Future Generation Computer Systems 108 (2020) 361–371

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Q-learning based dynamic task scheduling for energy-efficient cloud computing

Ding Ding [a,*], Xiaocong Fan [b], Yihuan Zhao [a], Kaixuan Kang [a], Qian Yin [a], Jing Zeng [a]

[a] *School of Computer and Information Technology & Beijing Key Lab of Traffic Data Analysis and Mining, Beijing Jiaotong University, Beijing 100044, China*
[b] *College of Science and Mathematics, California State University San Marcos, CA 92096, USA*

A R T I C L E   I N F O

Article history:
Received 31 May 2019
Received in revised form 14 November 2019
Accepted 5 February 2020
Available online 28 February 2020

Keywords:
Cloud computing
Task scheduling
Energy efficiency
Queueing model
Q-learning

A B S T R A C T

High energy consumption has become a growing concern in the operation of complex cloud data centers due to the ever-expanding size of cloud computing facilities and the ever-increasing number of users. It is critical to find viable solutions to cloud task scheduling so that cloud resources can be utilized in an energy-efficient way while still meeting diverse user requirements in real time. In this research we propose a Q-learning based task scheduling framework for energy-efficient cloud computing (QEEC). QEEC has two phases. In the first phase a centralized task dispatcher is used to implement the M/M/S queueing model, by which the arriving user requests are assigned to each server in a cloud. In the second phase a Q-learning based scheduler on each server first prioritizes all the requests by task laxity and task life time, then uses a continuously-updating policy to assign tasks to virtual machines, applying incentives to reward the assignments that can minimize task response time and maximize each server's CPU utilization. We have conducted simulation experiments, which have confirmed that implementing a M/M/S queueing system in a cloud can help to reduce the average task response time. The experiments have also demonstrated that the QEEC approach is the most energy-efficient as compared to other task scheduling policies, which can be largely credited to the M/M/S queueing model and the Q-learning strategy implemented in QEEC.

1. Introduction

The cloud computing paradigm promises to provide on demand access to always-on computing resources such as storage, platforms, and software applications [1,2]. Cloud computing, along with IoT and big data applications, has dramatically changed the IT operations in the industry, and become the key infrastructure for the next generation of communications.

Due to the vast and ever-expanding size of cloud computing facilities and the ever-increasing number of users [3,4], energy consumption has emerged as one of the key challenges in the operation of complex cloud data centers. It is reported that the average increase of energy consumption has reached 12% per year from 2005 to 2010 and becomes higher and higher in recent years [5]. High energy consumption not only translates to high cost, but also produce excessive heat emissions, which often leads to system unreliability and performance degradation.

Among various ways towards energy-efficient cloud computing, being responsible for resource management and optimization, cloud task scheduling is deemed as one of the key areas to search for viable solutions to the energy consumption issue. Task scheduling has a direct impact on both the utilization of cloud resources and the QoS (Quality of Service) in response to user requests, therefore it is the very place where we can explore innovative designs that can manage resources in an energy-efficient way while still meeting diverse user requirements in real time.

With our focus on energy-efficient cloud computing, in this research we propose QEEC, a two-phase framework for cloud task scheduling. The first phase is at the cloud level, where a centralized task dispatcher is used to assign each of the arriving user requests to an appropriate server by pushing the request onto a global queue. The second phase is at the virtual machine level, where a Q-learning based scheduler on each server uses a time window to determine when to process the tasks on the request queue. For every time window, the scheduler first prioritizes all the requests (tasks) by the user constraints specified in SLAs (Service Level Agreements), then uses a continuously-updating policy to assign tasks to virtual machines, applying incentives to reward the assignments that can minimize task response time and maximize each server's CPU utilization.

* Corresponding author.
E-mail addresses: dding@bjtu.edu.cn (D. Ding), sfan@csusm.edu (X. Fan), 16120462@bjtu.edu.cn (Y. Zhao), 18120371@bjtu.edu.cn (K. Kang), 19120428@bjtu.edu.cn (Q. Yin), 19120435@bjtu.edu.cn (J. Zeng).

https://doi.org/10.1016/j.future.2020.02.018
0167-739X/© 2020 Elsevier B.V. All rights reserved.

The main contribution of this work can be summarized as follows:

1. The QEEC framework is proposed to tackle the issue of energy consumption in both task assignment and task scheduling. Task assignment operates at a global level where a cloud data center is treated as a whole. Task scheduling operates at a local level where each server further schedules tasks to individual virtual machines.
2. In global task assignment, we have compared two queueing models, S-M/M/1 and M/M/S, and chosen the M/M/S model by implementing a centralized task dispatcher in QEEC. We have shown, both analytically and experimentally, that the M/M/S model can yield shorter task response time, which leads to improved energy efficiency.
3. In local task scheduling, a dynamic task ordering strategy is used first to prioritize user tasks based on constraints such as task laxity and task life time. In so doing, more urgent tasks can be considered earlier by the Q-learning based task scheduler. This helps in enhancing the quality of cloud services.
4. Also in local task scheduling, we have designed a reward function such that the scheduler will be rewarded for assigning a task to a virtual machine (VM) only when the VM can meet the task's deadline, it can offer the least waiting time, and its service rate is the closest match with the service rate requested by the task. This helps the scheduler to dynamically adapt to the diverse user requests and the heterogeneity of cloud resources, improving the resource utilization and thereby reducing energy consumption of the whole cloud system.

The remainder of this paper is organized as follows. Related works are given in Section 2. We provide a simple energy consumption model and introduce the QEEC framework in Section 3. In Section 4 we focus on analytical analysis of two queueing models and argue that the M/M/S queuing system is more favorable in improving energy efficiency of a cloud data center because it can yield shorter task response time. In Section 5 we discuss the design of a Q-learning based task scheduler. Section 6 presents the simulation experiments and performance evaluation, and Section 7 concludes the paper.

2. Related works

Task scheduling is very critical to the operation of a cloud data center. A good task scheduling strategy can not only effectively reduce the response time of user tasks and meet user requirements with a varying levels of constraints, it can also improve the utilization of cloud resources, reduce energy consumption and operational costs [6].

With energy consumption becoming a major sustainability issue in cloud computing system, energy management approaches have been studied extensively in the context of energy-aware task scheduling. Heuristic-based methods have been proposed to design dynamic scheduling strategies. Xu et al. [7] have adopted a simulated annealing method to find the optimal allocation strategy to reduce the energy consumption of cloud computing systems. A multi-objective evolutionary algorithm combined with a low-level backfill heuristic has been proposed in [8] to find out the effective mapping of workflow to resources, which attempts to maximize several measures related to the quality of services and to save energy at the same time. However, these heuristic-based methods can only obtain approximate, suboptimal solutions and are not suitable for task scheduling in heterogeneous environments.

Machine learning approaches have been studied for resource and power management in large-scale cloud systems. Vasic et al. [9] have proposed a framework to model energy consumption in a cloud resource management system, which can react to workload changes quickly and automatically by learning the preferred resource allocation from past experience. Dabbagh et al. [10] have proposed a resource management framework to reduce energy consumption in cloud data centers with k-means for data clustering and random Wiener filters for workload prediction. However, these methods can be over-tuned by the scenarios given at the training time. The unexpected situations that were not included in training can lead to SLA violations, which may not be useful for the environments with large dynamic changes. Tesauro et al. have applied the reinforcement learning (RL) approach to autonomic resource allocation in cloud computing environment [11]. Recent studies have further shown the feasibility of RL approaches in resource allocation and power management. For instance, Hussin et al. [12] have presented a dynamic scheduling algorithm that incorporates reinforcement learning for optimizing resource utilization and improving energy efficiency. Farahnakian et al. [13] have proposed a reinforcement learning-based dynamic consolidation method to optimize resource utilization and reduce energy consumption. Lin et al. [14] have proposed a reinforcement learning-based power management framework for data centers to reduce the server pool energy consumption with reasonable average job response time. Liu et al. [15] have proposed a novel hierarchical framework by using deep reinforcement learning technique for solving the overall resource allocation and power management problem in cloud computing systems.

