrm{h})\end{aligned}$$

And:

$$\begin{aligned}\mathrm{P}\,\{(\theta,\mathrm{i}) \to\ (\theta+1,\mathrm{i})\} = \frac{\min(c-\theta, i-\theta)\alpha}{\lambda + \min(c-\theta, i-\theta)\alpha + \theta\mu} * \\ [(\lambda + \min(c-\theta, i-\theta)\alpha + \theta\mu)\,\mathrm{h} + \mathrm{o}\,(\mathrm{h})] + \mathrm{o}\,(\mathrm{h}) \\ = \min(c-\theta, i-\theta)\alpha\mathrm{h} + \mathrm{o}\,(\mathrm{h})\end{aligned}$$

The probability of more than one event will occur during the period $(t, t+h)$ is $o(h)$ with $h \to 0$, it conforms to the formula $P\{(\theta,\mathrm{i}) \to (\theta',\mathrm{j})\} = o(h)$ even in the others cases.

Theorem 1 *In ON/OFF/MIDDLE/c/K model, the average number of servers that are switched from MIDDLE to ON successfully in time period t is given by:*

$$E\,[M] \approx \alpha t \sum_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\min\,(c-i, \mathrm{j}-\mathrm{i}) \tag{8}$$

The computational complexity order for $E[M]$ is $O(cK)$.

Proof Assume K_h is the number of servers that are switched to ON state in time period h. Thus, $K_h > 0$ when the transition $(\theta, i) \to (\theta', j)$ $(\theta' > \theta)$ occurs. According to Lemma 3, $P\{(\theta, i) \to (\theta', j)\} = o(h)$ if $\theta' - \theta > 1$, hence $P(K_h > 1) = o(h)$. $K_h = 1$ when the system switches from (θ, i) to $(\theta+1, i)$ $(i > \theta)$. Therefore, according to the law of total probability:

$$\begin{aligned}P\,(K_h = 1) &= \sum_{j>\mathrm{i}} \pi_{\mathrm{i},j}.P\,\{(\mathrm{i},\mathrm{j}) \to\ (\mathrm{i}+1,\mathrm{j})\} \\ &= \sum_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\,(min\,(c-i, j-i)\,.\alpha h + o\,(h)) \\ &\approx \alpha h \sum_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\min\,(c-i, \mathrm{j}-\mathrm{i})\end{aligned}$$

Then we have $E\,[K_h] \approx \alpha h \sum\limits_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\min\,(c-i, \mathrm{j}-\mathrm{i})$

Dividing t into small enough time period h, assume $K_h^{(z)}$ is the number of servers switched to ON state in z-th time period. We have:

$E\left[K_h^{(z)}\right] = \alpha h \sum\limits_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\min\,(c-i, \mathrm{j}-\mathrm{i})$,

$M = \sum\limits_z K_h^{(z)}$.

Thus

$$\begin{aligned}E\,[M] = E\left[\sum_z K_h^{(z)}\right] &= \sum_z E\left[K_h^{(z)}\right] \\ &\approx \alpha h \sum_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\min\,(c-i, \mathrm{j}-\mathrm{i})\,.\frac{t}{h} \\ &= \alpha t \sum_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\min\,(c-i, \mathrm{j}-\mathrm{i})\end{aligned}$$

According to Lemma 1, all the limit probabilities $\pi_{i,j}$ are expressed in term of $\pi_{0,0}$ and could be calculated by using a simple recursion with computational complexity $O(cK)$. Therefore, according to the formula (8), $O(cK)$ is also the computational complexity for $E[M]$.

3.2.2 Control Algorithm

The Theorem 1 gives the formula for the expectation of the server number that switch from MIDDLE to ON state successfully. The requirement for our strategy is that the number of servers switched from OFF to MIDDLE state must equal the expectation of the server number switched from MIDDLE to ON state in any time period.

Algorithm 1 describes the switching strategy for servers from OFF to MIDDLE state gradually. The algorithm is called *MIDDLE Server Switching Strategy* (abbreviated as MSSS), in which one OFF server is powered to MIDDLE state every time period τ. The change of parameter τ leads to changing the number of servers switched to MIDDLE state in total. Consequently, the parameter τ is calculated again and again to satisfy the above requirement. With the expectation of the server number switched to ON state successfully given by formula (8), we get time period τ as follows:

$$\tau = \frac{t}{E\,[M]} = \frac{1}{\alpha \sum_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\min\,(c-i, \mathrm{j}-\mathrm{i})} \tag{9}$$

where $E[M]$ and t are given in Theorem 1.

According to the Lemma 1, since the computational complexity order for calculating all $\pi_{i,j}$ $(i \le c, j \le K)$ is $O(cK)$, the computational complexity order for τ is also $O(cK)$.

Algorithm 1 MIDDLE Server Switch Strategy (MSSS)

Input: λ, μ, α
Output: τ, time to power on an OFF server

```
while true do
    switch one server from OFF to MIDDLE
    calculate time τ by formula (9)
    wait time period τ
end while
```

With MSSS algorithm, the turn OFF to ON process of servers is divided into two states, each stage is managed in a different manner. In the first stage power (power-on stage), OFF servers are periodically switched to MIDDLE state every time period τ by MSSS algorithm. The second stage is managed by ON/OFF/MIDDLE policy presented in the Subsection 3.1 that MIDDLE servers are turned into ON according to the number of jobs waiting in queue. In ON/OFF model presented in [5], [26], and [27], the turning process is just managed by the ON/OFF policy. This policy makes many servers to be turned on unsuccessfully. The reason is that OFF servers are switched to ON state according to the number of jobs in queue but it is usually not stable. So, if the job number in queue decreases, some servers in turning process are turned off. In contrast, our strategy combines MSSS algorithm and ON/OFF/MIDDLE policy to take both the advantages of them, and manages servers in a better way. The MSSS algorithm guarantees that just a sufficient number of servers are switched from OFF to MIDDLE while ON/OFF/MIDDLE policy ensures that some MIDDLE servers continue to be switched to ON state according the number of jobs in queue.

4 Elastic queue management mechanisms

Data centers provide physical resource base for all operations of cloud services and applications. To take a full advantage of investment costs and to ensure the service feasibility for as many appliances and users as possible, cloud vendors often build very big data centers. They exploit thoroughly the virtualization technology to accommodate resources elastically and quickly according to actual requirements of consumers. With a large number of servers running inside, the cloud data centers have to serve the huge number of computational jobs with different arrival intensities. As a result, if the center is managed by only a single job queue, the capacity of queue must be very large. However, then the service waiting time of the system will be longer. Therefore, to ensure the QoS, we propose multiple queues model with elastic management mechanisms. In this model, the large data center is managed by a lot of finite-capacity queues, which are operated simultaneously. This helps the systems to perform jobs efficiently.

In this section, based on the server farm strategy management with a single finite-capacity queue developed in Section 3, we bring forward a generic architecture to serve very large quantity of jobs in data centers. In this system, the arriving job rate is regarded as very high and changes continuously in a certain period time. To execute the arriving jobs, the number of physical servers is also considered to be very large. Indeed, data centers of Google, Microsoft, Yahoo and Amazon, etc. have tens of thousands of servers and the number is increasing annually [13]. However, there are still two questions, which should be addressed in this context:

- If one job queue for the system is not suitable, so how many queues is enough for the large number of arriving jobs?
- Otherwise, how to manage the queues flexibly according to job quantity to reach the aim of energy saving and a quick service?

