The Journal of Systems and Software 147 (2019) 147–161

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Energy-aware virtual machine allocation for cloud with resource reservation☆

Xinqian Zhang[a], Tingming Wu[a], Mingsong Chen[a,*], Tongquan Wei[a], Junlong Zhou[b], Shiyan Hu[c], Rajkumar Buyya[d]

[a] Shanghai Key Lab of Trustworthy Computing & MOE International Joint Lab of Trustworthy Software, East China Normal University, Shanghai 200062, China
[b] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China
[c] Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931, USA
[d] School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC 3010, Australia

ARTICLE INFO

Article history:
Received 8 August 2018
Accepted 27 September 2018
Available online 19 October 2018

Keywords:
Cloud computing
Virtual machine allocation
Evolutionary algorithm
Energy efficiency
VM acceptance ratio

ABSTRACT

To reduce the price of pay-as-you-go style cloud applications, an increasing number of cloud service providers offer resource reservation-based services that allow tenants to customize their virtual machines (VMs) with specific time windows and physical resources. However, due to the lack of efficient management of reserved services, the energy efficiency of host physical machines cannot be guaranteed. In today's highly competitive cloud computing market, such low energy efficiency will significantly reduce the profit margin of cloud service providers. Therefore, how to explore energy efficient VM allocation solutions for reserved services to achieve maximum profit is becoming a key issue for the operation and maintenance of cloud computing. To address this problem, this paper proposes a novel and effective evolutionary approach for VM allocation that can maximize the energy efficiency of a cloud data center while incorporating more reserved VMs. Aiming at accurate energy consumption estimation, our approach needs to simulate all the VM allocation updates, which is time-consuming using traditional cloud simulators. To overcome this, we have designed a simplified simulation engine for CloudSim that can accelerate the process of our evolutionary approach. Comprehensive experimental results obtained from both simulation on CloudSim and real cloud environments show that our approach not only can quickly achieve an optimized allocation solution for a batch of reserved VMs, but also can consolidate more VMs with fewer physical machines to achieve better energy efficiency than existing methods. To be specific, the overall profit improvement and energy savings achieved by our approach can be up to 24% and 41% as compared to state-of-the-art methods, respectively. Moreover, our approach could enable the cloud data center to serve more tenant requests.

1. Introduction

Based on the virtualization of computing resources, cloud computing enables on-demand service provision in a pay-as-you-go manner, which makes the upgrades and maintenance of both software and hardware easier than before (Buyya et al., 2009). Therefore, instead of incurring high upfront costs in building their own private platforms, more and more enterprises and communities choose cloud computing platforms to deploy their commercial and scientific applications. However, the proliferation of cloud computing requires the establishment of large-scale data centers that contain thousands of computing nodes, which in turn results in excessive energy consumption and negative environmental impacts (Wajid et al., 2016).

Although the execution of more cloud services needs more energy, the energy consumption in cloud data centers is mainly due to the low utilization of computing resources. It has been estimated that the average resource utilization in most data centers is lower than 30% (Barroso et al., 2013) and the energy consumed by idle nodes is more than 70% of peak energy (Fan et al., 2007a). In other words, most energy is wasted for doing nothing. The rising energy consumption increases the ownership cost and reduces

☆ This work was supported by Natural Science Foundation of China (Grant Nos. 61872147 and 61802185), the Fundamental Research Funds for the Central Universities, Shanghai Municipal Natural Science Foundation (Grant No. 16ZR1409000), Natural Science Foundation of Jiangsu Province under Grant BK20180470, and China HGJ Project under Grant 2017ZX01038102-002.

* Corresponding author.
E-mail address: mschen@sei.ecnu.edu.cn (M. Chen).

https://doi.org/10.1016/j.jss.2018.09.084
0164-1212/© 2018 Elsevier Inc. All rights reserved.

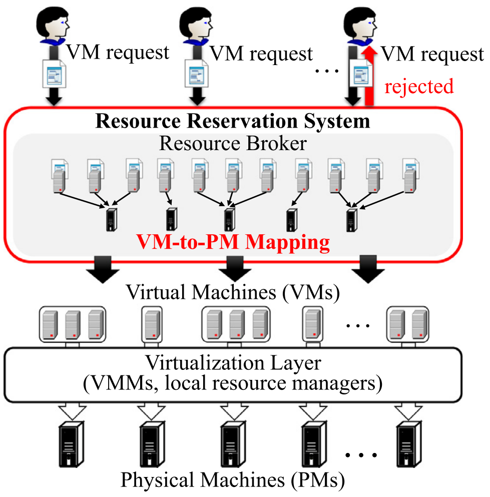


Fig. 1. Workflow of reservation-based cloud computing.

the return on cloud infrastructure investments. Therefore, how to fully explore the capacity of data centers to achieve high energy efficiency is becoming a major concern of cloud service providers.

To efficiently manage cloud resources and reduce the unit price, an increasing number of cloud providers support the resource reservation option, which allows tenants to customize their cloud service requests. For example, Amazon Elastic Compute Cloud (EC2) (Cloud, 2016) has the option *Reserved Instances Pricing* which allows cloud capacity reservations within specific time windows (i.e., a fraction of a day or a week). Such a kind of computing mode is also supported by Microsoft Azure automation, Alibaba Cloud Batch Compute, and other mainstream cloud platforms. Fig. 1 shows how virtual machine requests are handled in the workflow of reservation-based cloud computing. Initially, cloud tenants submit their virtual machine reservation requests with specific requirements (e.g., required type of resources, time windows) to an IaaS (Infrastructure as a Service) provider. For example, if one tenant wants to conduct 3D rendering using the reservation-based cloud, he/she needs to provide the start time, CPU and GPU requirements and the longest rendering time to the IaaS provider. After receiving such a batch of VM reservations, the IaaS provider needs to figure out a new Virtual Machine (VM) to Physical Machine (PM) mapping based on available resources. From the perspective of IaaS providers, such a mapping should achieve maximum profit and meet all the tenants' requirements. Meanwhile, the newly incorporated VMs should not degrade the performance of other existing VMs in service. If there are not enough physical resources that can accommodate all the VMs, i.e, the Service Level Agreement (SLA) of some VM requests cannot be ensured, the unsatisfied requests will be rejected by their IaaS providers. When the reservation is confirmed, at the beginning of the next VM reservation cycle, all the accepted VMs will be dispatched to their designated PMs.

The VM-to-PM mapping plays an important role in determining the utilization of data centers (Pietri and Sakellariou, 2016; Chen et al., 2018). To achieve more profits in the very competitive cloud computing market, IaaS providers need to explore efficient VM-to-PM mappings that can achieve the highest energy efficiency while consolidating as much VM workload as possible. Since the VM allocation problem is an NP-hard problem, various energy-aware VM mapping approaches have been investigated (Kim et al., 2011; Calheiros and Buyya, 2014; Hwang and Pedram, 2013). However, most of them are based on the trade-off between energy consumption and system performance assuming that there are unlimited physical resources. This is not suitable for reservation-based cloud computing. In order to fully utilize the PMs and facilitate the management of reserved VMs, IaaS providers usually offer VM reservation on a limited number of exclusive PMs without mixing reserved and non-reserved VMs together. Consequently, VM rejection should be considered during the mapping generation. Moreover, due to the increasing new VM requests, the reservation cycle time is shortened drastically (e.g., from one week to one day). Accordingly, the update of reservation plans is becoming much more frequent.

In order to save operating cost and quickly respond to the new VM requests, IaaS providers should consider the following questions during the derivation of mapping solutions: (i) how to quickly achieve a VM-to-PM mapping which has the highest energy-efficiency? and (ii) how to incorporate more new VM requests to achieve the highest PM utilization without violating tenants' requirements? To address these two questions, this paper makes the following three major **contributions**:

- We propose a novel fitness function based on our defined term instruction-energy ratio, which can effectively reduce the overall energy consumption and maximize the resource utilization of reservation-based cloud data centers.
- Based on the evolutionary algorithm, we introduce a comprehensive VM allocation approach that can efficiently converge to a VM-to-PM mapping with best possible energy efficiency.
- To further accelerate the exploration of optimal VM allocation solutions, we develop an efficient simulation engine and integrate it into the cloud simulator CloudSim (Calheiros et al., 2011), which can drastically reduce the evaluation time of VM allocation solutions in each evolutionary iteration.

The rest of the paper is organized as follows. Section 2 presents related work on energy-aware VM allocation. After an introduction to the modeling and problem definition of energy-aware reservation-based VM allocation in Section 3, Section 4 details our evolutionary VM allocation approach. Section 5 evaluates our approach using both simulation-based and real-world examples. Finally, Section 6 concludes the paper.

2. Related work

Due to the proliferation of cloud computing and escalation of energy consumption and operational cost, sustainable computing has become a major concern of cloud service providers. To reduce the energy consumption, various energy-aware approaches have been studied (Pietri and Sakellariou, 2016; Berl et al., 2010; Beloglazov et al., 2012; Goudarzi and Pedram, 2016; Paya and Marinescu, 2017). For example, by formulating the VM consolidation problem as a multi-capacity stochastic bin packing problem, Hwang and Pedram (2013) proposed a hierarchical and scalable approach that can maximize the energy-efficiency considering both the correlation and resource type of VMs. Lee and Zomaya (2012) presented two energy-conscious task consolidation heuristics that can maximize resource utilization considering both active and idle energy consumption. Corradi et al. (2014) presented a cloud management platform which can optimize VM consolidation along three main dimensions (i.e., power consumption, host resources, and networking). Xu et al. (2016) introduced an energy consumption model for applications across cloud computing platforms and proposed an efficient approach that can conduct scientific workflow executions in an energy-aware manner. By formulating the VM placement problem as an integer programming problem, Dai et al. (2016) investigated multiple greedy approximation heuristics that can reduce the energy while satisfying the tenants' service level agreements.