Queueing theory plays an important role in analyzing the scheduling performance of stochastic service systems. Some recent studies have taken the queueing-theory-based approach to examine the energy consumption issue in cloud data centers. A queueing model for cloud performance management has been proposed in [16], where virtual machines are used as service institutions and can be added or deleted dynamically so as to adapt to changes in system size. Tan et al. [17] have used a M/M/1 queueing system to model a cloud computing system for analyzing the mean power consumption. Jiang et al. [18] have also used the M/M/1 model to study performance management in cloud computing. In [19] a system has been treated as a series of M/M/1 interacting submodels.

Most of the studies mentioned above are based on the simple M/M/1 queueing model. However, the M/M/1 model ignores too many system details so it is not appropriate for modeling the actual cloud environments and the heterogeneous settings of virtual machines where user tasks are scheduled to run. A few researchers have studied energy-saving strategies in cloud data centers using multi-server vacation queuing theory [20,21]. Peng et al. [22] have also proposed a system model comprised of separate submodels including a task schedule submodel, a task execute submodel and a task transmission submodel, which can be analyzed in the order of processing of user requests. However, they have only considered servers using random scheduling strategies such as FCFS.

3. Two-phase task scheduling framework

We frame the context as follows. The servers in a cloud computing environment are indexed by i where $1 \leq i \leq S$. Each server i contains a fixed number n_i of virtual machines, represented by $\mathrm{VM}_i^1, \mathrm{VM}_i^2, \ldots, \mathrm{VM}_i^{n_i}$, respectively. We assume that user requests are independent when they arrive at the cloud environment: (a) each request is treated as an integral task and cannot be further split into smaller tasks; (b) each task can be fully fulfilled by one and only one virtual machine; (c) at any time each virtual machine is either idle or executing only one task; and (d) there are no inter-task communications or dependencies, although there may exist exchange of statistical information among servers.

Below we start with a simple energy consumption model, then introduce our proposed two-phase task scheduling framework toward energy-efficient cloud computing.

3.1. *Energy consumption in cloud*

The energy consumption of a cloud system within t units of time is given by:

$$E(t) = \int_0^t P_{cloud}\, dt, \tag{1}$$

where P_{cloud} is the power consumption rate (electrical energy consumed per unit time) of the could system under consideration. Obviously, P_{cloud} varies over time, and it is proportionally related to how many servers and how many virtual machines in each server are servicing tasks.

A server is idle when all its virtual machines are idle and is busy otherwise. When a server i is ready (for serving tasks), let p_i^0 be the probability of the server i being idle. The probability of the server i being busy is $1 - p_i^0$.

Hence, the energy consumption model in Eq. (1) can be broken down as the sum of power consumption by all servers:

$$E(t) = \int_0^t (\sum_{i=1}^{S} (p_i^0 \times P_i^{\perp} + (1 - p_i^0) \times P_i^{\top}))\, dt, \tag{2}$$

where $P_i^{\perp}$ is the power consumption rate of the server i when it is idle, and $P_i^{\top}$ is the power consumption rate of the server i when it is busy, i.e., at least one of its virtual machines is servicing a task request.

In previous studies it has been shown that in general $P_i^{\perp}$ can be above 50% of the server's peak power consumption [23]. In addition, $P_i^{\top}$ often has a close relation with the resource utilization rate of the server [24], where resources mainly include CPU, run-time memory, and disk storage. With our concentration on computing-intensive tasks and their scheduling, we choose to stick with the linear CPU-power consumption model as proposed in [25]:

$$P_i^{\top} = c_i U_i + b_i, \tag{3}$$

where U_i is the CPU-utilization rate of the server i, c_i is the influence factor of the utilization rate and b_i is a constant; both c_i and b_i can be obtained by linear regression.

Given a fixed set of user task requests, let t be the time duration counted from the instant when the tasks start to arrive at the cloud system under concern till the instant when all the tasks have been completely serviced, we have the following observations from Eq. (2):

(1) The energy consumption of a cloud caused by a set of task requests depends on how long the tasks stay in the cloud. With everything else being fixed (say P_{cloud} does not vary over time), the shorter the task response time, the more energy saved.

(2) The energy consumption is also closely related to the CPU-utilization rate of the servers in a cloud. It is been widely accepted that the increase of CPU-utilization contributes to the improvement of energy efficiency. In Eq. (2), $P_i^{\top}$ increases as each U_i increases, but at the same time the task throughput of the server i will also be increased and typically at a much higher rate (say, 15% increase of the power consumption rate may result in 80% increase of execution speed). Consequently, the task response time t would be reduced significantly, leading to improved energy efficiency.

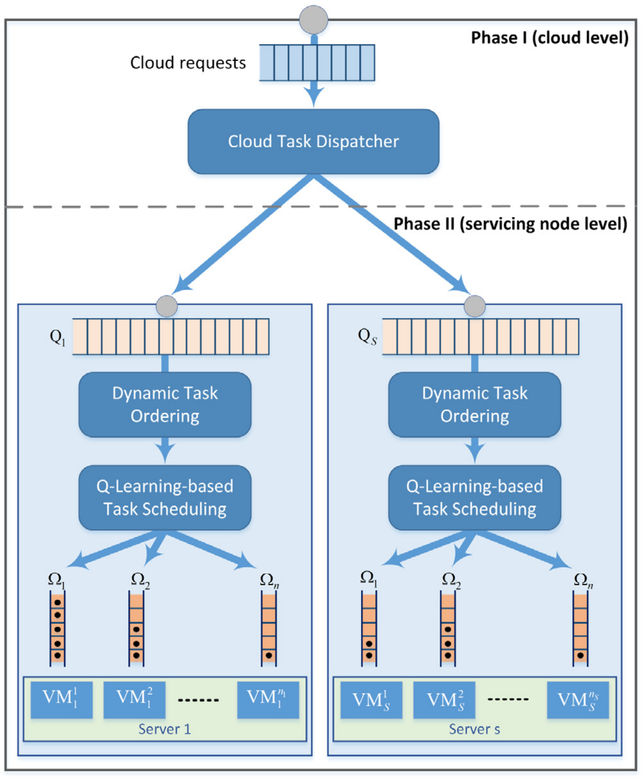


Fig. 1. Two-phase QEEC framework for energy-efficient cloud computing.

Based on the energy consumption model explained above, our objective is to propose and evaluate a cloud task scheduling framework that can achieve overall energy efficiency by reduced task response time and increased CPU-utilization rate.

3.2. *Two-phase QEEC framework*

As shown in Fig. 1, our QEEC framework has two phases.

- **Phase I: centralized task allocation** A cloud data center can have a large number of servicing nodes, typically of the order of hundreds or thousands. Due to the nature of cloud data centers and diversity of user requests, we choose to implement a centralized dispatcher for task allocation. In particular, we assume that a global request queue is employed at a cloud data center to buffer all the incoming user requests. The task dispatcher keeps monitoring and managing the outstanding user requests, and takes prompt actions to dispatch them to each physical servicing node. In Section 4 we will explain analytically why it is beneficial to implement a centralized task dispatcher at a cloud data center.
- **Phase II: Q-learning-based task scheduling:** User requests are stored in a local request queue at each servicing node (physical server). The requests are dynamically re-ordered based on the requirements that are derived from the SLAs associated with each request. A Q-learning-based adaptive scheduling approach is then adopted to dispatch tasks to each virtual machine with the objective of minimizing the overall CPU-utilization rate. In Section 5 we will cover the strategies used in dynamic task ordering and the adaptive scheduling approach framed as a Q-learning problem.

4. Centralized task allocation and performance analysis

Given a fixed set T of user task requests, let W_{cloud} be the average task response time by the cloud system under concern. The total energy consumption for all the tasks in T, counted from the instant when the tasks start to arrive at the cloud system under concern till the instant when all the tasks have been completely serviced, can be approximated as

$$E(t) = \int_0^t P_{cloud}\, dt \approx (|T| \times W_{cloud}) \times P_{cloud}, \tag{4}$$

When $|T|$ is fixed, reducing energy consumption boils down to finding ways to reduce W_{cloud} and/or P_{cloud}. Below we examine two queueing analysis models to gain a better understanding of the key factors that may contribute to the reduction of W_{cloud} in a cloud system.

4.1. Analysis of coordinated servicing nodes

When a centralized task dispatcher is used in a cloud data center, as shown in Fig. 1, it can be modeled as a M/M/S queueing system. In general, a M/M/S queueing system assumes a Markov process (Poisson distribution) for the task arrivals, a negative exponential distribution for the service time, and a finite number S of independent servers providing the same type of services (with the same or varying service qualities) [26].