Dealing with those problems we have proposed an architecture with multiple job queues for data centers. The architecture is illustrated in the Fig. 4. Each data center is divided into many subsystems called server clusters, in which each cluster has a finite number of machines and uses one job queue. The switching process of servers is managed by our MSSS strategy developed in the previous section. In other words, each cluster is a single finite-capacity queue with servers defined by three states OFF, MIDDLE and ON. For better control, the arriving jobs are buffered in a scheduling server before transferring to be executed in an available cluster. In the architecture, the scheduling server plays the role of the decision making which cluster will serve new incoming job. In the case that no clusters are available, the server will buffer jobs, and thus we avoid the phenomenon of refusing service for jobs.

Definition 5 A data center is composed of many server clusters and is managed by a scheduling management server containing a job buffer. Jobs are then allocated elastically to clusters based on arrival rates and cluster availability. The buffer is used for storing jobs in case that there are no available clusters to serve them.

Definition 6 Each cluster is a single finite-capacity queue with servers defined by three states, namely OFF, MIDDLE and ON. The total number of servers in entire datacenter is calculated by the following formula:

$$n_{server} = \sum_i n_{OFF}^{(i)} + \sum_i n_{MIDDLE}^{(i)} + \sum_i n_{ON}^{(i)} \qquad (10)$$

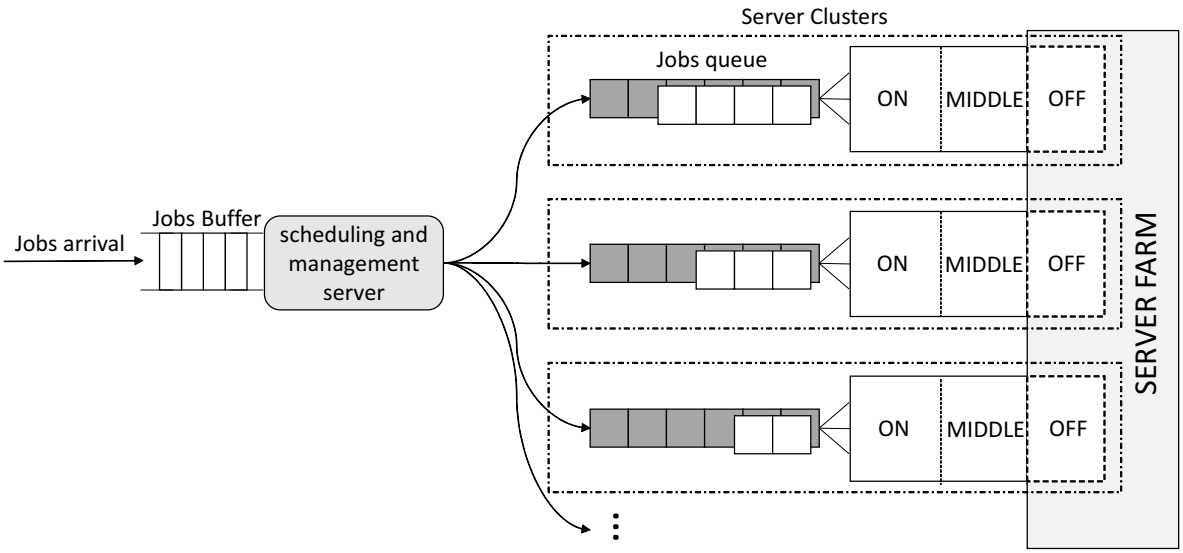

Fig. 4 Model of multi-cluster system

where $n_{OFF}^{(i)}$, $n_{MIDDLE}^{(i)}$, $n_{ON}^{(i)}$ are the number of OFF, MIDDLE and ON servers in the i-th cluster, n_{server} is number of servers in entire datacenter.

We assume that a cluster has c servers. Each of servers can execute at most one job at any point of time. It means that each server cluster can run at most c jobs at the same time. Therefore, if there are $(c + \delta)$ jobs arriving into a cluster, firstly, c jobs will wait for c servers to switch to ON from OFF or MIDDLE states, and the remaining δ jobs must wait more for the ON servers until they serve completely several jobs. To reduce service waiting time in this case, these δ jobs will be transferred to another cluster. We set the cluster's capacity K equals c and hence at most c jobs can stay in each cluster. In our model, the number of clusters is changed according to the number of jobs coming into the whole system. This process is realized through initialization or deactivation operations of a cluster. In addition, we also defined thresholds to determine the moment, when the number of clusters is increased or decreased automatically.

Specifically, there are four mechanisms that are designed for cluster management and jobs scheduling as follows:

- *Cluster states management* is a mechanism to manage clusters according to three states involving *killed*, *waiting* and *available*. The three states and procedures for switching clusters among these states are determined.
- *Jobs scheduling* is a strategy to allocate jobs to the best cluster or to buffer jobs if all clusters in the whole system are fully occupied. The cluster choice process relies on the information about the job quantity inside clusters.
- *Thresholds* are policies predefined to decide when to change the number of clusters based on initialization or killing operations. There are two threshold types: low and high thresholds.
- *Waiting server policy*: if a cluster is initialized, ω servers will be switched from OFF to MIDDLE state simultaneously. The ω is also the number of servers that are kept in MIDDLE when a cluster is deactivated to *waiting* state. This policy offers a formula to reckon ω in order to optimize the power consumption of servers in the MIDDLE state.

In the next subsections, these four mechanisms for cluster management and jobs scheduling will be presented in more details.

4.1 Cluster state management

As proposed above, in our architecture, data centers are able to increase or decrease elastically the cluster quantity to ensure that all jobs will be served as quickly as possible, and to take all advantage of data center resources as well. The goal of the cluster state management is to construct and destruct clusters on demand. In the similar manner of server control, we also define states for the clusters and model transition processes. Fig. 5 depicts those cluster states and their transitions. There are three states for the clusters, including:

- *Killed* is a cluster state, in which all servers are turned off, and the cluster cannot receive any jobs.

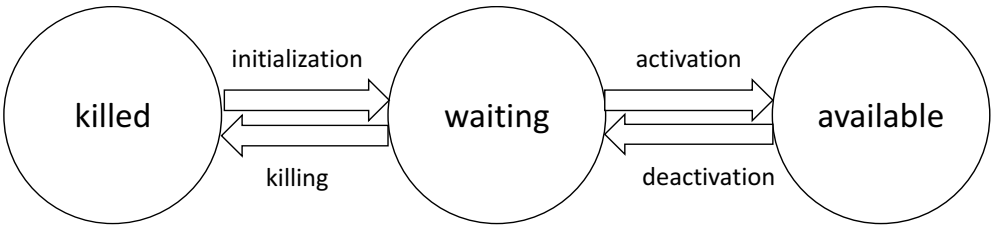

Fig. 5 State transition of cluster

- *Waiting* is a state of clusters in which clusters do not contain any jobs (not even in the queue) but still keep ω servers in MIDDLE state to wait for arriving jobs. In this state, the switching process from OFF to MIDDLE is temporarily disabled.
- *Available* is a state defined for clusters if they contain jobs. The jobs may be performed inside ON servers or waiting in the queue. In this state, the switching process from OFF to MIDDLE is enabled and controlled by Algorithm 1.