As an alternative of application migration and consolidation, Dynamic Voltage and Frequency Scaling (DVFS)-based approaches are gaining more and more popularity in reducing energy consumption of data centers. For example, Calheiros and Buyya (2014) presented an approach for CPU-intensive Bag-of-Tasks applications on cloud infrastructures. Based on both intelligent scheduling and DVFS, the proposed method can keep the CPU operating at the minimum voltage level that enables the application to complete before a user-defined deadline. Although all the above approaches can effectively minimize the energy waste, none of them were designed for reservation-based clouds.

In order to offer better service price and quality to tenants, reservation-based computing paradigm is becoming popular among IaaS providers. To achieve best cost performance and reliability for reservation-based cloud services, various resource allocation optimization methods have been developed. For example, Wang et al. (2013) introduced a new cloud brokerage service that can optimally exploit both pricing benefits of long-term instance reservations and multiplexing gains. Based on dynamic programming and approximate algorithms, they proposed multiple dynamic strategies that can rapidly handle large volumes of demands. To provide a better Internet service by using cloud resources, Hwang et al. (2014) presented a two-phase algorithm for service operators that can make an effective trade-off between the amount of long-term reserved resources and that of on-demand subscribed resources, thus minimizing the service provision cost. Fuerst et al. (2016) designed a system prototype called KRAKEN. It allows tenants to dynamically modify their resource reservation at runtime through an online resource reservation scheme which comes with provable optimality guarantees. Zhang et al. (2016) developed a novel server consolidation algorithm. By reserving a certain amount of extra resources on each PM to avoid live migrations, their approach can cope with the heterogeneous burstiness during the server consolidation. Although all these approaches are promising in improving the quality and price of services, few of them took energy consumption into account.

Since most cloud scheduling and allocation problems are NP-hard, to efficiently achieve a near-optimal solution, a wide spectrum of AI-based heuristics (e.g., evolutionary algorithms Zhan et al., 2015, genetic algorithms) have been proposed. Based on genetic algorithm, Zhao et al. (2009) proposed an optimized algorithm that can schedule independent and divisible tasks in order to adapt to the memory constraints and high request of performance in cloud computing. Their approach can be applied to heterogeneous system with dynamic scheduling, where resources are of computational and communication heterogeneity. To simultaneously minimize total resource wastage and power consumption, Gao et al. (2013) developed a multi-objective ant colony system algorithm for the virtual machine placement problem, which can efficiently obtain a set of non-dominated solutions (i.e., the Pareto set). Experimental results show that their approach can improve both power efficiency and resource utilization in a cloud computing environment. Tsai et al. (2013) presented an improved differential evolution algorithm based on their proposed cost and time models on cloud computing environment, which can be used to optimize task scheduling and resource allocation and find the Pareto front of total cost and makespan. However, so far, few of existing approaches considered the efficient multi-objective optimization of reservation-based VM allocation problems.

Similar to our work, Hung et al. (2013) presented a genetic algorithm for the static VM allocation problem to minimize the energy consumption of private clouds and maximize the fulfilment of requirements of VMs. However, they assume that there are sufficient PMs for the computation and do not consider the rejection of VM requests which is common in practice. Moreover, the work in Hung et al. (2013) does not take the frequent updates of reserved VMs into consideration, while our approach is optimized to enable the quick search of optimal VM allocation solutions with highest instruction-energy ratio. To the best of our knowledge, our work is the first attempt that can optimize VM allocation for reservation-based clouds considering both energy consumption and service request acceptance ratio. Based on our proposed evolutionary heuristic, our approach can help IaaS providers to quickly search for a VM-to-PM mapping with near-optimal instruction-energy ratio, so as to minimize the overall operating cost.

3. System models and problem definition

We aim to maximize the energy efficiency (i.e., profit) of cloud data centers while minimizing the overall energy consumption with reserved VM requests. This section models the components of cloud data centers including PMs and tenant requests, and defines the problem that we are trying to solve.

3.1. Modeling of physical machines and VM requests

Consider a cloud data center that is composed of M PMs (i.e., servers), denoted by $S = \{S_1, S_2, \ldots, S_M\}$. The server S_i $(1 \le i \le M)$ is characterized by its clock frequency F_i, CPU utilization U_i, processing capacity C_i, and power consumption P_i. The processing capacity C_i of server S_i is measured and denoted by the number of Million Instructions Per Second (MIPS), which is proportional to the system clock frequency of S_i. For example, the processing capacity of HP ProLiant ML110 G4 server is 3720 MIPS when its CPU operates at a frequency of 1860 *MHz*, since the server completes two instructions per clock cycle on average (Calheiros et al., 2011).

In the reservation-based cloud system, tenants need to submit their VM requests to a resource reservation system to apply for cloud resources. Since all the VM requests in this paper are reservation based, we assume that they are computation-intensive and independent. In other words, our approach only takes the CPU utilization of VMs into account without considering the communications between VMs. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$ denote the set of N submitted VM requests during a reservation cycle. Each VM request τ_j $(1 \le j \le N)$ can be characterized using a triplet (a_j, e_j, c_j), where a_j indicates the request arrival time (i.e., VM start time), e_j denotes the request execution time, and c_j represents the desired processing resources (i.e., the CPU utilization) required by τ_j. When all the VM requests are collected, the broker of the resource reservation system will try to figure out whether there are enough PMs to accommodate all these requests. If available cloud resources cannot meet the requirement of a VM request, the request will be rejected by the reservation system.

Let $\mathcal{M}(i, j)$ denote a VM-to-PM mapping indicating whether the VM request τ_j is allocated to the server S_i, i.e.,

$$\mathcal{M}(i,j) = \begin{cases} 1 & \tau_j \text{ is allocated to } S_i \\ 0 & \text{otherwise} \end{cases}. \tag{1}$$

We use $W(\mathcal{M})$ to denote the total number of VM requests accepted by the cloud data center, where

$$W(\mathcal{M}) = \sum_{i=1}^{M} \sum_{j=1}^{N} \mathcal{M}(i,j). \tag{2}$$

As aforementioned, a VM request will be rejected by the reservation system if sufficient physical resources are not available to serve the request. We use the request acceptance ratio (RAR) to denote the proportion of VMs accepted by the resource reservation system, which is represented by

$$RAR(\mathcal{M}) = \frac{W(\mathcal{M})}{N}. \tag{3}$$

3.2. Modeling of energy consumption

Our approach adopts the linear interpolation power model proposed in Beloglazov and Buyya (2012) and Fan et al. (2007b) to approximate the overall server power consumption (including CPU, memory, disk, and etc.), which assumes that the power consumption P_i of server S_i is linearly correlated to the CPU utilization U_i. Let $U_i = \{U_{i,0}, U_{i,1}, \ldots, U_{i,h}\}$ be a set of $h+1$ discrete CPU utilization levels for the server S_i, where $U_{i,0} = 0\%$, $U_{i,h} = 100\%$[1], and $U_{i,j} < U_{i,j+1}$ $(0 \le j < h)$. We use the power consumption set $P_i = \{P_{i,0}, P_{i,1}, \ldots, P_{i,h}\}$ to denote the real measured power consumption of the server S_i with different CPU utilizations, where $P_{i,j}$ indicates the power consumption of the server S_i with a CPU utilization of $U_{i,j}$. Let $u_i(t)$ denote the CPU utilization of S_i at time t. We use $p_{i,j}(t)$ to represent the power of S_i at time t when $U_{i,j} < u_i(t) \le U_{i,j+1}$ $(0 \le j < h)$. It can be formulated as follows:

$$p_{i,j}(t) = \begin{cases} \frac{(u_i(t)-U_{i,j})\times(P_{i,j+1}-P_{i,j})}{(U_{i,j+1}-U_{i,j})} + P_{i,j} & U_{i,j} < u_i(t) \le U_{i,j+1} \\ 0 & otherwise \end{cases}. \tag{4}$$

Note that when $u_i(t)$ equals $U_{i,j}$ $(j \in \mathbb{N}^+)$, we can get $p_{i,j}(t) = P_{i,j}$. When j equals 0, we can get $p_{i,0}(t) = 0$. This is because that if there is no VM assigned to S_i, we can place it to the sleep mode, where the servers switch off unneeded subsystems and put the RAM into a minimum power state. For example, IBM is developing a mode named *deep-sleep* for its new Power processors that will allow them to consume almost no power when they are idle. Moreover, our approach neglects the on/off switching overhead of servers, since the overhead of switching modern processors from sleep mode to active mode is only about 300 ms (Meisner et al., 2009). Comparing with the long execution time of VMs, such overhead is negligible.

Assume that one reservation cycle needs a time of T, the energy consumption of the server S_i within a time interval [0, T] can be calculated using

$$\int_0^T \sum_{j=0}^{h-1} p_{i,j}(t)dt. \tag{5}$$

The total energy consumption of PMs can be calculated using

$$\sum_{i=1}^{M} \int_0^T \sum_{j=0}^{h-1} p_{i,j}(t)dt. \tag{6}$$

Since in the triplet of a VM request τ_k we assume that c_k is constant, we can partition the reservation cycle interval [0, T] into a sequence of Φ time segments in the form of $(t_\phi, t_{\phi+1}]$ $(\phi \in \{0, 1, \ldots, \Phi-1\})$, where the overall cloud power does not change in this segment. The overall energy consumed by the cloud PMs denoted by Eq. (6) can be approximated by

$$E(\mathcal{M}) = \sum_{i=1}^{M} \sum_{\phi=0}^{\Phi-1} \sum_{j=0}^{h-1} p_{i,j}(t_{\phi+1}) \times (t_{\phi+1} - t_\phi). \tag{7}$$

When calculating $E(\mathcal{M})$ using Eq. (7), instead of explicitly figuring out all the Φ time segments, we adopt the modified CloudSim with an event-driven engine (see details in Section 4.3) which can accurately simulate all the VM allocation updates.