We consider the non-trivial situation where $|T| \gg S$, that is, the number of incoming requests is much larger than the number of servers. Let λ be the average number of requests arriving at the cloud data center per interval (i.e. λ is the rate parameter of the assumed Poisson distribution), and μ be the average departure (service completion) rate. Here we assume the service time of all the independent servers follows the same negative exponential distribution with parameter μ. Since the task dispatcher typically distributes requests to the servers evenly, when all the servers are treated as a whole, the overall service rate of S servers is $S\mu$. Let $\rho = \frac{\lambda}{S\mu}$, which is the service intensity of the M/M/S queueing system.

Let $P_j (j \geq 0)$ be the probability of there being j requests in the system. When the system is at a steady state, the arrival rate should be equal to the departure rate. We thus have the following state equilibrium equations:

$$\begin{aligned} &j = 0 : \mu \times P_1 = \lambda \times P_0 \\ &j \geq 1 : \lambda \times P_{j-1} + \mu \times P_{j+1} = (\lambda + \mu) \times P_j \end{aligned} \tag{5}$$

From the regularity condition $\sum_{j=0}^{\infty} P_j = 1$, the stable probability of the whole cloud system being idle, denoted by P_0, can be derived as given below:

$$P_0 = \left[\frac{(S\rho)^S}{S!(1-\rho)} + \sum_{j=0}^{S-1} \frac{(S\rho)^j}{j!} \right]^{-1} \tag{6}$$

The expected number of requests in the system, denoted by $L_{M/M/S}$, is given in Eq. (7):

$$L_{M/M/S} = \sum_{j=0}^{\infty} jP_j = \rho \frac{P_0 (S\rho)^S}{S!(1-\rho)^2} + S\rho \tag{7}$$

$L_{M/M/S}$ is the sum of the number of requests queued in the system waiting for service and the number of requests (tasks) currently being serviced by the servers in the cloud. The expected task response time of the M/M/S system, denoted by $W_{M/M/S}$, is given in Eq. (8):

$$W_{M/M/S} = \frac{L_{M/M/S}}{\lambda} = \frac{1}{\mu} \left(\frac{P_0 (\frac{\lambda}{\mu})^S}{(S-1)!(S - \frac{\lambda}{\mu})^2} + 1 \right) \tag{8}$$

Now, putting Eqs. (4) and (8) together, when $\frac{\lambda}{\mu} \leq 1$[27], a cloud data center will offer shorter average task response time as more servers are put into use (S is increased), and the total energy consumption would be reduced.

4.2. Analysis of multiple independent servicing nodes

Now let us consider another setting where there is no centralized task dispatcher in the cloud to coordinate S servers. In such a case user requests randomly arrive at each server independently and we can use S number of M/M/1 queueing systems to conduct a similar performance analysis.

Let $S = 1$, from Eq. (8) we have:

$$W_{M/M/1} = \frac{1}{\mu - \lambda}, \tag{9}$$

which gives us the expected task response time of the M/M/1 system. The expected task response time of S number of M/M/1 systems, denoted by $W^S_{M/M/1}$, can be derived by replacing λ in Eq. (9) by λ/S (due to parallel processing, the response time can be calculated by focusing on one M/M/1 system with the task interarrival rate reduced equally to λ/S), then we have,

$$W^S_{M/M/1} = \frac{S}{S\mu - \lambda}, \tag{10}$$

Take $S = 2$ for an example, the average response time of two M/M/1 systems is $W^2_{M/M/1} = \frac{2}{2\mu - \lambda}$, while the average response time of a M/M/2 system is $W_{M/M/2} = \frac{4\mu}{4\mu^2 - \lambda^2}$.

4.3. Coordinated vs independent servicing nodes

Now we compare $W^S_{M/M/1}$ with $W_{M/M/S}$.

When $S = 2$, $W^2_{M/M/1} = \frac{2}{2\mu-\lambda} = \frac{4\mu+2\lambda}{4\mu^2-\lambda^2} > \frac{4\mu}{4\mu^2-\lambda^2} = W_{M/M/S}$, which says the average task response time of a M/M/2 system is less than that of two M/M/1 systems. In general, we have the following result.

Theorem 1 (*Task Response Time Comparison*). *Given a M/M/S system and a system composed of S number of M/M/1 systems, we have $W^S_{M/M/1} \geq W_{M/M/S}$.*

Proof. Note that we have $\rho = \frac{\lambda}{S\mu}$ in the M/M/S system. From (6) we have:

$$\begin{aligned} \frac{1}{P_0} &= \frac{(S\rho)^S}{S!(1-\rho)} + \sum_{j=0}^{S-1} \frac{(S\rho)^j}{j!} \\ &= \frac{(S\rho)^S}{S!(1-\rho)} + \frac{1}{0!} + \frac{S\rho}{1!} + \cdots + \frac{(S\rho)^{S-1}}{(S-1)!} \\ &\geq \frac{(S\rho)^S}{S!(1-\rho)} + \frac{(S\rho)^{S-1}}{(S-1)!} = \frac{S(S\rho)^{S-1}}{S!(1-\rho)}. \end{aligned}$$

Then we have $\frac{\rho}{1-\rho} \geq \frac{P_0 (S\rho)^S}{S!(1-\rho)^2}$. Replace $\rho = \frac{\lambda}{S\mu}$ and simplify, we have

$$\frac{S}{S\mu - \lambda} \geq \frac{1}{\mu} \left(\frac{P_0 (\frac{\lambda}{\mu})^S}{(S-1)!(S - \frac{\lambda}{\mu})^2} + 1 \right),$$

which is $W^S_{M/M/1} \geq W_{M/M/S}$. The equality holds only when $S = 1$. □

From the above analysis, we see that a cloud data center may not necessarily offer shorter average task response time as more servers are put into use. In particular, a cloud data center with a centralized task dispatcher (i.e. one M/M/S system) will in general have shorter average task response time than a data center with the same number of completely independent servers (i.e. S-M/M/1 systems). Moreover, the more number S of servers in a M/M/S system, the shorter average task response time we may expect from the system.

We thus choose to adopt the M/M/S system in our QEEC framework by implementing a centralized task dispatcher. In our simulation experiments in Section 6 we will also see that the M/M/S system does help in reducing the average task response time, leading to improved energy efficiency.

5. Q-learning based dynamic task scheduling

Once a task has been assigned to a physical machine, it can then be scheduled to a virtual machine (VM) for execution. Since in this phase our focus is on the scheduling inside an individual server i, we will ignore the server index and refer to the VMs by VM_1, VM_2, and VM_n instead (where $n = n_i$). As shown in Fig. 1, when a task has just been assigned to a server it is first pushed onto a request queue $\mathbb{Q}$, waiting to be further scheduled to a specific virtual machine. Also, there is a buffer queue implemented for each virtual machine.

Briefly, the workflow of each server is as follows. The task scheduler on each server uses a time window to determine when to process the tasks on the request queue $\mathbb{Q}$. The time window can be a few seconds or minutes, depending on how often the server is configured to process user tasks. Due to the random nature of the arrival of user tasks, the number of tasks to be considered during different time windows can vary but follow a certain distribution. For every time window, the scheduler first removes all the tasks (requests) from its request queue $\mathbb{Q}$, then assigns each task to a specific virtual machine by pushing it to the buffer queue associated with that virtual machine. Independently, each virtual machine VM_k keeps grabbing and running tasks from its buffer queue Ω_k one after another, and it will be in its idle state in case Ω_k is empty when it completes a task and tries to grab another.

Algorithm 1: TaskBatchScheduling

```
  Input: $\mathbb{Q}$, timeWin ;      // timeWin is the time window
         for batch processing
1 Initialize Q ;                     // Q is the Q-value table
2 $\Omega = [\Omega_1, \Omega_2, \cdots, \Omega_k, \cdots, \Omega_n] = [\emptyset, \emptyset, \cdots, \emptyset, \cdots, \emptyset]$;
3 for every timeWin elapsed do
      // correspond to a Q-learning episode
4     rlist = RemoveAllRequestsFrom($\mathbb{Q}$);
5     plist = OrderingTasksByDynamicPriority(rlist);
6     $\overline{s_t} = (|\Omega_1|, |\Omega_2|, \cdots, |\Omega_k|, \cdots, |\Omega_n|)$;
7     $\Omega$ = QScheduling(Q, $\overline{s_t}$, $\Omega$, plist);
```

In practice, Ω_k must have a limited capacity. Also, different virtual machines on a server may be set with different capacities based on their resource settings. To simplify the design of our Q-learning algorithm, we choose to use the same capacity c for all VMs on a server. When $c = 0$, no task is allowed to wait; in other words, each task assigned to a VM has to be immediately executed by the VM. When $c = 5$, at most 5 more tasks can be assigned to a VM while it is running an existing task.