Meanwhile, the transitions among cluster states are based on four actions as follows:

- *Initialization*: this is the cluster initialization process from *Killed* to *Waiting* state. Firstly, a specific number of OFF servers is powered in order to become MIDDLE servers. Along with that, a job queue with finite capacity K is created and configured to receive arriving jobs. Costs required to create a job queue is regarded as insignificant and the creating time is set to zero.
- *Activation*: is an action that occurs when new jobs arrive at a *Waiting* cluster. At this time, the system uses Algorithm 1 to control the quantity of MIDDLE servers switched to ON state.
- *Deactivation*: an *Available* cluster is deactivated if its job queue is empty. To deactivate a cluster, OFF-to-MIDDLE switching process is disabled, however the system still keeps ω servers in MIDDLE state. If the number of MIDDLE servers is greater than ω, the remaining servers will be turned off immediately. Conversely, few OFF servers are continuously powered to MIDDLE state until reaching ω.
- *Killing*: this action turns off all servers in the cluster. After that the cluster cannot receive any jobs. The required time for *Killing* action is considered as zero.

Note that the number of servers is denoted by ω and formulated in the Section 4.4 below. These four transition actions for clusters are expressed by algorithms 2, 3, 4 and 5.

Algorithm 2 Cluster initialization

1: Calculate ω by formula (12)
2: Switch ω OFF servers to MIDDLE
3: Create jobs queue
4: Configure jobs queue

Algorithm 3 Cluster activation

1: z = number of jobs in queue
2: t = number of MIDDLE server
3: Switch $min(z, t)$ MIDDLE servers to ON state
4: Enable MSSS

Algorithm 4 Cluster deactivation

1: Disable MSSS
2: t = number of MIDDLE server
3: **if** $t > \omega$ **then**
4: turn off $(t - \omega)$ MIDDLE servers
5: **else**
6: switch $(\omega - t)$ OFF server to MIDDLE state
7: **end if**

Algorithm 5 Cluster killing

1: Remove jobs queue
2: Turn off all servers

4.2 Job scheduling

In multiple queues model, each cluster has the limitation of capacity K. This means the cluster cannot receive new jobs when it is processing K jobs at a certain moment. Job scheduling is a mechanism that enables to control job flow arriving to diverse clusters. The mechanism is set up on the scheduling management server to gather the information about job quantity inside all clusters in *Available* and *Waiting* states. Relying on the information, the scheduling mechanism will choose the best cluster to transfer jobs to perform. If there are no available clusters in two states above, the scheduling server will put arriving jobs in its buffer to avoid service denial phenomenon. To control cluster choice process, a policy called *First Available Cluster* (abbreviated as FAS) is used to transfer jobs to the first cluster that is full of jobs. Fig. 6 shows the FAS control policy. The arriving jobs will come to Cluster 1, if it still has the ability to perform more jobs. In the case that Cluster 1

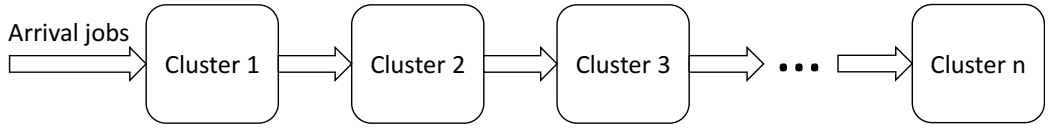

Fig. 6 First available cluster policy (FAS)

is full of jobs, the flow of arriving jobs will be directed to Cluster 2. This process will continue until all clusters can't receive any more jobs. The operation steps of the FAS policy are described in algorithm 6.

Algorithm 6 First Available Cluster (FAS) policy

```
for i = 1 to n do
    if cluster is in waiting or available states and number of jobs in i-th cluster < K then
        best cluster := cluster i; break;
    end if
end for
if best cluster is exist then
    transfer new jobs to the best cluster
else
    buffer new jobs in scheduling server
end if
```

4.3 Thresholds

Because the number of clusters is changed to maximize job service capability the server farm system frequently creates and destructs clusters. This leads to the rise of the long waiting time issue in the cluster initialization process. In this way, the service time increases and impacts the system QoS concerning the fastness of jobs serving. To resolve this issue, a job threshold is used to determine when the system has to start the process of creating a new cluster before all *Available* clusters contain maximum K jobs. Relying on the threshold, the waiting time for the cluster creation is decreased. In addition, we use another threshold to terminate clusters if the system does not need it anymore.

In short, there are two kinds of thresholds: upper (*thrUp*) and lower threshold (*thrDown*). Due to the usage of FAS policy for job scheduling, in all cases, the latest cluster will be chosen to receive new jobs while the previous clusters contain K jobs at maximum. Thus, the upper threshold is applied only to the n-th cluster, where n is the number of *Waiting* and *Available* clusters. The system will decide to initialize a new cluster if the number of jobs in the n-th cluster reaches *thrUp*.

Otherwise, in order to save energy consumption, we use lower threshold to destruct unnecessary no-load clusters. More specifically, if the number of jobs buffered in the scheduling server is zero and the n-th cluster also contains no jobs (the cluster is in *Waiting* state or in *initialization* process), the cluster will be destructed as soon as the number of jobs in $(n-1)$th cluster is below *thrDown*. The increase and decrease cluster algorithms using thresholds are described in algorithms 7 and 8. In these algorithms, n and K are used to denote the number of queues and maximal capacity of a cluster, respectively.

Note that the increase or decrease number of clusters is only one in both algorithms 7 and 8 mentioned above. However, there is also the requirement for managing effectively the *initialization* process in case if all *Available* clusters contain K jobs but a great number of new jobs still continues to arrive at the system. At the time, these new jobs are stored in the buffer and probably one new created cluster is not enough to process all jobs quickly. Therefore, the management mechanism must enable the initialization of a dynamic number of clusters based on the arriving job rate. In other words, if the number of jobs in the buffer increases constantly, not only one cluster is put into the *initialization* process, but probably many clusters are initialized simultaneously. The cluster quantity is calculated by the following formula:

$$n_{initializing\ servers} = \lceil \frac{n_{jobs\ in\ buffer}}{K} \rceil \tag{11}$$

The algorithm 9 provides a mechanism to control the quantity of clusters that is initialized immediately.

Algorithm 7 Cluster increase

```
if all of first (n − 1) clusters has K jobs then
    if number of jobs in n-th cluster = thrsUp − 1 and a new job arrives at system then
        Initialize one new cluster by Algorithm 2
    end if
end if
```

4.4 Waiting cluster policy

As designed above, in *Waiting* state, clusters maintain ω MIDDLE servers available. Therefore, the number of these servers must be optimized to reduce the power consumption. In addition, to switch OFF server to MIDDLE state, it takes a period time $t_{OFF \to MIDDLE}$. When

Algorithm 8 Cluster decrease

```
if buffer is empty then
    if n-th cluster has no jobs then
        if number of jobs in (n-1)th cluster decreases to thrDown then
            Kill n-th cluster by Algorithm 5
        end if
    end if
end if
```

Algorithm 9 Control of cluster quantity in initialization process

```
Calculate n_initializing servers by formula (11)
if buffer size increases to K * n_initializing servers then
    Initialize one new cluster by Algorithm 2
end if
if buffer size decreases to (K − 1) * n_initializingservers then
    Kill the latest initializing cluster by Algorithm 5
end if
```

a *Waiting* cluster is activated to *Available* state, due to needed time $t_{OFF \to MIDDLE}$, there is not yet OFF server, which is switched to MIDDLE state successfully. Thus, the number of MIDDLE servers in the time period $t_{OFF \to MIDDLE}$ equals: $n_{MIDDLE\ servers} = \omega - n_1$, where n_1 is the number of server in MIDDLE switched to ON state in the time period $t_{OFF \to MIDDLE}$.