To check the energy efficiency of a VM-to-PM mapping, we use the instruction-energy ratio (IER) to denote the average number of executed instructions for a given amount of energy, which is formulated as

$$IER(\mathcal{M}) = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N} \mathcal{M}(i,j) \times c_j \times e_j}{E(\mathcal{M})}. \tag{8}$$

3.3. Problem definition

In this paper, we consider the problem of assigning N reservation-based VMs to M PMs. We aim to generate a VM-to-PM mapping $\mathcal{M}$ with highest profit such that the overall energy consumption can be minimized and more VM requests can be served by the cloud data center. To achieve the highest profit, our approach tries to figure out a VM-to-PM mapping with maximized energy efficiency. The optimization problem can be formulated as:

$$\text{maximize} \quad \alpha \times \sum_{i=1}^{M} \sum_{j=1}^{N} c_j \times e_j \times \mathcal{M}(i,j) - \beta \times E(\mathcal{M}) \tag{9}$$

$$\text{subject to} \quad u_i(t) \le 100\% \quad \forall i \in \{1, \ldots, M\} \; \& \; t \in [0, T] \tag{10}$$

where α indicates the unit price of VM execution and β indicates the unit price of energy consumption of PMs. In other words, the formula presented in the optimization target indicates the profit earned by IaaS providers. Here we assume that the price of the VM request τ_j is proportional to the size of its workload (i.e., $c_j \times e_j$). To maximize the profit, IaaS providers need to consider two cases as follows:

1. When all the tenant requests can be accepted by the resource reservation system, only the overall energy consumption needs to be optimized.
2. When not all the tenant requests can be accepted by the resource reservation system, both the RAR and the overall energy consumption need to be considered.

For accurate estimation of energy consumptions, the above model involves extensive CloudSim simulations (e.g., for computing $E(\mathcal{M})$ in Eq. 7). This makes the classical heuristics for the bin packing problems (Martello and Toth, 1990) not applicable. Thus, we propose an evolutionary heuristic which can quickly find an optimized solution that makes a good tradeoff between the overall energy consumption and VM consolidation.

4. Our evolutionary approach

Aiming at maximizing energy efficiency and serving more VMs in reservation-based cloud data centers, we detail our evolutionary approach with an illustrative example. To reduce the evaluation time of VM-to-PM mappings, we developed a new simulation engine on top of CloudSim, which can be used to drastically reduce the evaluation time of VM-to-PM mappings.

4.1. An illustrative example

Recent studies have shown that the overall power consumed by PMs in cloud data centers can be accurately modeled by the linear relationship between CPU utilization and power consumption. Based on the observation from Beloglazov and Buyya (2012) and Fan et al. (2007b), we can find that even when a PM is running idle (i.e., $U_{i,0} = 0\%$) without sleeping, its power consumption (i.e., $P_{i,0}$) can still be quite large. Moreover, when running at full capacity (i.e., $U_{i,0} = 100\%$), the highest power (i.e., $P_{i,h}$) of a PM generally is less than two times of $P_{i,0}$. In other words, low CPU utilizations of PMs will lead to low energy efficiency, since $P_{i,0}$ cannot be apportioned by many VMs. Therefore, to save the overall power consumption, it is better to fully utilize the PMs. In addition to consolidating more VMs on a PM, the dynamic power management (DPM) method can be used to put PMs in sleep mode when they are idle. Since the sleep mode requires far less energy than $P_{i,0}$, the overall energy consumption of a cloud data center within a reservation cycle can be reduced.

To make the above observation clear, let us consider an example of allocating six VM requests (i.e., VM 1–6) to two PMs. Table 1(a) presents the VM settings of this example in terms of the triplet defined in Section 3.1. Table 1(b) shows the power step information

[1] $U_{i,h}$ is the CPU resource constraint, which means that the utilization rate of CPUs cannot be larger than 100%.

Table 1
VM and PM settings of the illustrative example.

(a) Settings of the six VM requests

VM requests	1	2	3	4	5	6
Arrival time	0	2	4	7	9	10
Execution time (in hours)	8	8	5	6	7	8
Processing capacity (MIPS)	1280	1300	1000	1020	960	1000

(b) Power consumption information of PMs (in Watts) Beloglazov and Buyya (2012)

CPU Utilization (%)	0	10	20	30	40	50	60	70	80	90	100
PM A (HP G4)	86	89.4	92.6	96	99.5	102	106	108	112	114	117
PM B (HP G5)	93.7	97	101	105	110	116	121	125	129	133	135

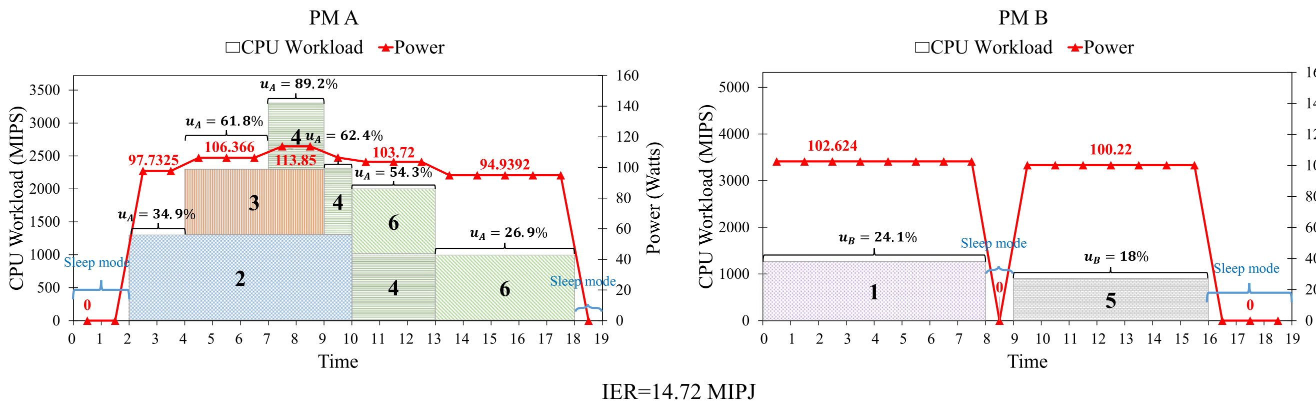


Fig. 2. An example of assigning six VMs to two PMs using MBFD approach (with an energy consumption of 3157.125 Watt-hours)

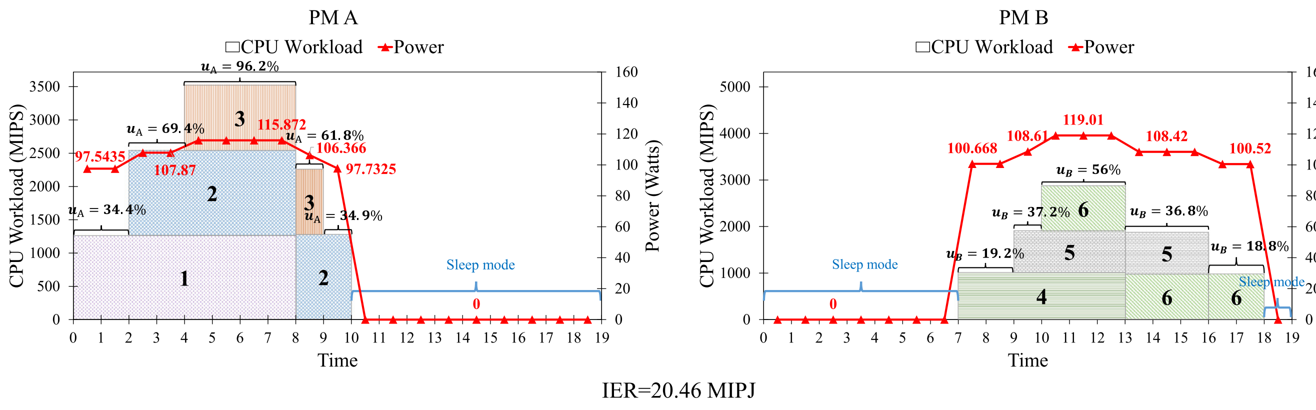


Fig. 3. An optimized VM allocation solution for the example presented in Fig. 2 (with an energy consumption of 2271.891 Watt-hours)

of the two PMs (HP ProLiant G4 and HP ProLiant G5) which are obtained from Beloglazov and Buyya (2012). Fig. 2 shows VM-to-PM allocation results using the Modified Best Fit Decreasing heuristic (MBFD) approach (Beloglazov et al., 2012). In this figure, we use u_A and u_B to denote the CPU utilization of the two PMs, respectively. We use red polylines to indicate the power consumptions of the PMs, which are calculated using Eq. (4). In this example, we adopt the DPM approach to put a PM to sleep mode when there is no VM running on it. In this case, the power consumption of the PM is 0. Based on the MBFD heuristic, we can complete all the VMs with an overall energy consumption of 3157.125 Watt-hours and an IER of 14.72 MIPJ.[2] From this figure, we can find that the CPU utilization of PM B is quite low (18%～24.1%). In this scenario, PM B requires around 100 Watts for the execution, while PM B in idle state needs a power of 93.7 Watts. Therefore, considerable energy is wasted by PM B due to the low CPU utilization and small power difference.

Fig. 3 shows the results of an optimized VM-to-PM allocation for the same problem as the one presented in Fig. 2. In this figure, VM 4 and VM 6 are moved from PM A to PM B, and VM 1 is moved from PM B to PM A. Based on this adjustment, we can find obvious improved CPU utilizations for both PMs A and B, e.g., from 61.8% to 96.2% within time interval [4,7] for PM A, and from 18% to 56% within time interval [10,13] for PM B. Moreover, we can find that PM A is put to sleep mode within the time interval [10,19], and PM B is put to sleep mode within the time interval [0,7], which are much longer than the ones shown in Fig. 2. Therefore, the VM-to-PM allocation solution in Fig. 3 is much better than the one presented in Fig. 2, since the VM execution only needs a total energy of 2271.891 Watt-hours with an IER of 20.46 MIPJ. From this example, we can find that even MBFD is a near-optimal VM allocation approach, we can still further exploit the energy potential for the VM allocation in reservation-based cloud data centers.

[2] MIPJ stands for Million Instructions Per Joule.