Algorithm 1 gives our batch-style design of task scheduling. It is worth noting that in step (1) the Q-value table can be randomly initialized. Even better, Q can be pre-learned by pilot-running Algorithm 1 using some realistic user requests in the actual cloud environment. Once the Q has been tuned for a certain period of time such that it can support task scheduling reasonably well, it then can be used in online task scheduling.

Two strategies are employed in Algorithm 1 to boost the efficiency of task scheduling. In step (5), we choose to dynamically order all the tasks from $\mathbb{Q}$, and in step (7) we use a Q-learning-based approach to reward task assignments that can help reduce the task response time and improve the CPU-utilization. We next explain the two strategies in detail.

5.1. Dynamic task ordering

Q-learning is a value-based reinforcement learning(RL) approach [28]. It is model-free; as the environment is explored (with an exploration policy like ϵ-greedy), the action-value function (Q-table) will be iteratively updated using the Bellman Equation, allowing the decision maker to always choose the best action at each decision time.

In most of the existing Q-learning-based approaches to cloud task scheduling, user requests are often processed on a first-come first-served (FCFS) basis. This is problematic because the performance of a cloud data center depends heavily on how well it can meet its users' needs as specified in the SLA requirements. For example, the submitted user tasks may have a variety of requirements on task deadlines (completion time). Intuitively, all others being equal, the task with a shorter deadline should be scheduled first.

In this study we focus on computing-intensive tasks [29] where the (data) transmission time is negligible. Formally a task is denoted as a tuple:

$$T_j = \langle len_j, \phi_j, d_j, w_j \rangle,$$

where len_j, ϕ_j, d_j, and w_j represent the length of task T_j (say, how many MI–million instructions), its demand on service rate, its deadline, and how long it is been in the system. The values for len_j, ϕ_j and d_j can be derived from the SLA of the user request j, w_j is the time elapsed since the user request j has been put into the request queue of the server.

In order to shorten the overall task response time we choose to order all the tasks by their priorities before they are considered by our Q-learning-based task scheduler. Specifically, priorities are assigned to tasks based on the following two measures.

- **Task laxity:** The laxity of a task at a given time is the maximum time its execution can be delayed before it is sure to miss its deadline. The least laxity first algorithm is well-known in the real-time systems field. Given a task $T_j = \langle len_j, \phi_j, d_j, w_j \rangle$, its laxity is given by $laxity(T_j) = (d_j - t - len_j/\phi_j)$, where t is the current system time, and len_j/ϕ_j is the execution time demand of task T_j (Note that we treat len_j/ϕ_j as a hard demand on execution time; it is the time that the task T_j will occupy a VM, even though the VM has a higher service rate than ϕ_j. In such a case the VM would be idle for part of the demanded execution time and its CPU-utilization rate would be less than 1). The smaller laxity a task has, the more urgent the task is to be executed, so the higher priority should be assigned to the task.
- **Task life time:** While it is critical to meet task deadlines, it is also important to shorten the total time a task stays in the system as much as possible in order to offer faster task response. Hence, when everything else being equal, a task with a longer life time is assigned with a higher priority.

Based on the definition above, the dynamic rank for a task T_j, denoted by $rank(T_j)$, can be formulated as follows:

$$rank(T_j) = \delta_1 \times laxity(T_j) - \delta_2 \times w_j, \qquad (11)$$

where δ_1 and δ_2 are adjustable positive weights for task laxity and task life time, respectively. A task with a lower rank value has a higher scheduling priority.

5.2. Q-learning based task scheduling

We now formulate cloud task scheduling as a Q-learning problem. Our task scheduler uses the Q-learning algorithm to gradually learn connections between the state space and the action space.

5.2.1. State space and action space

The state space $\mathbb{S}$ is a finite set of states that the server can be in. A state at time t, denoted by $\overline{s_t}$, is defined as a tuple:

$$\overline{s_t} = \langle l_1, l_2, \ldots, l_k, \ldots, l_n \rangle,$$

where $l_k = |\Omega_k| (1 \le k \le n)$, is the number of tasks waiting in the buffer queue of VM$_k$ at time t. For example, given $\overline{s_t} = \langle 3, 1, \ldots, l_k, \ldots, 0 \rangle$, it represents a state where Ω_1 contains 3 user tasks assigned to VM$_1$, Ω_2 contains 1 user task assigned to VM$_2$, and Ω_n is empty. Note that a VM must be busy in running a task when its buffer queue is not empty. A VM will become idle if it finds its buffer queue empty after completing a task. In order to improve the CPU-utilization rate of a server, a heuristic is to keep all its VMs busy by adding new tasks to any buffer queues that may become empty soon.

The action space $\mathbb{A}$ is a set of actions that can be performed. In this context, selecting an action means to consider assigning the first task in the ordered task list to a specific VM$_k (1 \le k \le n)$, and taking the action means the task has been successfully assigned to VM$_k$ (i.e., the task has been appended to the end of Ω_k—the buffer queue of VM$_k$). In general, an action can be represented by a vector with each element indicating assigned/not assigned (1/0). Because each task is considered individually in task scheduling, an action can have one and only one element taking the value of 1. For example, $a_t = (0, 1, 0, \ldots, 0)$ refers to the action where the current task under concern is to be assigned to the virtual machine VM$_2$ at time t.

5.2.2. Observations from the cloud environment

Assume at the current state the selected action a is to assign a task $\eta = \langle len', \phi', d', w' \rangle$ (i.e., the first one in the task list ordered by priorities) to VM$_x$, that is, all elements but the xth element of a are 0s. By taking the action a, the scheduler assigns the task η to VM$_x$ by appending it to the end of the buffer queue Ω_x.

In Q-learning, immediately after performing an action, the learning agent will be aware of the new state, and it can receive/observe a signal from the environment. The signals, either a reward or penalty value, can help the scheduler to learn an optimal policy over time which may incrementally improve its performance in future task scheduling. Let us next design a reward function for the scheduler.

Assume the current user task running on VM$_k (1 \le k \le n)$ is $\hat{T}_k$ and its remaining execution time demand is $\hat{e}_k$. The expected waiting time (from now on) of a task η, if it were scheduled to get service from VM$_k (1 \le k \le n)$, is defined as:

$$\text{waiting}(\eta, \text{VM}_k) = \hat{e}_k + \sum_{T_j = \langle len_j, \phi_j, d_j, w_j \rangle \in \Omega_k} \frac{len_j}{\phi_j} \tag{12}$$

The reward function is defined as follows:

$$r = \begin{cases} 1 & (\text{waiting}(\eta, \text{VM}_x) + len'/\phi' < d')\ \& \\ & (\text{waiting}(\eta, \text{VM}_x) = \min_k(\text{waiting}(\eta, \text{VM}_k)))\& \\ & (x = \arg\max_{\{k | v_k \ge \phi'\}} (\phi'/v_k)) \\ 0 & \text{waiting}(\eta, \text{VM}_x) + len'/\phi' < d' \\ -1 & \text{otherwise.} \end{cases} \tag{13}$$

That is, by taking the action a (i.e. assigning the task η to VM$_x$), the scheduler will receive a reward of 1 in case that (a) the deadline of the task η can be satisfied by VM$_x$; (b) the expected waiting time from VM$_x$ is the minimal (no greater than that when η were assigned to any other VM); and (c) VM$_x$ promises the best CPU-utilization for the task η (among all the VMs that can meet η's demand on service rate (i.e., $v_k \ge \phi'$), v_x is one of the closest match to ϕ').

Note that the objective of condition (a) is to avoid breaking task deadlines; the objective of condition (b) is to minimize the overall task response time (always allocating tasks to a VM that can offer a shorter waiting time); and the objective of condition (c) is to prefer scheduling a task on a VM with a service rate that can best match with the task's demand on service rate. All others being equal, by condition (c) we can regulate the scheduler to maximally utilize each VM's service capacity.[1] In so doing, the server's overall CPU-utilization rate can be kept high, which can consequently improve the server's energy consumption efficiency.