To ensure that MIDDLE servers in those clusters are always available, the following condition must be satisfied: $\omega \geq n_1$. This condition means that ω must at least equal the number of servers switched from MIDDLE to ON state in time period $t_{OFF \to MIDDLE}$. From formula (8) and (9) we get minimal ω:

$$\omega = \alpha t_{OFF \to MIDDLE} \sum_{j>\mathrm{i}} \pi_{\mathrm{i},j}.\,(\mathrm{j} - \mathrm{i}) = \frac{t_{OFF \to MIDDLE}}{\tau} \tag{12}$$

5 Experiments and evaluations

In this section, we present experiments as well as evaluations for the single queue model with MSSS, and multiple queues model with the elastic management algorithms. These experiments are divided into three groups as follows:

1. For a single queue model: in order to prove the operations of MSSS in a single queue model, we carry out tests to determine the time period τ and the number of servers in MIDDLE state. We also compare the mean waiting time of arriving jobs between three-state and traditional two-state model to show the effectiveness of MSSS in the aspect of QoS.
2. For a multiple queues model: in order to show the operability of our management mechanisms and the effect of the capacity and thresholds in a multiple queues system, we measure waiting time of arriving jobs and the mean number of active clusters.
3. Power consumption: in order to test the power impact, we measure energy consumption of our proposed strategies in both the single queue and the multiple queues model.

Together with those experimental results, we also bring forward observations to evaluate our solutions.

5.1 Experimental setup

We use the CloudSim [4] to simulate our models in cloud environment with data centers, physical servers, computational jobs and queues. The simulation is set up with a server farm with 10000 hosts, where each host is configured with 16 CPU cores (1200 MIPS/core), 32GB of RAM, and 1TB of storage. We consider an arriving job as the process of deploying a VM into the host farm. In all experiments, we fix $\mu = 0.2$ and $t_{OFF \to MIDDLE} = 400$ seconds.

5.2 Single queue system with finite capacity

For the first group tests, we simulate only one server cluster, which has a single job queue and the finite capacity. The cluster has the following features $c = 100$, $\alpha = 0.002$, $K = 100$, 200 and 500 respectively. These tests aim to prove that our control algorithms work well and efficiently in the comparison with ON/OFF model. In particular, we measure the following four parameters:

- (S1) time period τ and (S2) the number of SETUP and MIDDLE servers to show the effect of Algorithm 1 for server switch management process.
- (S3) the mean waiting time and (S4) block proportion to demonstrate that our strategy works effectively and reduces the waiting time and block ratio of arriving jobs. In this way, the approach increases the QoS for our system.

Dealing with direction, Fig. 7 shows the time period τ for (S1), which is calculated by our Algorithm 1. We observe that in all cases of α and K, the time period τ decreases/increases depending on λ. The reason is that the continuous change of the arriving job rate leads to a change of the number of servers needed to be powered from OFF state. In other words, if λ increases, more jobs arrive at the system, then there is a need of turning on more servers, so τ will decrease. However, in the case

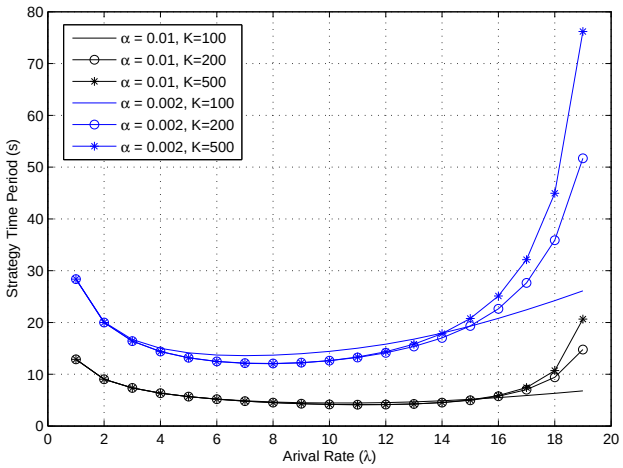

Fig. 7 Time period τ

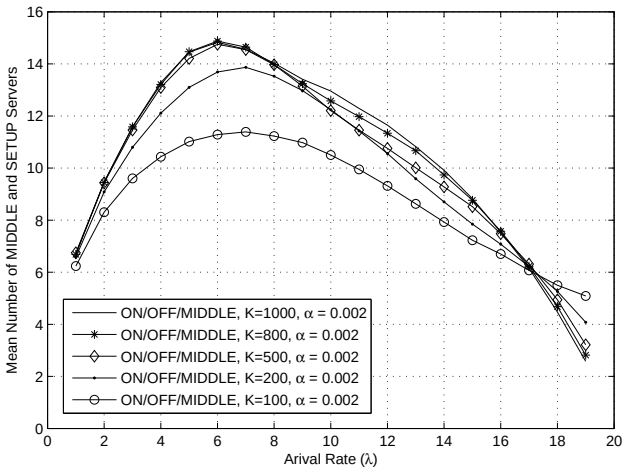

Fig. 8 Mean number of SETUP and MIDDLE servers in three-state model with single finite-capacity queue

of λ is very large, almost all servers in the system are already in ON state and the number of OFF servers remains quite small, then the system cannot turn on many OFF servers, and hence τ will increase.

Fig. 8 illustrates the mean number of SETUP and MIDDLE servers against the arrival rate in three-state model for (S2). The achieved results are like the time period τ test above, in which the quantity of SETUP and MIDDLE servers increases and decreases in the proportion with λ. In Fig. 8, while λ is in the range (1, 6), the arriving jobs number augments. This leads to phenomenon of more MIDDLE servers are put into SETUP mode to switch to ON state. Conversely, in the rest of λ range, when λ increases to 19, the traffic intensity of system $\rho = \frac{\lambda}{c\mu}$ tends to 1, then the system reaches the overload point, most of servers are in ON state. Consequently, the number of servers in SETUP and MIDDLE is less if λ is larger.

Next, we measure and compare the mean job waiting time between three-state and ON-OFF model with a finite capacity queue for (S3). The outcomes of this experiment are presented in Fig. 9. The waiting time is calculated by formula:

$$t_{waiting} = \frac{\sum (t_{execute} - t_{create})}{n_{jobs}}$$

where $t_{execute}$ is the moment when the system starts to serve a job, and t_{create} is moment when the job arrives at the system, n_{jobs} is the total quantity of jobs that arrive at the system.

As can be seen from Fig. 9, if λ is in range $(1, 8)$, the mean waiting time of ON/OFF/MIDDLE is significantly less than ON/OFF model about from 15% to 30%.

Otherwise, if λ increases from 9 to 19, the job waiting time of the three-state is slightly greater than the two-state model. The reason is that the arrival job rate λ is quite high and at this time almost all servers are in ON state and the number of SETUP and MIDDLE servers is very small. In this case, it is obvious that the effect of MSSS algorithm is insignificant. In addition, as proposed in Section 3.1, since the server switch process is divided into two stages, the algorithm MSSS will cause the increase of waiting time. However, this increase is not too large as the results shown in Fig. 9. For this reason, we can conclude that if the arrival rate is very high, then the two models are the same.