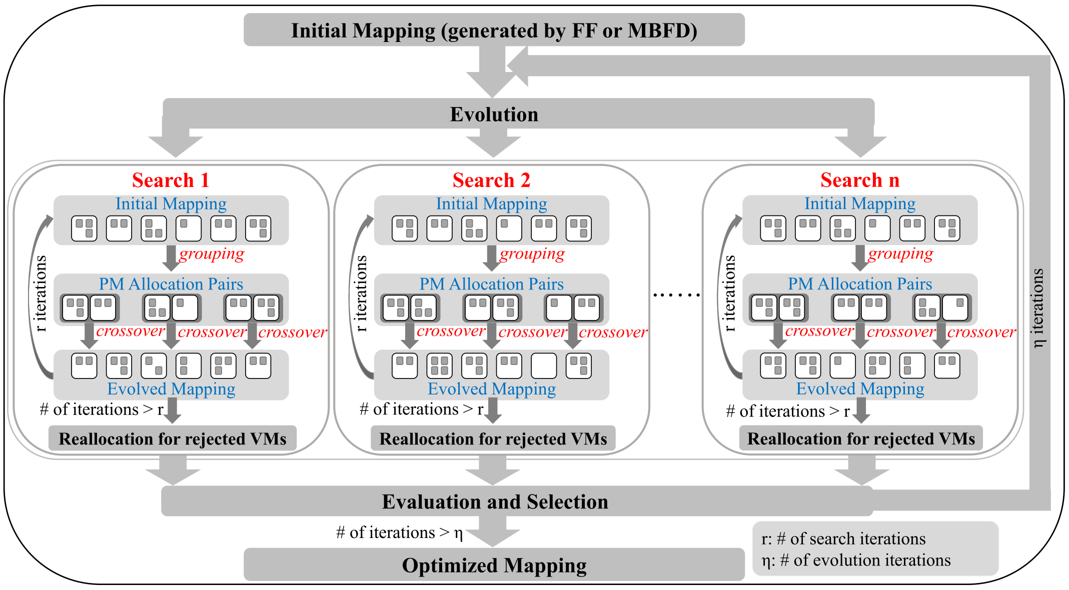


Fig. 4. The workflow of our evolutionary approach.

4.2. Our evolutionary VM allocation approach

Fig. 4 presents an overview of our VM-to-PM allocation approach. The input of our approach is an initial VM-to-PM mapping that can be obtained using the classical methods, e.g., First Fit heuristic (FF) and MBFD. Base on the initial VM-to-PM mapping, our approach iteratively searches for an optimal VM-to-PM mapping guided by our defined fitness function (see details in Section 4.2.1).

Our approach separates the optimal solution search into multiple evolutions where each evolution consists of n searches (i.e., search 1, ..., search n) to explore best possible VM-to-PM mappings. During the execution of an iteration within an evolution search, we firstly group PMs into random pairs for the following crossover operations. For the crossover operation, our approach tries to exchange VMs between PMs in the same pair, where the exchanged VMs have the smallest execution overlap with the other VMs on the same PM (see details in Section 4.2.3). After r internal iterations of a search, the evolved VM-to-PM mapping may improve the utilization of some PMs. If there exist rejected VMs due to the CPU resource constraints, at the end of a search our approach will try to accommodate more VMs by reallocating the rejected VMs to PMs in a First Fit manner. When all the n evolution searches finish, our approach will try to evaluate all the obtained n evolved VM-to-PM mappings and select the best one based on our defined fitness function as the new initial mapping for the next round of evolution iteration. Our approach will terminate when η evolutions complete, and the last survived mapping will be used as the optimized solution for the real VM deployment.

Algorithm 1 describes the workflow details of our approach, which has been illustrated in Fig. 4. Among the six inputs, the parameter $\mathcal{M}$ is an initial VM-to-PM mapping generated by existing heuristics (e.g., FF, MBFD). The parameters n, r, η are specified by IaaS providers indicating the number of evolution searches, search iterations, and evolution iterations, respectively. In this algorithm, after the initialization of evolution index in line 2, lines 3–17 conduct the evolutions iteratively. In each iteration, we use an array $\mathcal{ES}[n]$ to represent the n searches (lines 5–7) within an evolution, where $\mathcal{ES}[i]$ contains all the data structures used in the ith search. In the search iteration, line 6 uses *evolutionSearch* (see details in Algorithm 3) to search for a better solution than $\mathcal{M}$ using our proposed grouping and crossover methods. Note that the inputs of *evolutionSearch* is passed by value rather than by reference. After identifying the search which achieves the highest fitness value in lines 8–14, the obtained mapping will be used as the initial mapping for the next evolution iteration as indicated in line 15. Finally, line 18 reports an optimized VM-to-PM mapping. The following subsections detail the important steps of our approach.

Algorithm 1: Optimization of VM-to-PM Mappings.

```
Input: i) $\mathcal{M}$, an initial VM-to-PM mapping;
       ii) $S$, the set of PMs in the cloud data center;
       iii) $\mathcal{T}$, the set of VM requests submitted by tenants;
       iv) $n$, the number of searches in an evolution;
       v) $\eta$, the number of evolution iterations;
       vi) $r$, the number of iterations in a search;
Output: An optimized VM-to-PM mapping
1  genOptMapping($\mathcal{M}, S, \mathcal{T}, n, \eta, r$)begin
2      $iter = 0$;
3      while $iter < \eta$ do
4          initialize an array $\mathcal{ES}[n]$ representing $n$ searches;
5          for $\ell$ =0 to n-1 do
               // search for a better solution based on $\mathcal{M}$
6              $\mathcal{ES}[\ell].m = evolutionSearch(\mathcal{M}, S, \mathcal{T}, r)$;
7          end
8          $maxFitness = -1$, $sid = 0$;
9          for $\ell$=0 to n-1 do
               // select the best solution with the highest fitness
10             if $maxFitness < \mathcal{ES}[\ell].getFitness()$ then
11                 $maxFitness = \mathcal{ES}[\ell].getFitness()$;
12                 $sid = \ell$; // the evolution search ID
13             end
14         end
15         $\mathcal{M} = \mathcal{ES}[sid].m$; // update the mapping $\mathcal{M}$
16         $iter$++;
17     end
18     return $\mathcal{M}$;
19 end
```

4.2.1. Fitness function

Fitness function plays an important role in the evolutionary algorithm as it determines the quality of solutions in the evolution process. As described in Section 3.3, to achieve the highest profit as defined in Formula (9), IaaS providers expect to serve more VM requests while minimizing the overall energy consumption. Therefore, we take both the energy efficiency and request acceptance ratio into account when defining the fitness function of our approach. When PMs can accommodate all the VM requests (i.e., RAR equals to 100%), to achieve the optimal value for Formula (9), we only need to reduce the part of $\beta \times E(\mathcal{M})$, since the value of $\alpha \times \sum_{i=1}^{M}\sum_{j=1}^{N} c_j \times e_j \times \mathcal{M}(i,j)$ is fixed. Otherwise, we need to achieve a VM-to-PM mapping with both the highest PM workload and instruction-energy ratio. In our approach, we define the fitness function as

$$fitness(\mathcal{M}) = IER(\mathcal{M}) \times RAR^{\theta}(\mathcal{M}), \tag{11}$$

where $IER(\mathcal{M})$ is used to achieve a VM-to-PM mapping with optimal instruction-energy ratio, $RAR(\mathcal{M})$ is adopted to incorporate more VMs, and $\theta \in \mathbb{N}$ is used to tune the weight of request acceptance ratio during allocation solution evaluation. When all the VMs can be accommodated, the fitness function equals to $IER(\mathcal{M})$. When there exists space for rejected VMs, if we increase the value of θ, the chance of incorporating more VMs will increase. Note that $IER(\mathcal{M})$ and $RAR(\mathcal{M})$ are two conflicting factors, since $IER(\mathcal{M})$ prefers VM requests with large e_i and c_i but $RAR(\mathcal{M})$ prefers VM requests with small e_i and c_i.

4.2.2. Operations of PM grouping and VM crossover

In our approach, an evolution process consists of n searches to explore superior VM-to-PM mappings with higher fitness values via two operations, i.e., PM grouping and VM crossover.

Grouping: To achieve an optimized VM-to-PM mapping, one naive way is to enumerate all the M^N VM-to-PM mappings where M indicates the number of PMs and N denotes the number of VMs. However, this will be extremely time consuming and infeasible even for small-scale reservation-based cloud computing. Instead of exhaustively searching for the best solution in the whole state space, our approach focuses on local optimizations of small PM groups. We randomly divide the non-idle PMs into pairs, where proper optimization can be conducted to improve the fitness value by adjusting the VM allocations between the two PMs in the same pair. Note that due to the randomness of grouping operation, the evolution will not be stuck at local search, since different evolution searches have different optimization directions.

Crossover: Within an evolution search, we use the crossover operation to locally improve the fitness value of a PM pair. Fig. 5 presents an example of how our crossover operation works. In this example, the input PM pair p_i consists of two PMs (i.e., PM i and PM j), where a_1, a_2 indicate the allocations for the two PMs, respectively. Our crossover operation considers four scenarios based on three kinds of VM allocation actions: (i) *clone* that just duplicates the original PM allocation pair; (ii) *move* that migrates one VM from one PM to the other; and (iii) *exchange* that swaps two VMs between PM i and PM j. The reason why we adopt *clone* action in a crossover operation is because we do not expect a resultant pair p_r which has a worse fitness value than the original PM pair p_i. All the four newly-generated PM allocation pairs are stored in an array $p[\,]$, and the pair in $p[\,]$ with the highest fitness value will be selected as an optimized result obtained by the crossover operation. Note that both the actions *move* and *exchange* may result in the violations of CPU resource constraints. Thus the crossover operation may reject the VMs which cannot be accommodated in their destination PMs. For example, when adopting the

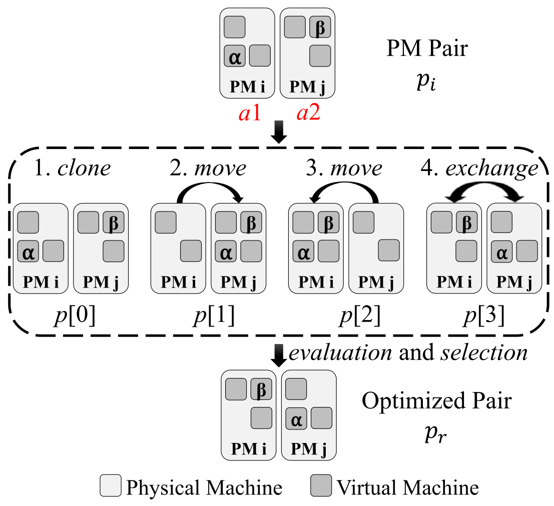


Fig. 5. An example of crossover operation.

exchange action in a crossover operation, if the VM α cannot be accepted by its host PM j, it will be labelled as a rejected VM.