In Eq. (13), the scheduler will receive a reward of 0 if at least the deadline of the task will not be broken by taking the assignment action. Otherwise, the scheduler will be punished with a negative penalty of -1. In sum, the reward function is designed to train the scheduler to improve the server's CPU-utilization rate and to reduce task response time, leading to an energy-efficient cloud computing environment in the long run.

5.2.3. Q-learning algorithm

Algorithm 1 gives the context of our Q-learning approach, where $Q : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is initialized. Q is also called an Action-Value function that maps a state–action combination $(\bar{s}, a)$ to a real number that represents the "quality" (or cumulative reward) of taking the action a in the state $\bar{s}$.

Because each $\Omega_k (1 \le k \le n)$ can have at most c number of waiting tasks, the dimension of Q is $c^n \times n$: there are c^n different states (rows) in total with c distinct states for each VM, and there are n distinct assignment actions. The dimensionality of the state space can be adjusted by setting a feasible value for c.

Each iteration of the loop in line (3) of Algorithm 1 is an episode of the Q-learning, where QScheduling() given in Algorithm 2 is invoked to update both Q and Ω.

Algorithm 2: QSCHEDULING

```
   Input: Q, $\overline{s_t}$, $\Omega$, plist
   // $\alpha$, $\gamma$ are predefined learning parameters
1  repeat
2      Wait if $|\Omega_k| = c$ for all $1 \le k \le n$;
3      $\eta_t$ = RemoveFirstRequestFrom(plist);
4      $a_t$ = SelectAction(Q, $\overline{s_t}$) ;          // using $\epsilon$-greedy
5      Take action $a_t$;    // append $\eta_t$ to $\Omega_x$ assume all but
       the $x^{th}$ element of $a_t$ are 0s
6      Observe $r_t$ and the new state $\overline{s_{t+1}}$;
7      $Q(\overline{s_t}, a_t) = (1-\alpha)Q(\overline{s_t}, a_t) + \alpha[r_t + \gamma \cdot \max_a Q(\overline{s_{t+1}}, a)]$;
8      $\overline{s_t} = \overline{s_{t+1}}$;
9  until plist is empty;
   Output: Q, $\Omega$
```

In Algorithm 2, QScheduling() takes Q and Ω as inputs, as well as the current state $\overline{s_t}$ and a new list of tasks *plist* ordered by priorities. Each iteration of the loop processes one task η_t until all tasks have been scheduled to some virtual machines. Note that in line (2) the scheduler will throttle itself in case that the

[1] We deem it as short sighted if a task with a service rate demand of 50mips were assigned to a VM with a service rate of 100mips. It is better reserve this VM if it is highly likely that another task with a service rate demand of 100mips is arriving in a moment.

buffer queues for all the VMs are full, because no user task should be dropped as it moves to this stage. While it is waiting the scheduler can also take advantage of the time to learn from past experience without taking 'real' actions, but this is beyond the scope of this paper.

Given the current system state $\overline{s_t}$, the scheduler first chooses an action according to the ϵ-greedy policy [30] (do more exploration than exploitation in the beginning of the training by using a larger ϵ, which is progressively reduced as the Q-values move closer to optimal). Because it is always the first task in *plist* that is to be assigned, choosing an action here virtually means to select one out of all the possible next states. Given the current state $\overline{s_t} = \langle l_1, l_2, \ldots, l_k, \ldots, l_n \rangle$, there are at most n possible states as shown below to be considered (Which state is possible is relative to the capacities of the buffer queues. For example, the state $\overline{s_k}$ should not be considered if $l_k + 1 > c$):

$$\begin{array}{lllllllll}
\overline{s_1} & = & \langle & l_1+1, & l_2, & \cdots, & l_k, & \cdots, & l_n\rangle, \\
\overline{s_2} & = & \langle & l_1, & l_2+1, & \cdots, & l_k, & \cdots, & l_n\rangle, \\
\vdots & & & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots, \\
\overline{s_k} & = & \langle & l_1, & l_2, & \cdots, & l_k+1, & \cdots, & l_n\rangle, \\
\vdots & & & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots, \\
\overline{s_n} & = & \langle & l_1, & l_2, & \cdots, & l_k, & \cdots, & l_n+1\rangle.
\end{array}$$

In case of exploration, the system randomly chooses one out of all the possible states. In case of exploitation the system chooses the possible state which can produce the most reward according to the existing Q-values. Assume that $\overline{s_x}$ is the chosen state, the chosen action a_t would have 1 for the xth element and all 0s elsewhere. In case of exploitation we also have $Q(\overline{s_t}, a_t) = \max_{a'} Q(\overline{s_t}, a')$.

In line (5), by performing the action, the task η_t is appended to the end of Ω_x—the buffer queue of the VM$_x$. In line (6), by observation $\overline{s_x}$ indeed becomes the new state, which is denoted by $\overline{s_{t+1}}$ indicating that it is the next state after $\overline{s_t}$. By observation, the system can also collect an immediate reward r_t from the environment (c.f. Eq. (13)), which is then used to update the Q-value function based on the following Bellman Equation (14):

$$Q'(\overline{s_t}, a_t) = (1-\alpha)Q(\overline{s_t}, a_t) + \alpha[r_t + \gamma \cdot \max_a \ Q(\overline{s_{t+1}}, a)] \quad (14)$$

where α is a learning rate parameter, and γ is the discount factor which determines the importance of future rewards.

The cumulative rewards in Q are thus progressively updated by Eq. (14), and in the long run the system may converge to an optimal policy for online cloud task scheduling.

6. Experimental validation

To evaluate the performance of our proposed QEEC approach, we followed the time-shared policy [31] to set up our experimental environment in CloudSim [32]. We have experimented with two small clouds (with 6 and 10 servers, respectively) and a larger cloud (with 20 servers). Since the results from these clouds allowed us to draw the same conclusions, we therefore only presented the experiment results from the cloud with 20 servers. The experiments were performed on Huawei TaiShan 2280 V2 with two Kunpeng 920, 48 Core@2.6 GHz CPU, and 8*32 GB memory. For each scenario setting, the results given below are the average of 10 experiments conducted independently.

6.1. Phase I: S-M/M/1 vs. M/M/S

In the first set of experiments, we compared two different settings for Phase I: with vs without a centralized task dispatcher utilized to forward requests to the independent cloud servers. According to our analysis, the former corresponds to a M/M/S system whereas the latter corresponds to a collection of M/M/1 systems.

Table 1
Parameter settings for a simulated cloud (without VMs).

Parameters	Homogeneous	Heterogeneous
Number of hosts (servers)	20	20
Host service rate (μ)	3000 mips	(500,3000) mips
Total number of tasks	1000	1000
Task service rate demand (ϕ_j)	(100,300) mips	(100,300) mips
Task length (len_j)	(500,3000) MI	(500,3000) MI

The key parameter settings are given in Table 1. In the homogeneous setting, all servers offered the same service rate of 3000 mips, while in the heterogeneous setting, servers offered different service rates that were randomly generated from a uniform distribution ranging from 500 mips to 3000 mips. There were 1000 cloudlets generated to represent user requests, which arrived at the cloud system following a Poisson distribution. The lengths of user tasks are random integers within the range (500, 3000)MI.

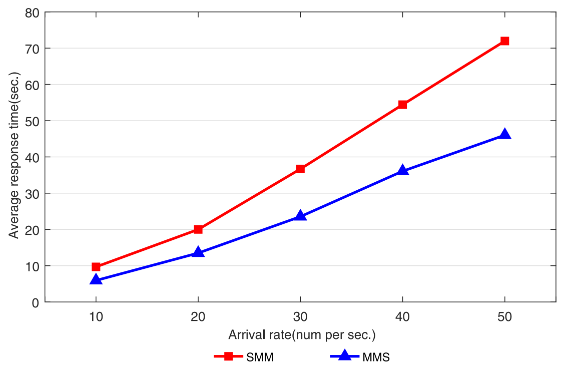
The experiment results about the average task response time are given in Table 2 and plotted in Fig. 2.