Finally, for the single queue model, we carry out the experiment to measure the job block proportion of the whole system (S4). The achieved results are presented in Fig. 10. They show the block proportion is depended on the queue capacity. Through the measurement, we can make an important observation that with the three-state model, the job blocks proportion decreases significantly in range $(1, 9)$ of λ. If λ is in range $(9, 19)$, the block proportion of ON/OFF/MIDDLE is greater than ON/OFF model. The phenomenon also occurred like the mean waiting time experiment and it also can be explained in a similar way.

In summary, with four experiments presented above, we proved that our three-state model for servers and MSSS can work well with a single finite capacity queue system. In addition, we also concluded that if the arrival rate is small, our three-state model not only decreases the mean waiting time of jobs but also reduces the block proportion. From the view of users, our proposals thus improve QoS. However, if the arrival rate is very large, the traffic intensity ρ tends to 1, the system reaches the overload point, then the effect of the MSSS algorithm is insignificant and there are almost no differences between the two models. This is consistent with the situation in small data centers operations.

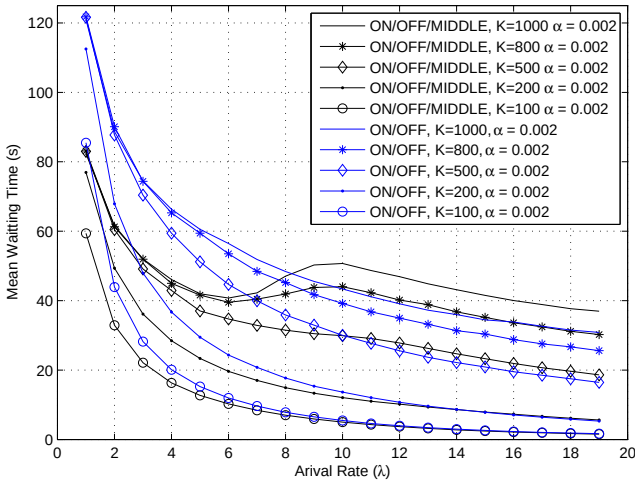

Fig. 9 Mean waiting time of jobs in three-state and two-state model with single finite-capacity queue

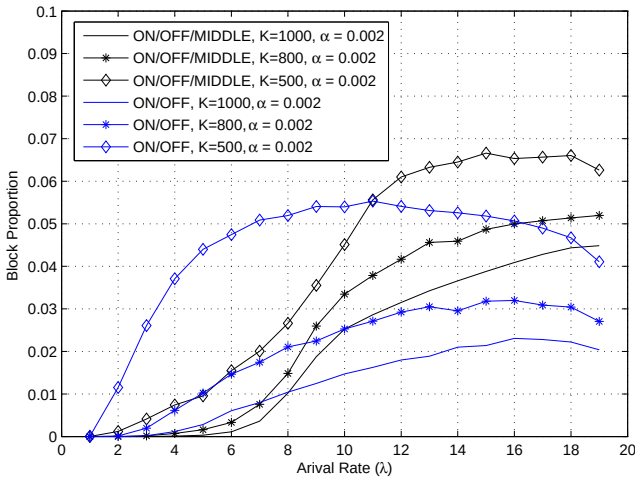

Fig. 10 Block ratio of jobs in three-state and two-state model with single finite-capacity queue

5.3 Multiple queues system

In the second group test, we simulate a very large data center with multiple finite-capacity server queues using the management mechanisms introduced in Section 4. We consider $\alpha = 0.002$ and $c = K = 50, 100, 200, 500$ respectively. The experiments in this group are carried out to demonstrate manageability of our mechanisms with multiple queues, in which the system can change automatically and elastically the number of clusters (note that each cluster is a finite-capacity queue, according to Definition 6) based on the diverse arrival job rate. Particularly, for this group tests, we measure the following three parameters:

- (M1) the average number of *Available* clusters with the different arrival job rate to affirm the operability of our mechanism in complex system.
- (M2) the job waiting time in system with different values of thresholds to show the impact of thresholds in the environment of multiple finite-capacity server queues.
- (M3) the mean waiting time of the job against the arrival job rate λ in the different cases of c and K to prove the effectiveness of the model.

For (M1), the average number of *Available* clusters is shown in Fig. 11. Some observations can be made from the outcomes: in all cases of c and K values, the *Available* cluster number increases according to λ. We also recognize that the mean number is larger when K is less. The reason is that if K is small, the system must create more clusters to ensure that there are enough servers for processing new arriving jobs. These results proved our strategy for the elastic queues management work well and all clusters are managed elastically.

We evaluate impact of queue thresholds inside the system by calculating the job waiting time while changing the values of upper thresholds from 40 to 80 percent for (M2). The results are depicted in Fig. 12. Through measurement process, we remark that the mean waiting time of jobs in the system is small if upper threshold is decreased. The phenomenon can be explained as follows, if the upper threshold value is kept smaller, as a consequence, new clusters will be initialized earlier. Therefore, the mean waiting time for turning ON servers also will be reduced.

For (M3), we change the values of c and K. Fig. 13 illustrates the mean waiting time of jobs against the arriving job rate λ. The outcomes show that the waiting time decreases while λ increases. In contrast to the single queue model, there is no job, which is blocked because the system with multiple queues is able to expand the service capability based on adjusting the number of clusters. In comparison between the cases of c and K, we observe that when arrival rate λ is in the range $(1, 6)$, the system with the bigger cluster capacity has the longer waiting time, and otherwise, if the arrival rate λ is in the range $(11, 19)$ the system with the smaller cluster capacity has the longer waiting time. The experiment suggests that the system with the bigger capacity of clusters will do better in the case if the arrival jobs rate is high. Nevertheless, it might cause the rise of the power consumption. This is reason why we continue to perform the energy evaluations presented in the Subsection 5.4 described below.

Through the experiments in the second group, we demonstrated the elastic manageability of proposed mechanisms for a large server farm, which uses multiple

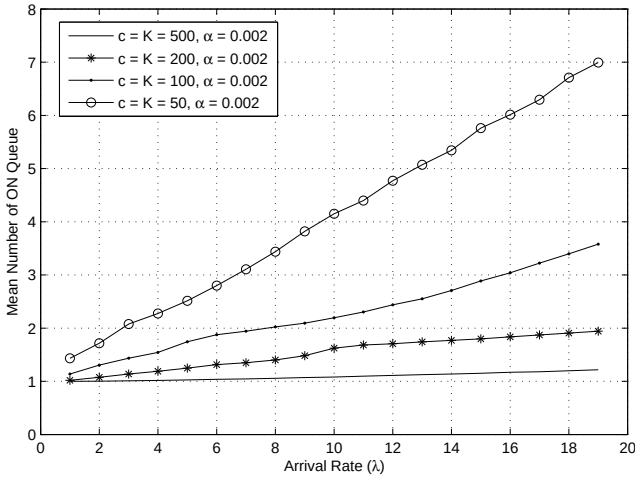

Fig. 11 Mean number of *Available* cluster

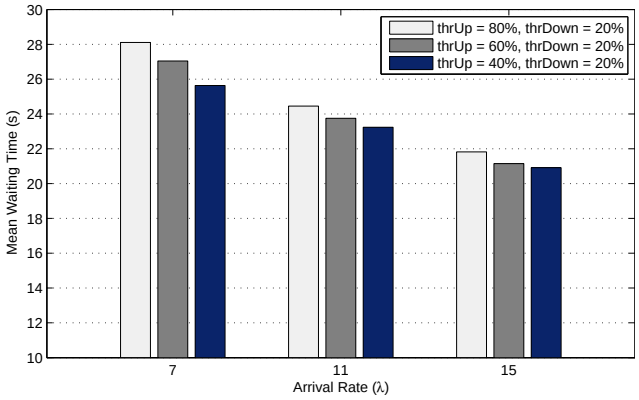

Fig. 12 Effect of threshold to waiting time ($\mu = 0.2, \alpha = 0.002$)