When conducting a crossover operation, it is important to identify which VMs need to be processed between pairwise PMs. To effectively increase the fitness value of the optimized allocation pair, we consider the time overlapping of VMs within a PM allocation. Based on the example shown in Table 1 and Fig. 3, we can observe that the more VMs are stacked at the same time to get high CPU utilization, the better IER the PMs can achieve. In our approach, we try to move the VM that has the minimum accumulative overlapping time (AOT) with all the other VMs in the same PM to its counterpart PM in the same pair. This is because if such a VM is withdrawn from its host PM, the PM may have a higher chance to have more time in the sleep mode. Consequently, the overall energy of the paired PMs can be drastically reduced.

Assume that there are n VM requests (i.e., $\tau_{S_x} = \{\tau_1, \ldots, \tau_n\}$) allocated to a PM S_x. The accumulative overlapping time of τ_i with all the other VMs in the PM can be calculated using

$$AOT(\tau_{S_x}, \tau_i) = \sum_{j=1, j\neq i}^{n} length([a_j, a_j + e_j] \cap [a_i, a_i + e_i]), \tag{12}$$

where a_i and e_i denote the arrival time and execution time of τ_i respectively, and $length(a,\ b)$ indicates the intersection length of two time intervals a and b. In our approach, the VM with the smallest AOT will be selected as the candidate VM for the crossover operation. As an example shown in Fig. 2 that is derived from Table 1, PM A and PM B are grouped into a pair for energy improvement. Since $AOT(\tau_{PM_A}, \tau_2) = 8$, $AOT(\tau_{PM_A}, \tau_3) = 7$, $AOT(\tau_{PM_A}, \tau_4) = 8$ and $AOT(\tau_{PM_A}, \tau_6) = 3$, our approach will select VM 6 from PM A for the exchange action. Similarly, our approach will choose VM 1 from PM B for the crossover operation, since it is the first VM that has the smallest AOT value (i.e., 0) in PM B.

Fig. 6 shows the optimized allocation for the PM pair after one crossover operation. In this case, the first move action succeeds among all the four kinds of actions, since it has the best fitness value. From Fig. 6, we can observe that although the power of PM B is increased by around 8.2 Watts within the time interval [10,16] and 100.52 Watts within the time interval [16,18], PM A is switched to sleep mode within time interval [13,18]. Due to the longer time staying in sleep node, the optimized allocation has a better IER value (i.e., 15.99 MIPJ) than the allocation solution shown in Fig. 2. Note that when we consider the move action that sends VM 1 from PM B to PM A, the crossover operation will reject

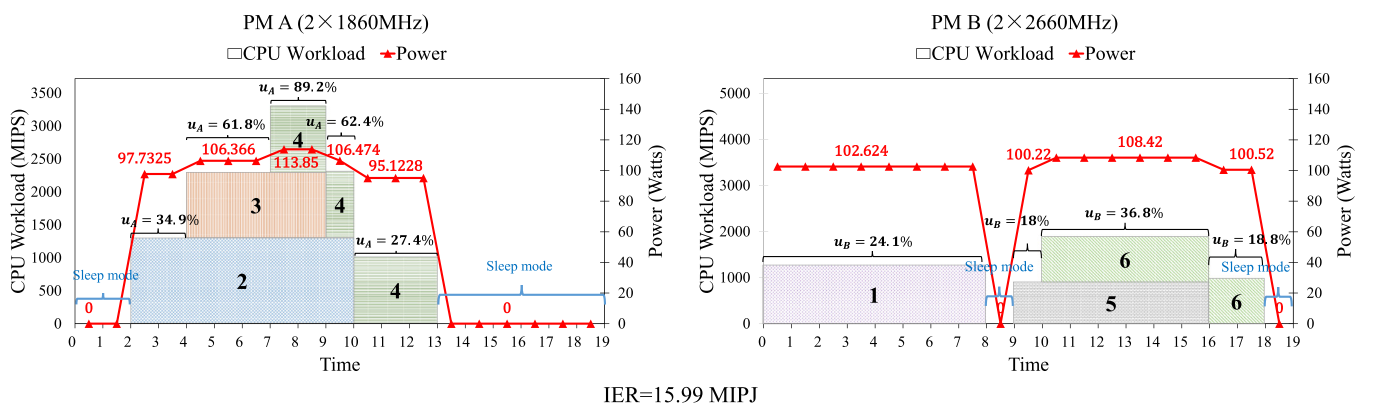


Fig. 6. Optimized allocation (with an energy consumption of 2804.25 Watt-hours) for the PM pair shown in Fig. 2 using one crossover operation.

the VM, since PM *A* cannot accommodate VM 1 due to the CPU resource limit within the time interval [7,9]. In this case, both RAR and fitness value will be reduced.

Algorithm 2 presents the details of our crossover operation,

Algorithm 2: Crossover Operation.

```
Input: i) $\mathcal{M}'$, an input VM-to-PM mapping;
       ii) $p_i$, a PM pair for crossover operation;
Output: An optimized mapping based on crossover on $p_i$
1  crossover($\mathcal{M}'$, $p_i$)begin
2      $(\alpha, \beta) = p_i.getMinAOT()$;
3      $p[0] = p_i.clone()$;
4      $p[1] = p_i.move1(\alpha)$;
5      $p[2] = p_i.move2(\beta)$;
6      $p[3] = p_i.exchange(\alpha, \beta)$;
       // select the best pair with the highest fitness value
7      $maxFitness = -1$;
8      for $i = 0$ to 3 do
9          if $maxFitness < p[i].getFitness()$ then
10             $maxFitness = p[i].getFitness()$;
11             $p_r = p[i]$;
12         end
13     end
14     return $\mathcal{M}'.update(p_r)$;
15 end
```

which takes a VM-to-PM mapping $\mathcal{M}'$ and a PM allocation pair p_i as inputs. In this algorithm, line 2 obtains two VMs (i.e., α, β) with the minimum AOT from p_i. Lines 3–6 consider the four actions of a crossover operation. All the four enumerated scenarios are saved in *p*[ ], which is an array of VM-to-PM mappings. Note that during the execution of an action, α or β can be rejected due to the CPU resource constraint. Lines 7–13 try to figure out which solution has the highest fitness value. Finally, line 14 returns a new VM-to-PM mapping with the best fitness value.

4.2.3. Implementation of evolutionary search

Unlike the crossover operation which only aims at improving the allocations of one PM pair, the objective of an evolution search is to improve the allocation on all PMs. Algorithm 3 presents the details of the evolution search. In this algorithm, lines 2–10 iteratively search for better VM-to-PM mappings with our proposed crossover operation. In each search iteration, line 4 figures out the non-idle PMs based on the current mapping information. Line 5 randomly divides the non-idle PMs and groups them into pairs. For each pair, we apply one crossover operation (lines 6–8) to obtain

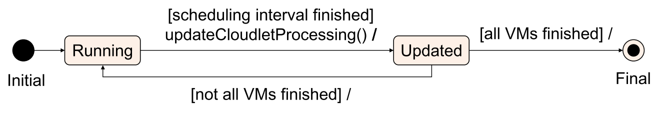
Algorithm 3: Evolution Search.

```
Input: i) $m$, an input VM-to-PM mapping;
       ii) $S$, the set of PMs;
       iii) $\mathcal{T}$, the set of VM requests;
       iv) $r$, the number of search iterations;
Output: An evolved mapping $m$
1  evolutionSearch($m, S, \mathcal{T}, r$)begin
2      $iter = 0$;
3      while $iter < r$ do
4          $S' = S - idle(S, m)$;
5          $G = grouping(m, S')$;
6          for $i = 0$ to G.size()-1 do
7              $m$ =crossover($m, G[i]$);
8          end
9          $iter$++;
10     end
11     $(\mathcal{T}_a, \mathcal{T}_r) = classifyVM(m, \mathcal{T})$;
12     if $\mathcal{T}_r \neq \emptyset$ then
13         $\mathcal{T}_r' = sortVM(\mathcal{T}_r)$;
14         $m = reallocate(m, S, \mathcal{T}_r')$;
15     end
16     return $m$;
17 end
```

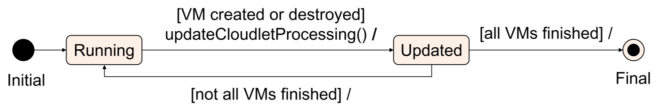
a locally optimized VM-to-PM mapping. Note that when conducting a crossover operation, the selected VMs with minimum AOT may be rejected by their target PMs. By performing crossover operations, our approach may squeeze more space out of the given paired PMs, which can be used to accommodate the unallocated VMs. Therefore, at the end of an evolution search, our approach tries to reallocate such VMs to PMs to achieve higher fitness value. In line 11, we use the function *classifyVM*$(m, \mathcal{T})$ to figure out the allocated and rejected VMs which are denoted by two sets $\mathcal{T}_a$ and $\mathcal{T}_r$, respectively. If $\mathcal{T}_r$ is not empty, line 13 will first sort the VM in $\mathcal{T}_r$ based on their execution time lengths in an ascending order. Then, line 14 will try to allocate the sorted VMs in $\mathcal{T}_r$ to PMs in a First Fit manner. Finally, line 16 returns a globally optimized VM-to-PM mapping derived from an evolution search.