Fig. 2(a) plots the average task response time resulted from twenty M/M/1 systems (denoted as SMM in the figure) and a M/M/20 system (denoted as MMS), both have homogeneous servers offering services at the same rate. For both SMM and MMS, the average task response time increased as the arrival rate of user tasks increased from 10/s to 50/s. This is because a higher arrival rate means more user tasks will be queued in the system waiting for services. Overall, the average task response time of MMS was improved by 55.7% compared to SMM.

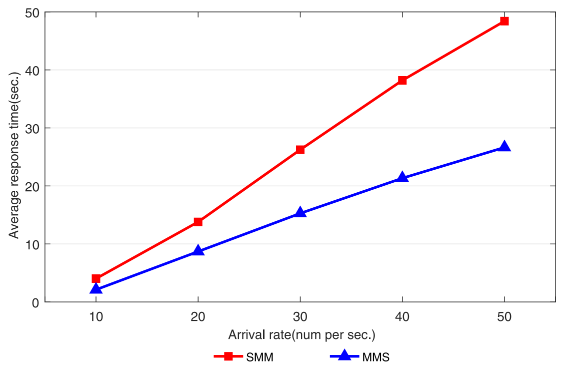
Fig. 2(b) plots the average task response time resulted from SMM and MMS, both have twenty heterogeneous servers offering services at varying rates. First of all, we see that MMS still outperformed SMM in the heterogeneous setting. Actually, according to the results plotted in Figs. 2(a) and 2(b), the response time was reduced largely when the cloud data center was configured with heterogeneous servers. Moreover, in Fig. 2(b) SMM shows a greater increase rate than MMS, causing bigger and bigger performance gaps as the task arrival rate increases. Overall, the average task response time of MMS was improved by 68.7% compared to SMM.

This set of experiments has confirmed our analysis given in Section 4 that the M/M/S model can offer shorter task response time than the S-M/M/1 model under the same conditions.

It is worth noting that, as explained in Section 4.1, Eq. (8) implies that the benefit of putting more servers into use in a cloud center is under the condition that the incoming user task rate is less than the service rate, i.e., $\lambda < \mu$. This has been confirmed in our pre-experimental studies: the response time rises exponentially as the user task arrival rate increases and the service rate decreases. In order to keep the response time at a tolerable level, we either have to increase the service rate or reduce the user task arrival rate. In Fig. 2 we see that the average response time can be above 20 s when the user task arrival rate is approaching to 20/s. Moreover, the trend remains stable if we compare the results when λ is between 10/s and 20/s and the results when λ is between 20/s and 50/s. Hence, in other experiments reported below the task arrival rate was set to be between 10/s and 20/s.



(a) Homogeneous servers: average response time



(b) Heterogeneous servers: average response time

Fig. 2. Performance comparison between a cloud with twenty M/M/1 servers and a cloud with twenty servers forming a M/M/20 system. In this set of experiments, no virtual machines are configured for the servers.

Table 2
Average response time comparison between S-M/M/1 and M/M/S.

Cloud server	System	Request arrival rate (num per sec.)					Improvement of M/M/S over S-M/M/1
		$\lambda = 10$	$\lambda = 20$	$\lambda = 30$	$\lambda = 40$	$\lambda = 50$	
Homogeneous	S-M/M/1	9.68	20.01	36.68	54.41	70.97	55.70%
	M/M/S	5.94	13.50	23.55	36.07	46.02	
Heterogeneous	S-M/M/1	4.03	13.80	26.24	38.20	48.40	68.71%
	M/M/S	2.12	8.70	15.28	21.34	26.64	

Table 3
Parameter settings for a simulated cloud (with VMs).

Parameters	Settings
Number of hosts (servers)	20
Host service rate (μ)	(500,5000) mips
Max number of virtual machines	50
Virtual machine service rate (v_k)	(100,1000) mips
Total number of tasks	1000
Task service rate demand (ϕ_j)	(50,1000) mips
Task length (len_j)	(500,3000) MI

6.2. Phase II: Dynamic task ordering and Q-Learning

In this set of experiments we evaluate the two strategies implemented in Phase II: dynamic task ordering and Q-learning based scheduling. The key parameter settings are given in Table 3. Here each server is set up to contain at most 50 virtual machines with varying service rates ranging from 100 mips to 1000 mips ($\mu = \sum_{1\leq k\leq n} v_k$).

We first evaluated the impact of dynamic task ordering on the task response time. We compared two settings: the first one, denoted as MMS-O0, used a centralized task dispatcher (i.e. M/M/S system) without dynamic ordering before tasks were scheduled; the second one, denoted as MMS-O1, used both a centralized task dispatcher in Phase I and the task ordering strategy in Phase II. In particular, we chose to treat task deadline as a hard constraint and task life time as a secondary factor. The best performance could be obtained when δ_1 and δ_2 in Eq. (11) were set to 0.7 and 0.3 in the pre-experiments, so we used the same settings in this experiment.

Fig. 3(a) plots the results. It can be seen that MMS-O1 performed better than MMS-O0, and the performance gains became bigger as the task arrival rate increased. Overall, the average response is reduced by 10.1% in MMS-O1 as compared to MMS-O0. According to Eq. (11), tasks are ordered by their respective laxity and life time, where tasks with a shorter laxity and a longer life time are serviced with a higher priority. Obviously, this dynamic task ordering strategy helped in reducing the average task response time.

Next we further considered the use of the Q-learning based scheduler (designed in Section 5.2) to evaluate the impact of dynamic task ordering on the task response time. We compared two settings: the first one, denoted as MMS-O0Q, used a centralized task dispatcher in Phase I and Q-learning based scheduler in Phase II; the second one, denoted as MMS-O1Q, had the same settings as MMS-O0Q except that in Phase II the task ordering strategy was also employed before Q-learning based scheduler.

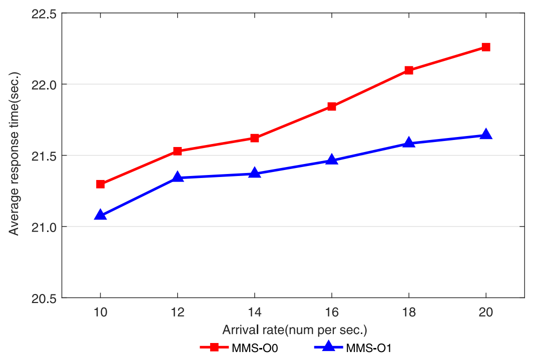
In this experiment, we used the following settings for the learning-related parameters. The learning rate α was set to 0.5 so that old and new information are treated equally in the learning process [33]. The discount factor γ was set to 0.9 to aspire the learning agent to minimize the long-term penalty. we used a larger exploration rate $\epsilon = 0.5$ in the beginning of the training, which was reduced by 0.05 in each learning epoch to encourage exploitation as the Q-values moved closer to the optimal.

The experiment result is plotted in Fig. 3(b). We can still see that the dynamic task ordering strategy continues to show its important impact on reducing the average task response time when Q-based learning is employed. Putting Figs. 3(a) and 3(b) together, we have the following observations: (a) overall MMS-O0Q performed better than MMS-O1, which suggests that the Q-learning strategy is relatively more critical than the task ordering strategy; (b) the use of the Q-learning based scheduler significantly reduced the average task response time (compare MMS-O0Q with MMS-O0, and MMS-O1Q with MMS-O1); (c) the best performance was produced by MMS-O1Q, which employed both the dynamic task ordering and the Q-learning based scheduling.