finite-capacity queues at the same time. The architecture enables data centers to serve quickly and effectively a huge number of arrival computational jobs. In this way, it enhances serviceability, decreases the service waiting time for a huge number of jobs for the whole system from the point of view of both vendors and users. The experiments also expressed the effect of the queue capacity and thresholds on the waiting time in the multiple queues model.

5.4 Power consumption

In the last experiment group, we evaluate the power consumption in ON/OFF/MIDDLE model with finite capacity as comparison with ON/OFF model. We assume that the consumption of servers in power-on states (ON, MIDDLE) and switching processes (SETUP, from OFF to MIDDLE) are invariable. We obtain the mean

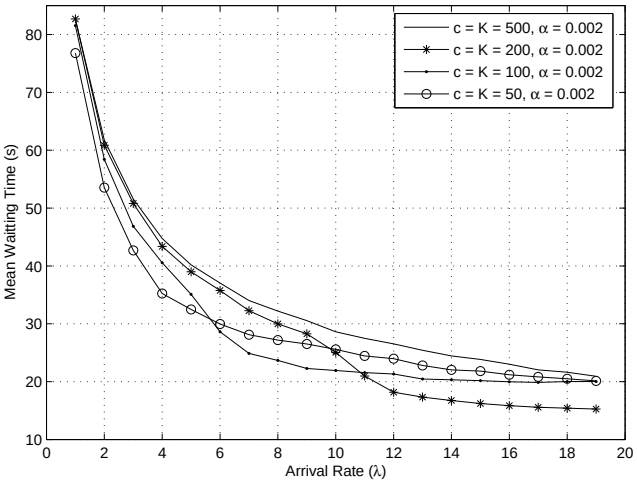

Fig. 13 Mean waiting time in multiple queues system

power consumption of the whole system based on several following formulas.

In three - state model, the mean power consumption is given by:

$$\begin{aligned} P_{three-statemodel} = {} & P_{ON}.E[n_{ON}] + \\ & P_{MIDDLE}.E[n_{MIDDLE}] + \\ & P_{SETUP}.E[n_{SETUP}] + \\ & P_{OFF\rightarrow MIDDLE}.E[n_{OFF\rightarrow MIDDLE}] \end{aligned}$$

where n_{ON}, n_{MIDDLE}, n_{SETUP} are the number of ON servers, MIDDLE servers, SETUP servers, respectively. $n_{OFF\rightarrow MIDDLE}$ is the number of servers that are in turning process from OFF to MIDDLE state. P_{ON}, P_{MIDDLE}, P_{SETUP} are the power consumption of a server in ON, MIDDLE, SETUP state respectively. $P_{OFF\rightarrow MIDDLE}$ is power consumption of a server in turning process from OFF state to MIDDLE state.

Similarly, the mean power consumption of ON-OFF model is reckoned by:

$$\begin{aligned} P_{\text{two}-\text{statemodel}} = {} & P_{ON}.E[n_{ON}] + \\ & P_{OFF\rightarrow ON}.E[n_{OFF\rightarrow ON}] \end{aligned}$$

where $n_{OFF\rightarrow ON}$ is the number of servers that are in turning process from OFF to ON state. $P_{OFF\rightarrow ON}$ is the power consumption of a server in the turning process from OFF to ON state.

The power consumption of the switching process from OFF to ON state is given by:

$$P_{OFF\rightarrow ON} = \frac{P_{OFF\rightarrow MIDDLE} * t_{OFF\rightarrow MIDDLE} + P_{SETUP} * t_{SETUP}}{t_{OFF\rightarrow ON}}$$

According to the IBM analysis [1], we assume the power consumption of a server in ON state is $P_{ON} =$

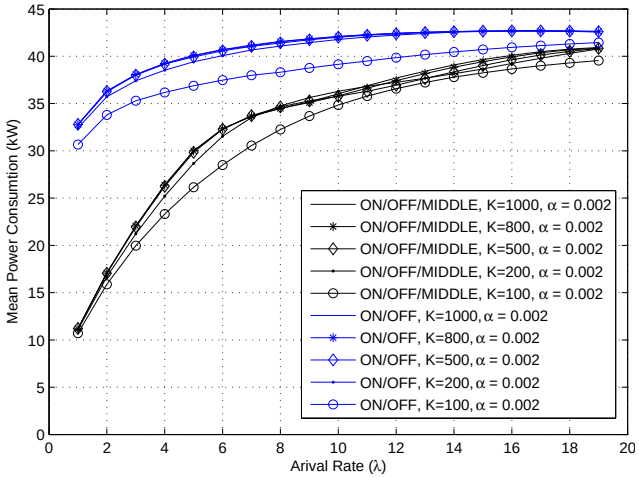

Fig. 14 Power consumption in three-state model and two-state model with single queue

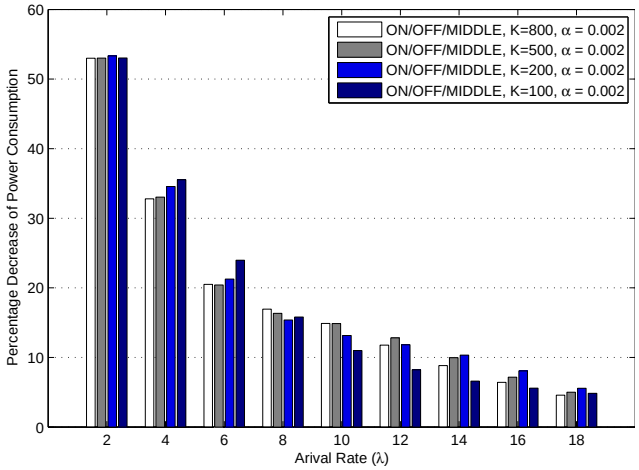

Fig. 15 Percentage decrease of power in single queue system

425(W). The power consumed by a SETUP server is larger: $P_{SETUP} = 1.5P_{ON}$. Based on [2], a server which does not run any task, consumes about 60% energy of its peak power when running a job, so that $P_{MIDDLE} = 0.6P_{ON}$. We assume $P_{OFF \to MIDDLE} = 0.8P_{ON}$ in both models to realize the tests.

5.4.1 Single queue model

Firstly, we measure the consumption of a single queue system. We set all related parameters like the experiments in Section 5.2. The test aims to prove the efficiency of our control algorithms to save the energy in comparison of two-state model.