4.3. Simulation time reduction

Although our evolutionary approach can derive energy-efficient VM-to-PM mappings, it is very time-consuming for complex VM scheduling scenarios, since it requires a large quantity of repeated simulation-based evaluation operations (i.e., *getFitness()* function)

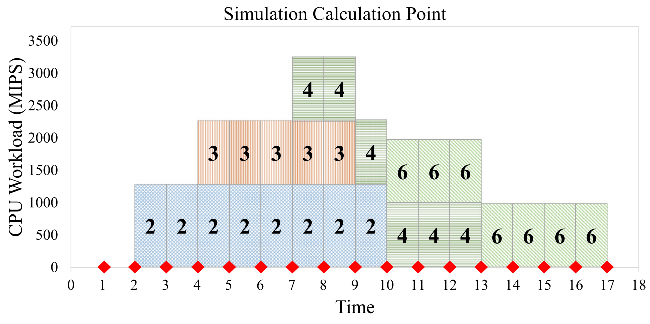


(a) Original version of CloudSim

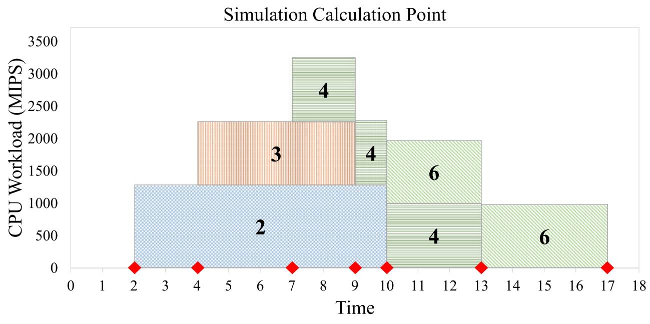


(b) Modified version of CloudSim

Fig. 7. State machine diagrams for different versions of CloudSim.



(a) Simulation execution using original CloudSim.



(b) Simulation execution using modified CloudSim.

Fig. 8. Original CloudSim versus modified CloudSim.

in the evolution. We use the simulator CloudSim (Calheiros et al., 2011) for the evaluation of cloud scheduling. Equipped with a time-driven engine, CloudSim periodically updates the data center statistics (e.g., energy consumption, the number of executed instructions) with the advance of scheduling intervals.[3] Consequently, if allocated VMs have long execution time, the evaluation time spent by *getFitness()* functions will be extremely long because of a large quantity of update operations performed by CloudSim. Fig. 7(a) shows the implementation of the time-driven CloudSim in the form of a state machine diagram. The *Running* state denotes that the normal execution of the cloud data center. When an interrupt denoted by the predicate *[scheduling interval finished]* is triggered, the function *updateCloudletProcessing()* will be invoked to update the data center statistics. The state *Updated* is an instant state that indicates the completion of updates. In other words, only the *Running* state can advance the scheduling intervals.

Since our approach focuses on the energy consumption, when there are no VM allocation change on a PM, calculating the updates of data center statistics in each scheduling interval is a waste of time. This is because both the consumed energy and the number of executed instructions grow linearly. To reduce the simulation time, we propose a coarse-grained engine for CloudSim as shown in Fig. 7(b). Unlike the time-driven approach, our approach only updates the power and other statistics of the data center based on the events of VM allocation changes. For example, when a VM is created on some PM, proper update will be conducted to calculate the new data center statistics. Since such events happen much less frequently than the scheduling interval-based events, the overall evaluation time can be drastically reduced.

Fig. 8 presents a comparison of simulation executions between the original CloudSim and our modified CloudSim. In this example, we only show the simulation of the VM executions on a PM. Since the executions on different VMs are independent, we can have the same conclusion for the simulation executions of the other PMs in a data center. From Fig. 8(a), we can find that if we consider all the scheduling intervals, we need to calculate 17 updates. However, by using our simplified engine, our modified CloudSim only needs to calculate 7 updates as shown in Fig. 8(b), which is much more time-efficient than the one shown in Fig. 8(a). Note that although our modified CloudSim focuses on quickly calculating the overall energy consumption of generated VM-to-PM mappings, it can be easily extended to deal with other event-driven issues.

5. Performance evaluation

To validate the performance of our approach, we conducted experiments on both the simulation-based platform CloudSim (Calheiros et al., 2011) and a real cloud environment. We compared our approach with two well-known baseline methods (FF and MBFD Beloglazov et al., 2012) from the perspectives of overall profit, energy consumption, instruction-energy ratio, and request acceptance ratio.

5.1. Simulation-based experiment

Since it is too expensive and time-consuming to conduct experiments on real cloud computing platforms, to carry out large scale experiments, we adopted the cloud simulator CloudSim (Calheiros et al., 2011) which supports various virtualized hardware resources and VM provisioning techniques. We modified the CloudSim and integrated our simplified simulation engine, which can accelerate the simulation process. When using our approach as presented in Algorithm 1, we set the inputs n, η, r to 10, 5, 10, respectively. All the simulation results are obtained from both the original CloudSim and modified CloudSim on a desktop equipped with 3.10 GHz Intel Core i5 CPU and 10 GB RAM.

5.1.1. Modeling of physical machines

This experiment simulates a data center consisting of 30 PMs. Among these PMs, 15 of them are IBM x3250 servers where each server equips with an Intel Xeon X3470 CPU that has four cores with a frequency of 2933 MHz. Similar to Beloglazov and Buyya (2012), we model a multi-core PM with n cores each having m MIPS as a single-core PM with the total capacity of $n \times m$ MIPS. Therefore, if each core only issues one instruction per clock cycle, the overall throughput of an IBM x3250 server will be $2933 \times 4 = 11732$ MIPS. For the remaining 15 PMs, we adopted IBM x3550 servers where each server has two CPUs and each CPU consists of six cores with a frequency of 3067 MHz. In other words, each IBM x3550 server has a capacity of $2 \times 6 \times 3067 = 36804$ MIPS. Note that Cloudsim has built-in power models in the form of Eq. (4) for both IBM x3250 and x3550 servers. To accurately approximate the real power of such servers, Cloudsim provides the mea-

[3] Scheduling interval is used as a time unit during the execution of CloudSim. By default, one scheduling interval represents one second of the physical time during the simulation.

Table 2
Power (in Watts) settings of the two IBM servers.

CPU Utilization (%)	0	10	20	30	40	50	60	70	80	90	100
IBM x3250	41.6	46.7	52.3	57.9	65.4	73	80.7	89.5	99.6	105	113
IBM x3550	58.4	98	109	118	128	140	153	170	189	205	222

sured system power (see Table 2) for 11 uniformly distributed CPU utilization levels from 0% to 100% with an interval of 10%. Based on Eq. (4), we can figure out the power of the servers when their CPU utilization rate is in between these intervals. For example, when CPU utilization rates reach 20% and 30%, the power of an IBM x3250 server is 52.3 Watts and 57.9 Watts, respectively. When the CPU utilization of an IBM x3250 server is 24%, according to Eq. (4) we can figure out that the server power is $\frac{24\%-20\%}{30\%-20\%} \times (57.9 - 52.3) + 52.3 = 54.54$ Watts.

5.1.2. Modeling of virtual machine reservations

To objectively evaluate the effectiveness of our approach, we conducted the simulation on the PMs modelled in Section 5.1.1 with various VM requests made by tenants. We investigated the arrival time, capacity, and execution time of VMs in the experiment, since they are significant factors that strongly affect the performance of cloud systems. Similar to the work in Li et al. (2014), in this experiment we generated a large set of stochastic VM requests for the simulation. Let N be the total number of VM requests arrived within a day. We used the Poisson distribution to model the arrival time of VMs where $\lambda = N/24$ indicating the average number of arrived VM requests per hour. To model VMs with different processing capacities, we adopted the normal distribution $N(\mu, \sigma^2)$ where μ indicates the mean processing capacity (i.e., MIPS) of VMs and σ denotes the standard deviation of VM processing capacities. We used the uniform distribution of $[e_{\min}, e_{\max}]$ to model the duration of VM executions, where $e_{\min}$ and $e_{\max}$ indicate the minimum and maximum execution time of the generated VMs, respectively.

5.1.3. Baseline approaches and evaluation metrics

In this experiment, we compared our approach with two state-of-the-art VM allocation heuristics as follows:

- **First Fit Algorithm (FF)**. From a list of sorted PMs, FF algorithm always assigns a given VM to the first PM that meets its CPU resource requirement.
- **Modified Best Fit Decreasing Algorithm (MBFD)** (Beloglazov et al., 2012). MBFD approach schedules VMs in deceasing order of their CPU resource demands. When allocating a VM, MBFD always selects a PM that both meets the VM's CPU resource requirement and leads to the least increase of overall power consumption.

Note that the above two heuristics can be used to generate initial VM-to-PM mappings (i.e., the input $\mathcal{M}$ of Algorithm 1) for our approach. In other words, our approach can be considered as an extension of the above two approaches, which can further improve the overall energy consumption of reserved VMs. We use the terms FF-EX and MBFD-EX to denote our methods based on the initial mappings generated by FF and MBFD, respectively. To show the effectiveness of our approach, we considered the following four performance metrics:

- **Overall Profit**. As an important metric for energy efficiency, overall profit denotes the total profit achieved by a cloud data center considering both the VM workload and energy consumption, which is formulated in Eq. (9).

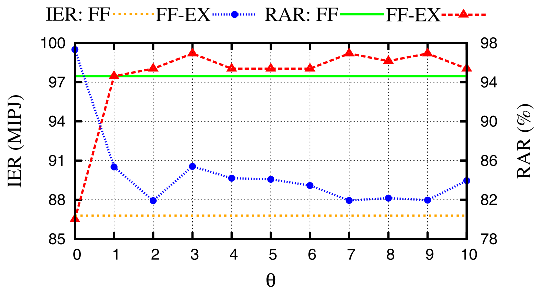


Fig. 9. IER and RAR trends for different θ using FF and FF-EX.