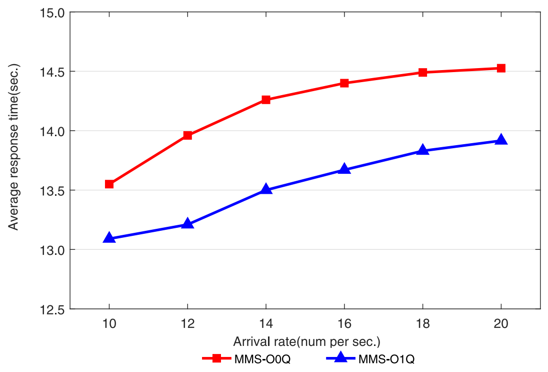
6.3. Comparison with other scheduling policies

The MMS-O1Q setting as used in the last subsection is exactly the approach illustrated in the QEEC framework in Fig. 1, henceforth we refer to it by QEEC and compare it with three widely used task scheduling policies [34] (all have the same M/M/S settings as QEEC in Phase I):

- MMS-RANDOM: in which user tasks are assigned to virtual machines randomly;

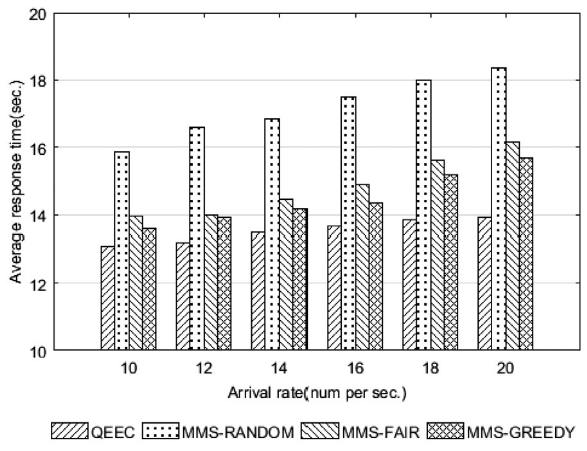


(a) Dynamic task ordering: average response time

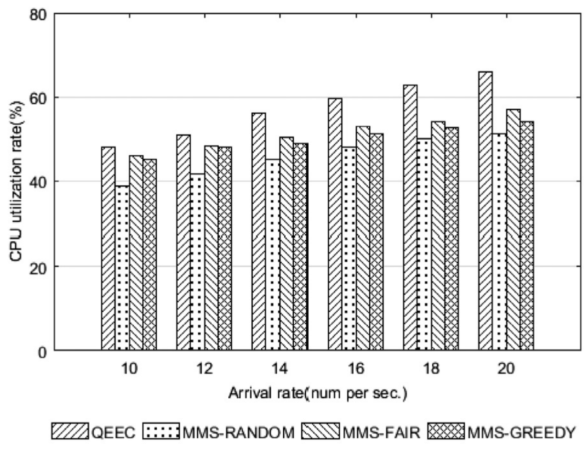


(b) Q-learning based scheduling: average response time

Fig. 3. Performance of a cloud with twenty servers forming a M/M/20 system. In this set of experiments, each server in a cloud contains up to 50 virtual machines with varying service rates.



(a) Comparison of average task response time



(b) Comparison of CPU-utilization rate

Fig. 4. Performance comparison: different task scheduling policies employed in Phase II.

- MMS-FAIR: in which it takes turns to assign user tasks to virtual machines;
- MMS-GREEDY: in which a user task is assigned to the virtual machine which can offer the minimum completion time.

Fig. 4 plots the average response time and the average CPU utilization rate when the user task arrival rate increased from 10/s to 20/s.

It can be seen that the RANDOM policy produced the longest average response time and the lowest CPU utilization rate. The reason is that it tries to schedule tasks to virtual machines at random without considering the current workloads of the virtual machines. The FAIR policy aims to balance loads among all the virtual machines. However, it becomes more and more difficult to achieve balanced task allocation as the task arrival rate increases. The main idea of the GREEDY policy is to achieve faster task execution by finding the virtual machine with the minimum completion time for each user task, however, too much emphasize on each individual task may not lead to the overall minimum completion time for all the tasks. In contrast, in our QEEC approach, the minimization of task response time is treated as one of the key factors both in prioritizing user tasks and in rewarding the Q-learning based scheduler. The scheduler is also regulated (via the Q-learning process) to maximize the utilization of each VM's service capacity. In so doing, it can help each server, thus the whole cloud, to maintain a high overall CPU-utilization.

To close the loop, in Section 3 we have claimed that energy-efficient cloud computing can be achieved by reducing task response time and increasing CPU-utilization rate. It has been shown in Fig. 4 that our QEEC approach, among the policies considered, can produce the shortest average task response time and the highest CPU-utilization rate. Analytically, QEEC should also be the most energy-efficient. To see this clearly, in Fig. 5 we plot the total system energy consumption when the four scheduling policies were used, summing up the power consumption of cloud servers for a fixed duration of time (ref. Eq. (4)). Clearly, QEEC consumed the least amount of energy among the four policies. With the increase of λ, the energy saving is more remarkable as compared to the other three approaches.

Furthermore, we conducted another experiment to further compare our QEEC approach with two other Q-learning based approaches for energy efficient task scheduling: a baseline Q-learning based approach (basic QL in short) and an improved Q-learning based approach proposed by Mostafavi et al. [35](referred to as improved QL in short). Fig. 6 plots the results.

It can be seen that QEEC performed the best on the energy consumption. This can be attributed to the use of a centralized task dispatcher in the task assignment phase and the dynamic task ordering and Q-learning based scheduling in the task scheduling phase.

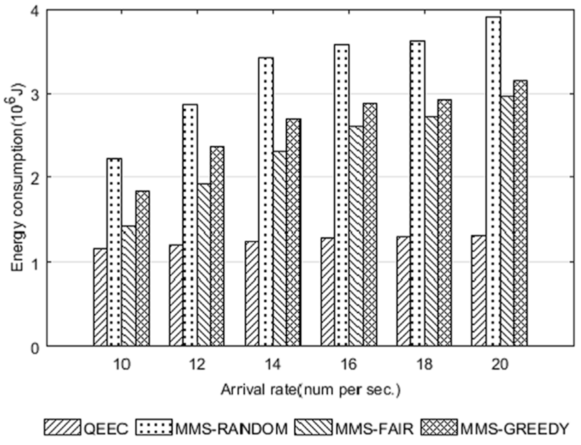


Fig. 5. Energy consumption of four scheduling policies.

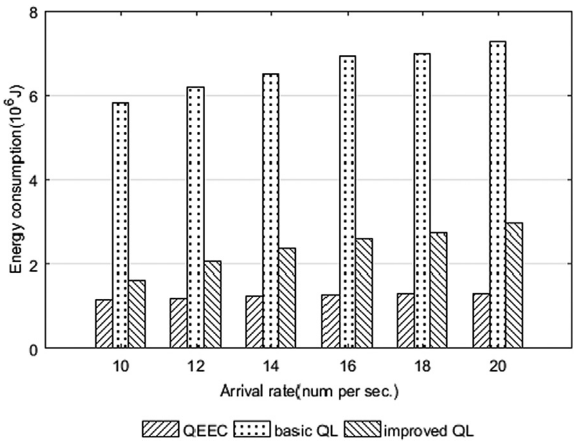


Fig. 6. Energy consumption of three QL-based scheduling policies.

7. Conclusion and future work

Cloud task scheduling is challenging because cloud data centers are often composed of heterogeneous computing nodes with various resource types and their availability is often changing on the fly. The heterogeneity and dynamics of cloud resources, along with high variability of requests from the ever-increasing number of users make it hard to offer high-quality cloud computing services while at the same time achieve a competent energy-efficiency.

We have proposed a novel task scheduling framework, named QEEC, to investigate effective ways to improve energy saving in cloud data centers. QEEC distinguishes the task assignment phase and the task scheduling phase. The task assignment phase is at the cloud level, where a centralized task dispatcher is used to assign each of the arriving user tasks to an appropriate server by pushing the task onto the server's request queue. In the task scheduling phase, a Q-learning based scheduler on each server uses a time window to determine when to process the tasks on the request queue. For every time window, the scheduler first prioritizes all the requests (tasks) by the user constraints specified in SLAs, then uses a continuously-updating policy to assign tasks to virtual machines, applying incentives to reward the assignments that can minimize task response time and maximize each server's CPU utilization.

The experimental results have confirmed our theoretical analysis that using a centralized task dispatcher in a cloud environment (i.e. implementing a M/M/S queueing system) can help to reduce the average task response time. The experiments have also shown that the QEEC approach outperformed other task scheduling policies. This can be attributed to the minimization of task response time being treated as one of the key factors both in prioritizing user tasks and in rewarding the Q-learning based scheduler. The scheduler is also rewarded to maximize the utilization of each VM's service capacity. In so doing, it can effectively reduce the energy consumption in a cloud environment.

One future direction of this work is to further evaluate the QEEC framework in large scale cloud environments with hundreds of servicing nodes, and to investigate other heuristics that may further improve the performance of Q-learning for cloud task scheduling.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (61300176, 61672086) and Fundamental Research Funds for the Central Universities (2019JBM025).

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, Commun. ACM 53 (2010) 50–58.