In this way, Fig. 14 shows the comparison of the power consumption between the three-state and two-state models in a single queue model. Generally, we conclude that the consumption of three-state is less

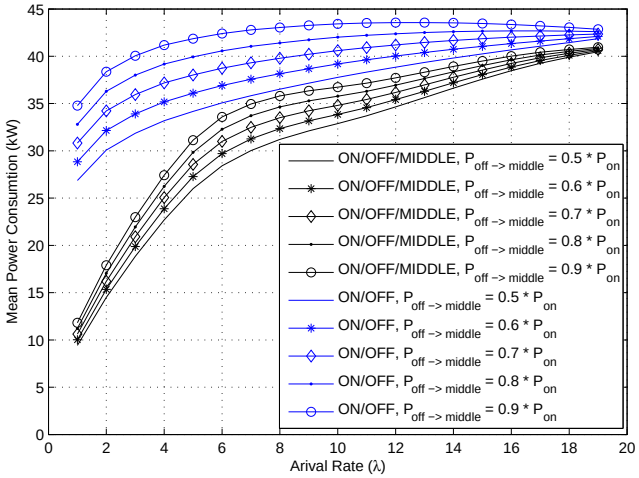

Fig. 16 Effect of $P_{OFF->MIDDLE}$ to power consumption in single queues system

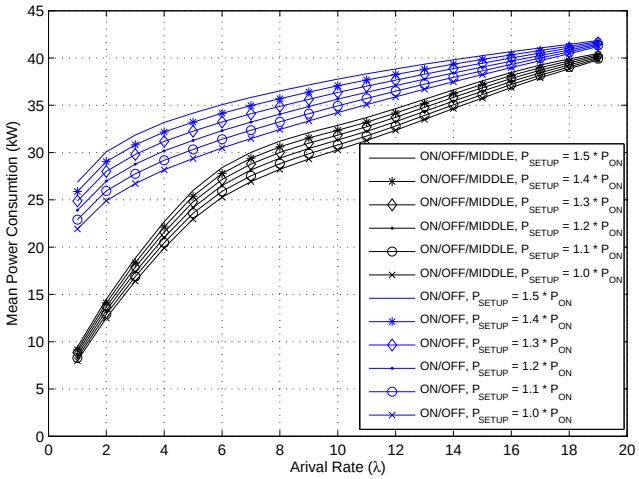

Fig. 17 Effect of P_{SETUP} to power consumption in single queues system

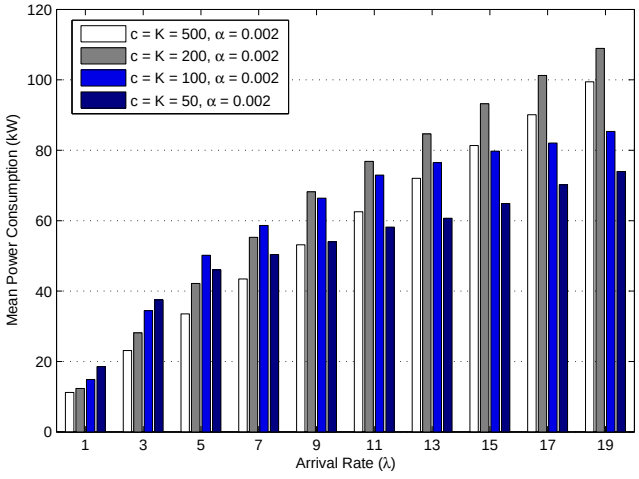

Fig. 18 Power consumption in multiple queues system

than two-state model. In addition, we also recognize that if λ tends to 19, the power consumptions figures in the two models are very close. The percentage of power decrease in the three-state is described in Fig. 15. Specifically, it decreases from approximate 50 to 5 if λ increases.

Through achieved results of the mean waiting time and power consumption, we realize that in the single queue model, the server management strategy not only decreases waiting time but also decreases power consumption. This demonstrates the performance effectiveness of our model in comparison with the two-state model, and our strategy manages switch processes of servers better than the ON/OFF model. Let us note the example in the two-state model if the system has not idle ON servers and then z jobs arrive. The queue size now is z and z OFF servers must be put immediately into SETUP mode to switch to ON state. However, during these z OFF servers are in SETUP mode, some jobs have been completed and jobs queue size is z' less than z, so $(z - z')$ servers in SETUP mode are turned off. Apparently, these $(z - z')$ servers will induce the energy wasting. By contrast, in our model, the turning process is controlled by our strategy. Then the number of servers in SETUP mode equals the number of servers that will be turned successfully into ON state. In this way, the number of servers in SETUP mode that is turned off is very small. However, also in Fig. 14, when λ tends to 19, system reaches the overload point, and our strategy does not work efficiently as interpreted before.

Fig. 16 and Fig. 17 show the effect of power consumption of servers in Setup mode (P_{SETUP}) and switching process from OFF to MIDDLE ($P_{OFF \to ON}$). In both figures, P_{SETUP} and $P_{OFF \to ON}$ have effects on the power consumption of system. However, both the power consumptions of two-state model and three-state model change in the same way that they either decrease or increase. So that the performance of our strategy is not affected.

5.4.2 Multiple queues model

We simulate multiple queues in a data center using our elastic queue management mechanisms. The goal of this testing is to measure the power consumption to evaluate the effect of our proposals. The consumption of system with multiple queues is shown in Fig. 18, in which the consumption increases with the arrival rate λ. The reason is that if arrival rate is higher, then more servers are turned on. We observe that if λ is small, the system with the smallest cluster capacity K has the highest power consumption. But, if λ is higher, the power consumption goes higher if the cluster capacity K is bigger.

From the experimental results of the multiple queue system we conclude that the cluster capacity of system affects both the mean waiting time of jobs and the power consumption. If the arrival rate is small, the higher capacity of the cluster decreases the power consumption, but it also decreases the QoS by longer waiting time. By contrast, if arrival rate is high, the higher capacity of cluster causes the increase of the power consumption and the decrease of the waiting time.

In summary, the gained results through energy tests and evaluations of both proposed models provided insights into the system performance in the aspect of performance-energy trade-off. While the single finite-capacity queue is proved its outstanding effectiveness as compared with the traditional ON-OFF model from the views of power consumptions as well as performance, the trade-off was shown very clearly by the multiple finite-capacity queue model tests above.

6 Conclusions

The study presented in this paper was carried out in the emerging context of cloud computing and large data centers bringing the issue how to improve the capability to serve the huge number of computational jobs at the same time. The research has concentrated on developing an elastic architecture for cloud data centers that allows to reduce service waiting time and thus to improve the QoS for the whole system. The experiments were performed and evaluated in the cloud environment by using simulation tool CloudSim. The research results bring both the theory and practical contributions as follows:

1. **State model for physical cloud servers**: besides two traditional states ON and OFF, the work proposes an intermediation state MIDDLE for servers in data centers. Based on the new state, a number of power-on servers can be kept available to eliminate the process of turning on servers during serving the arrival jobs. In this way, the service time for jobs equals only the time needed to switch servers from MIDDLE to ON through SETUP mode. From the view of users and customers of data center infrastructures, we can decrease the service waiting time for jobs.
2. **Single finite-capacity server queue using three-state model**: unlike other studies in the past, the work develops a server farm model with the single finite-capacity queue using three states for servers. The mathematical model is proved successfully that

can be a premise for other studies in the future. To ensure the operability of the model, a control algorithm called MSSS is proposed to manage the switching state process of servers from OFF to MIDDLE state in the manner of always keeping a certain server number in MIDDLE according to arrival job rate.