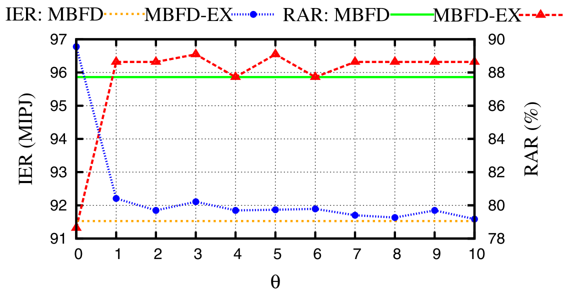


Fig. 10. IER and RAR trends for different θ using MBFD and MBFD-EX.

- **Request Acceptance Ratio**. As an important SLA metric, RAR equals the percentage of VM requests accepted by the cloud resource reservation system over all the requests submitted by tenants.
- **Energy Consumption**. It indicates the total amount of energy consumed by PMs to execute all the accepted virtual machines.
- **Simulation time**. It indicates the wall time of executing a given VM-to-PM mapping on a simulated cloud platform.

As described in Section 4.2.1, IER and RAR are two conflicting factors which strongly affect the profit of reservation-based cloud data centers. Therefore, in the fitness function we use the parameter θ to make the tradeoff between IER and RAR. Typically, a larger θ can lead to higher acceptance ratio during the evolution iterations. Figs. 9 and 10 show the impacts of the different θ on both IER and RAR for both FF/FF-EX methods and MBFD/MBFD-EX methods, respectively. In Fig. 9, we consider a set of 130 VMs whose processing capabilities following the normal distributions $N(5000, 1000^2)$. In Fig. 10, we consider a set of 220 VMs whose processing capabilities following the normal distributions $N(3000, 1000^2)$. From these figures we can find that when θ equals to 0 both FF-EX and MBFD-EX can achieve their highest IERs and lowest RARs, since the fitness function only considers the effect of IER. Note that FF and MBFD approaches do not consider the impacts of θ during their executions. When $\theta \geq 1$, though the IERs of the VM allocation solutions generated by FF-EX and MBFD-EX drop, they are still better than their counterpart IERs of the VM allocation solutions generated by FF and MBFD. Although when $\theta \geq 1$ our approaches (FF-EX and MBFD-EX) can obtain better IERs and RARs than their counterparts (FF and MBFD), empirically we prefer to set θ to 1, 2 or 3. In the following experiments we set θ to 3, since it shows the best performance in both Figs. 9 and 10.

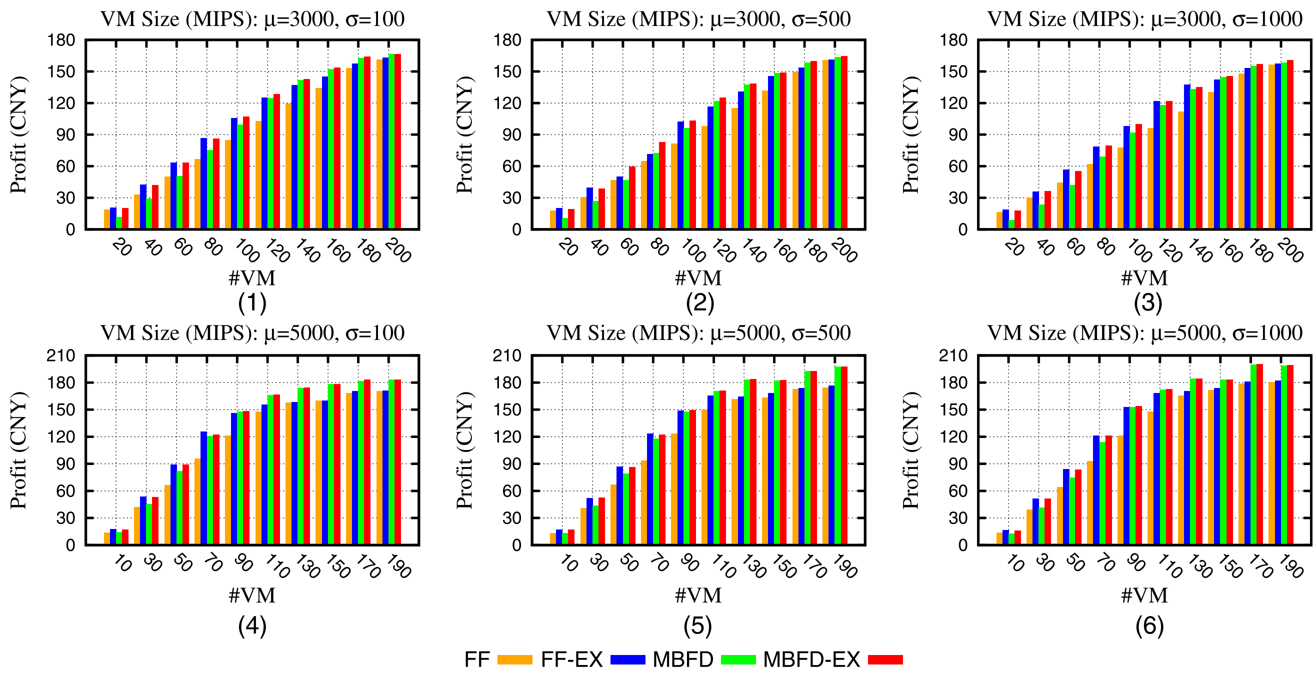


Fig. 11. A comparison of overall profit for the four VM allocation methods.

5.1.4. *Experiments for energy efficiency and consumption*

Fig. 11 presents a comparison between our approaches (i.e., FF-EX and MBFD-EX) and their counterparts in terms of overall profit within one reservation cycle (i.e., one day). In this comparison, we investigated reserved VMs with different settings including: (i) the number of VMs, (ii) VM arrival time that follows the Poisson distribution, (iii) VM processing capabilities that follow the normal distributions $N(3000, 100^2)$, $N(3000, 500^2)$, $N(3000, 1000^2)$, $N(5000, 100^2)$, $N(5000, 500^2)$, $N(5000, 1000^2)$, respectively, and (iv) the duration of VM executions that follows the uniform distribution where $e_{\min} = 14$ h and $e_{\max} = 16$ h. For each subfigure of Fig. 11, we evaluated a series of VM sets that are generated following the same distributions of VM processing capabilities (i.e., VM size) and VM execution durations (i.e., VM lengths). Note that within a subfigure different VM sets have different distributions for VM arrival time, since the size of these VM sets is different. Based on the pricing information provided by Alibaba Cloud (ECS, 2017), we set α to 9×10^{-9} which indicates that executing one million instructions needs 9×10^{-9} Chinese Yuan (CNY). We set β to 1.2, which denotes the average industrial electricity price (1.2 CNY per kilowatt-hour) in China.

From these six subfigures, we can find that our approaches (MBFD-EX and FF-EX) outperform their counterparts (MBFD and FF), and our MBFD-EX approach can always achieve the highest profits. As an example shown in Fig. 11(6), when a set of 90 reserved VM requests are submitted to the cloud data center, our FF-EX approach can achieve 32.1 CNY more than FF method, which accounts for 21% of FF's overall profit. In Fig. 11(3), when there are a set of 60 reserved VM requests investigated, our MBFD-EX approach can achieve 24% profit improvement over the MBFD method. Since we only have 30 PMs, when the size of VM set increases, some VM requests will be rejected. For example, in Fig. 11(1)-(3) the approaches start to reject VM requests when the size of VM set is larger than 180. In Fig. 11(4)-(6), VM rejection starts when the size of VM set is larger than 110. Since all the PMs are fully-loaded, the profit differences are negligible in these cases.

Fig. 12 presents the results of energy consumption using the same VM sets generated in Fig. 11. To fairly conduct the comparison, we do not consider the cases involving VM rejections in this figure. From this figure, we can find that our approaches can achieve significant energy savings compared with their counterparts. For example, in Fig. 12(3), when there are 60 VM requests in total for allocation, our MBFD-EX method can achieve 38% energy savings compared with MBFD method. Moreover, in Fig. 12(6), our FF-EX method can achieve 41% energy savings compared with its counterpart when there are 70 VM requests used in reservation.

5.1.5. *Experiment for request acceptance ratio*

To show the effectiveness of our approach from the perspective of request acceptance ratio, we adopted the same VM sets generated in Section 5.1.4 for the evaluation. In Fig. 13, we only investigated the scenarios where the PMs do not have enough space to host all the VM requests. From this figure, we can find that our approaches can accept more VM requests than their counterparts. As an example shown in Fig. 13(3), when there are 200 VM requests reserved in total, our FF-EX method can achieve more than 1.5% improvement of request acceptance ratio compared with FF method. From all the other five subfigures, we can also observe the superiority of our approaches. The reason of this trend is mainly because that during the evolutionary search iterations (see details in Algorithm 3) our approach always tries to find an optimal VM allocation solution with the highest instruction-energy ratio. Based on our proposed fitness function, VM moves guided by our evolutionary approach can explore more spare resources than existing methods, which can accommodate more extra VM requests.

5.1.6. *Experiment for simulation time*

Our approach uses CloudSim as the underlying simulator to evaluate the performance of VM allocation solutions generated in each evolutionary iteration. Since CloudSim needs to compute the specified metrics (e.g., energy consumption, number of executed instructions) in each scheduling interval, for complex VM scheduling solutions it requires a long simulation time. To reduce the overall evaluation time, we adopt the simplified simulation engine in CloudSim as described in Section 4.3, which only calculates performance metrics when necessary.