[2] P.M. Mell, T. Grance, The NIST Definition of Cloud Computing, Tech. Rep., National Institute of Standards and Technology, 2011.

[3] W. Lin, B. Lin, Distributed Computing, Cloud Computing and Big Data, China Machine Press, Beijing, China, 2015.

[4] S. Singh, I. Chana, A survey on resource scheduling in cloud computing: issues and challenges, J. Grid Comput. 14 (2016) 217–264.

[5] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (2012) 755–768.

[6] P. Liu, Cloud Computing, Electronic Industry Press, Beijing, China, 2015.

[7] X. Xu, L. Cao, X. Wang, Resource pre-allocation algorithms for low-energy task scheduling of cloud computing, J. Syst. Eng. Electron. 27 (2016) 457–469.

[8] S. Iturriaga, B. Dorronsoro, S. Nesmachnow, Multiobjective evolutionary algorithms for energy and service level scheduling in a federation of distributed datacenters, Int. Trans. Oper. Res. 24 (2016) 199–228.

[9] N. Vasic, D. Novakovic, S. Miucin, D. Kostic, R. Bianchini, Dejavu: accelerating resource allocation in virtualized environments, in: 17th International Conference on Architectural Support for Programming Languages and Operating Systems, 2012, pp. 423–435.

[10] M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, Energy-efficient cloud resource management, in: IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, 2014, pp. 386–391.

[11] G. Tesauro, N.K. Jong, R. Das, M.N. Bennani, A hybrid reinforcement learning approach to autonomic resource allocation, in: 3rd International Conference on Autonomic Computing, 2006, pp. 65–73.

[12] M. Hussin, Y.C. Lee, A.Y. Zomaya, Efficient energy management using adaptive reinforcement learning-based scheduling in large-scale distributed systems, in: International Conference on Parallel Processing, 2011, pp. 385–393.

[13] F. Farahnakian, P. Liljeberg, J. Plosila, Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning, in: Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2014, pp. 500–507.

[14] X. Lin, Y. Wang, M. Pedram, A reinforcement learning-based power management framework for green computing data centers, in: International Conference on Cloud Engineering, 2016, pp. 135–138.

[15] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, Y. Wang, A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning, in: International Conference on Distributed Computing Systems, 2017, pp. 372–382.

[16] H. Chen, S. Li, A queueing-based model for performance management on cloud, in: 6th International Conference on Advanced Information Management and Service, 2011, pp. 83–88.

[17] Y. Tan, G. Zeng, W. Wang, Policy of energy optimal management for cloud computing platform with stochastic tasks, J. Softw. 23 (2012) 266–278.

[18] J. Jiang, J. Lu, G. Zhang, G. Long, Optimal cloud resource auto-scaling for web applications, in: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2013, pp. 58–65.

[19] H. Khazaei, J. Mišić, V.B. Mišić, Performance analysis of cloud computing centers using m/g/m/m+r queuing systems, IEEE Trans. Parallel Distrib. Syst. 23 (2012) 936–943.

[20] D. Liao, K. Li, G. Sun, V. Anand, Y. Gong, Z. Tan, Energy and performance management in large data centers: a queuing theory perspective, in: International Conference on Computing, Networking and Communications, 2015, pp. 287–291.

[21] C. Yin, S. Jin, An energy-saving strategy based on multi-server vacation queuing theory in cloud data center, J. Supercomput. 74 (2018) 6569–6597.

[22] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, W. Lin, Random task scheduling scheme based on reinforcement learning in cloud computing, Cluster Comput. 18 (2015) 1595–1607.

[23] T. Chaabouni, M. Khemakhem, Energy management strategy in cloud computing: a perspective study, J. Supercomput. 74 (2018) 6569–6597.

[24] W. Lin, W. Wu, H. Wang, J.Z. Wang, C.-H. Hsu, Experimental and quantitative analysis of server power model for cloud data centers, Future Gener. Comput. Syst. 86 (2018) 940–950.

[25] A. Beloglazov, R. Buyya, Y.C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, in: Advances in Computers, vol. 82, Elsevier, 2010, pp. 47–111..

[26] Y. Hu, Operational Research Foundation and Application, Higher Education Press, Beijing, China, 2014.

[27] Z. You, Research on Cloud Resource Scheduling Strategy Based on M/M/N Queuing Model (Master's thesis), University of Electronic Science and Technology of China, 2014.

[28] R.S. Sutton, A.G. Barto, F. Bach, Reinforcement Learning: An Introduction, The MIT Press, 1998.

[29] S.K. Tesfatsion, E. Wadbro, J. Tordsson, A combined frequency scaling and application elasticity approach for energy-efficient cloud computing, Sustain. Comput. Inform. Syst. 4 (2014) 205–214.

[30] D. Xiao, Y. Jie, C. Shi, Non-liner energy consumption model for cloud computing, J. Beijing Univ. Posts Telecommun. 39 (2016) 107–111.

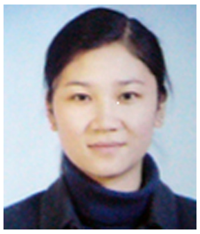
[31] H. Himani, H.S. Sidhu, Comparative analysis of scheduling algorithms of cloudsim in cloud computing, Int. J. Comput. Appl. 97 (2014) 27–33.

[32] R.N. Calheiros, R. Ranjan, A. Beloglazov, A.F.De Rose Cesar, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. - Pract. Exp. 41 (2011) 23–50.

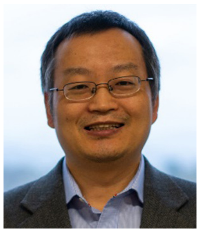
[33] A. Kontarinis, V. Kantere, N. Koziris, Cloud resource allocation from the user perspective: A bare-bones reinforcement learning approach, in: International Conference on Web Information Systems Engineering, 2016, pp. 457–469.

[34] D. Cui, Z. Peng, J. Xiong, B. Xu, W. Lin, A reinforcement learning-based mixed job scheduler scheme for grid or iaas cloud, IEEE Trans. Cloud Comput. (2017).

[35] S. Mostafavi, F. Ahmadi, M.A. Sarram, Reinforcement-learning-based foresighted task scheduling in cloud computing, 2018, https://arxiv.org/abs/1810.04718v1.



Ding Ding received her Ph.D. degree in Computer Application Technology from Beijing Jiaotong University in 2011. She is an associate professor at School of Computer and Information Technology in Beijing Jiaotong University. With main research interests in cloud computing and parallel and distributed computing, she has published more than 30 related papers in recent years. Her current research focuses on building highly efficient, fast and accurate resource scheduling in a massive resources cloud computing environment to enhance both the satisfaction of cloud users and the performance of whole cloud system.

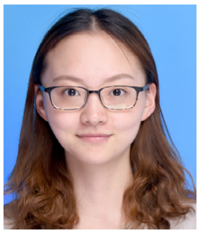


Xiaocong Fan received the Ph.D. degree in Software Engineering from Nanjing University, China, in 1999. He is currently a Professor of Computer Science and Software Engineering, California State University, San Marcos, CA, USA. He has actively researched in the field of Multi-Agent Systems and Machine Learning, including fundamental issues in team collaboration and theoretical studies in proactive information sharing. He played key roles in the development of several teamwork-ready intelligent agent architectures—CAST, R-CAST, and SMMall. He is a Senior member of IEEE.

Yihuan Zhao received her B.Eng. degree in Software Engineering from ShanXi University in 2016. She is studying in School of Computer and Information Technology of Beijing Jiaotong University for M.S. degree now. Her research interests include cloud computing, reinforcement learning and resource scheduling.

Kaixuan Kang received her B.S. degree in Software Engineering from YanShan University in 2018. She is studying in School of Computer and Information Technology of Beijing Jiaotong University for Ph.D. degree now. Her research interests include cloud computing and reinforcement learning.

Qian Yin received her B.S. degree in Information and Computing Science from Hebei GEO University in 2019. She is studying in School of Computer and Information Technology of Beijing Jiaotong University for M.S. degree now. Her research interests include cloud computing and workload prediction.



Jing Zeng received her B.Eng. degree in Computer Science and Technology from Central China Normal University in 2019. She is studying in School of Computer and Information Technology of Beijing Jiaotong University for M.S. degree now. Her research interests include cloud computing, VM migration and workload prediction.