3. **Multiple finite-capacity server queues using three-state model**: in fact, the number of computational jobs coming into data centers is very large. Therefore, the data centers must exploit many job queues simultaneously. This work deals with the issue how to design the data center architecture with the multiple finite-capacity server queue using the three-state model. The architecture allows increasing and decreasing the queue number based on the arrival job rate elastically. The mechanisms are proposed for architectures including: cluster state management (three states and four switching actions for clusters), job scheduling (cluster choice algorithm), queue thresholds (upper and lower) and waiting cluster policy. The architecture thus helps to serve more jobs at once, to reduce the job block proportion, and to decrease the job waiting time.
4. **Power evaluations**: in single queue model, although servers with an intermediation state will cause the energy waste in some cases. However, in general, the three-state model with MSSS offers advantages as compared to the two-state model because switching state process is controlled well, especially if job intensity is not too large. In the multiple queue model, the system power consumption is changed conforming to two key factors: arrival job rate and server queue capacity. Both of them will affect the number of queues that are created or destructed. In this way, it also affects the power consumption of the whole system. The results achieved are a valuable proof for administrators who thus have an option to choose: optimize the energy consumption or service capacity.

Based on the contributions of this study, we plan to extend our models for different kinds of jobs including sub jobs. In addition, we are also going further by looking into the cloud and storage middleware deployments on data centers, such as OpenStack and Hadoop, using our architectures.

Acknowledgements This research is supported by the Vietnamese MOET's project "Research on development and deployment of cloud-based bioinformatics services applying for metagenomics" No. B2015-01-89, the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant Number 102.05-2014.28 and the Slovak national project "Methods and algorithms for the semantic processing of Big Data in distributed computing environment" VEGA 2/0167/16.

References

1. Average power use per server. URL `http://www.vertatique.com/average-power-use-server`. Accessed: 2016-2-4
2. Barroso, L.A., Holzle, U.: The case for energy-proportional computing. Computer **40**(12), 33–37 (2007)
3. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience **24**(13), 1397–1420 (2012)
4. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1), 23–50 (2011)
5. Gandhi, A., Harchol-Balter, M., Adan, I.: Server farms with setup costs. Performance Evaluation **67**(11), 1123–1138 (2010)
6. Goswami, V., Patra, S.S., Mund, G.: Performance analysis of cloud with queue-dependent virtual machines. In: Recent Advances in Information Technology (RAIT), 2012 1st International Conference on, pp. 357–362. IEEE (2012)
7. Halabian, H., Lambadaris, I., Lung, C.H.: Network capacity region of multi-queue multi-server queueing system with time varying connectivities. In: Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on, pp. 1803–1807. IEEE (2010)
8. Hamilton, J.: Cooperative expendable micro-slice servers (cems): low cost, low power servers for internet-scale services. In: Conference on Innovative Data Systems Research (CIDR09)(January 2009). Citeseer (2009)
9. Jarzab, M., Zielinski, K.: Adaptable service oriented infrastructure provisioning with lightweight containers virtualization technology. Computing and Informatics **34**(6), 1309–1339 (2015)
10. Karthick, A., Ramaraj, E., Kannan, R.: An efficient tri queue job scheduling using dynamic quantum time for cloud environment. In: Green Computing, Communication and Conservation of Energy (ICGCE), 2013 International Conference on, pp. 871–876. IEEE (2013)
11. Karthick, A., Ramaraj, E., Subramanian, R.G.: An efficient multi queue job scheduling for cloud computing. In: Computing and Communication Technologies (WCCCT), 2014 World Congress on, pp. 164–166. IEEE (2014)
12. Kato, M., Masuyama, H., Kasahara, S., Takahashi, Y.: Effect of energy-saving server scheduling on power consumption for large-scale data centers. MANAGEMENT **12**(2), 667–685 (2016)
13. Katz, R.H.: Tech titans building boom. IEEE Spectrum **2**(46), 40–54 (2009)
14. Khazaei, H., Mišić, J., Mišić, V.B.: Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems. Parallel and Distributed Systems, IEEE Transactions on **23**(5), 936–943 (2012)
15. Kim, H.S., Shin, D., Yu, Y., Eom, H., Yeom, H.Y.: Towards energy proportional cloud for data processing frameworks. In: SustainIT (2010)
16. Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X.: Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: IN-

FOCOM, 2012 Proceedings IEEE, pp. 945–953. IEEE (2012)
17. Lin, C.C., Liu, P., Wu, J.J.: Energy-efficient virtual machine provision algorithms for cloud systems. In: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, pp. 81–88. IEEE (2011)
18. Liu, L., Wang, H., Liu, X., Jin, X., He, W.B., Wang, Q.B., Chen, Y.: Greencloud: a new architecture for green data center. In: Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session, pp. 29–38. ACM (2009)
19. Liu, Y., Whitt, W.: Algorithms for time-varying networks of many-server fluid queues. INFORMS Journal on Computing **26**(1), 59–73 (2013)
20. Liu, Y., Whitt, W.: Stabilizing performance in many-server queues with time-varying arrivals and customer feedback. Tech. rep., Working paper (2014)
21. Long, S., Zhao, Y., Chen, W.: A three-phase energy-saving strategy for cloud storage systems. Journal of Systems and Software **87**, 38–47 (2014)
22. Luo, Z., Qian, Z.: Burstiness-aware server consolidation via queuing theory approach in a computing cloud. In: Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 332–341. IEEE (2013)
23. Milenkovic, M., Castro-Leon, E., Blakley, J.R.: Power-aware management in cloud data centers. In: Cloud Computing, pp. 668–673. Springer (2009)
24. Nguyen, B.M., Tran, D., Nguyen, Q.: A strategy for server management to improve cloud service qos. In: Distributed Simulation and Real Time Applications (DS-RT), 2015 IEEE/ACM 19th International Symposium on, pp. 120–127 (2015). DOI 10.1109/DS-RT.2015.14
25. Nguyen, M.B., Tran, V., Hluchy, L.: A generic development and deployment framework for cloud computing and distributed applications. Computing and Informatics **32**(3), 461–485 (2013)
26. Phung-Duc, T.: Server farms with batch arrival and staggered setup. In: Proceedings of the Fifth Symposium on Information and Communication Technology, pp. 240–247. ACM (2014)
27. Phung-Duc, T.: Multiserver queues with finite capacity and setup time. In: Analytical and Stochastic Modelling Techniques and Applications, pp. 173–187. Springer (2015)
28. Powell, M.J.: A fast algorithm for nonlinearly constrained optimization calculations. In: Numerical analysis, pp. 144–157. Springer (1978)
29. Toporkov, V., Yemelyanov, D., Potekhin, P., Toporkova, A., Tselishchev, A.: Metascheduling and heuristic co-allocation strategies in distributed computing. Computing and Informatics **34**(1), 45–76 (2015)
30. Zhang, Q., Zhani, M.F., Zhang, S., Zhu, Q., Boutaba, R., Hellerstein, J.L.: Dynamic energy-aware capacity provisioning for cloud computing environments. In: Proceedings of the 9th international conference on Autonomic computing, pp. 145–154. ACM (2012)
31. Zhang, Q., Zhu, Q., Boutaba, R.: Dynamic resource allocation for spot markets in cloud computing environments. In: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, pp. 178–185. IEEE (2011)