To show the efficacy of our modified simulation engine, Table 3 compares the overall simulation time (with 500 evolutionary it-

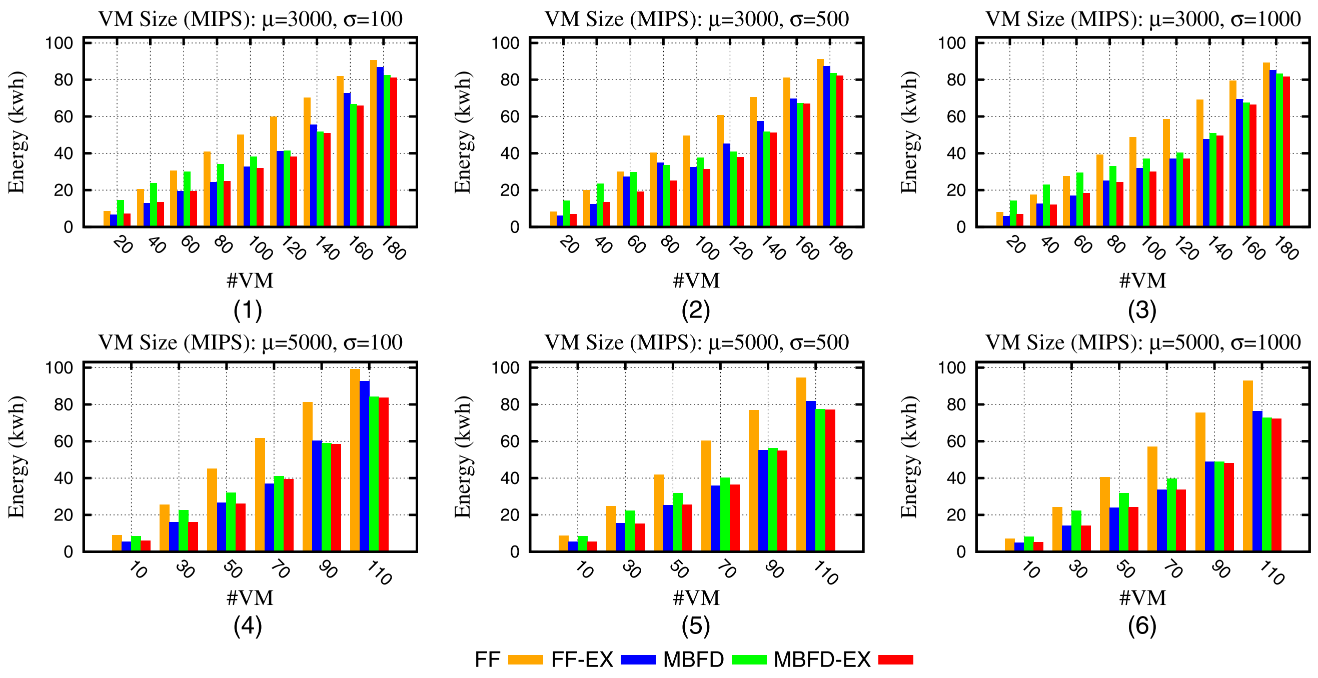


Fig. 12. A comparison of energy consumption for the four VM allocation methods.

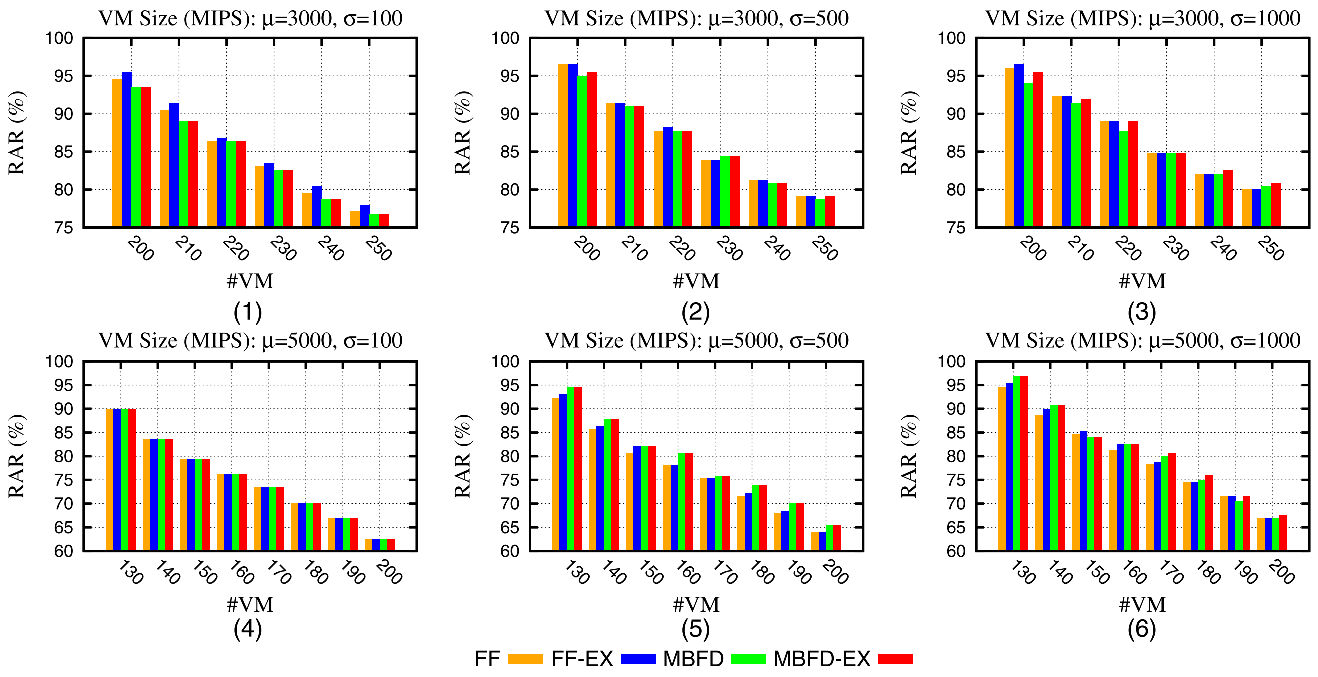


Fig. 13. A comparison of request acceptance ratio for the four VM allocation methods

Table 3
Simulation time (in minutes) of MBFD-EX approach: original CloudSim versus modified CloudSim.

#VM	*Original CloudSim*	*Modified CloudSim*	*Speedup*
60	1024.2	3.0	341
70	1048.1	3.4	308
80	1105.6	3.6	307
90	1172.8	4.4	266
100	1209.1	5.2	232

Table 4
CPU capacities of computing nodes in our testbed.

Compute Node Type	*CPU Capacity (in MIPS)*
Server with Xeon CPU	2400 × 8
Desktop with Intel Core i5 CPU	3093 × 4

erations) of MBFD-EX approach using both original CloudSim and modified CloudSim. Due to the limitation of space, here we only consider the case where the settings are the same as the ones used in Fig. 11(3). From Table 3, we can find that our modified CloudSim can improve the simulation time by several orders of magnitude. For example, when there are 60 VMs involved in the allocation, our modified CloudSim can achieve a speedup of 341 times over

Table 5
Measured power (in Watts) of servers used in the real-world experiment.

CPU Utilization(%)	0	10	20	30	40	50	60	70	80	90	100
Server with Xeon CPU	81.41	87.49	92.41	97.09	104.25	116.71	125.13	127.70	129.64	131.89	132.30
Server with Core i5 CPU	43.59	44.15	44.82	45.78	46.84	49.07	53.00	55.30	59.51	64.93	71.07

Fig. 14. Part of our OpenStack-based experiment environment.

Table 6
A comparison of energy consumption (in joules) for both experiments.

	FF	FF-EX	Energy saving rate
Real-world	273360	260114	4.84%
Simulation	294192	285804	2.85%
	MBFD	MBFD-EX	Energy saving rate
Real-world	361569	264449	26.8%
Simulation	394092	286092	27.4%

the original CloudSim. Note that for the other VM allocation approaches (i.e., FF, FF-EX, MBFD) with different VM settings, we can observe the similar trend.

5.2. Real-World experiment

For the evaluation and further validation of our approach in real-world cloud computing environment, we constructed an OpenStack-based (version 2014.1.5) testbed (part of which are shown in Fig. 14) that consists of one controller node and six compute nodes. These nodes use the Ubuntu operating system (version LTS 14.04).

The controller node adopted in our test bed is a desktop equipped with 4 GB RAM and an AMD FX-6100 CPU with six 3.3 GHz cores. We used two kinds of PMs as compute nodes in our testbed. Among the six compute nodes, one of them is a server equipped with 32 GB RAM and a Xeon CPU with 8 cores (16 threads) where each core has a frequency of 2.0 GHz. The remaining compute nodes are desktops, each of which has 10 GB RAM and an Intel Core i5 CPU with four 3.10 GHz cores. Table 4 shows the computing capacities of these two kinds of servers.

To obtain real-time power of PMs, we adopted the HP9800 power monitor (shown in the rightmost side of Fig. 14) produced by Shenzhen HOPI electronic technology limited company. We developed a Python program that periodically (every 1 s) reads the power information from the USB port of the power monitor, which can be used to calculate the overall energy consumed by the PMs. Since the power varies during the execution of PMs in real-world environment, all the experiments under this setting were carried out 5 times and we used the mean value for the performance comparison. Table 5 shows the measured real power information of compute nodes with different levels of CPU workload. To validate the effectiveness of our approach, we also use such information and Eq. (4) to approximate the power consumption in the simulation.

In this real-world experiment, we investigated the allocation of eighty VM requests on the six compute nodes, where all the VMs adopted the same operating system (i.e., Ubuntu Linux LTS 14.04). We developed a program written in C programming language which can control the MIPS usage of its host VM and its execution duration based on user inputs. Similar to the VM generation in Section 5.1.2, the arrival time of these VMs follows the Poisson distribution ($\lambda = 3.3$). The processing capacities of these VMs follow the uniform distribution where the maximum value is 1200 MIPS and the minimum value is 250 MIPS. The durations of the VM executions follow the uniform distribution where $e_{\max} = 648$ s and $e_{\min} = 432$ s. Note that in this experiment we assumed that all the VMs are computation-intensive and all the VM executions are independent (i.e., no communications between tasks).

Table 6 presents the energy consumption results for both real-world and CloudSim-based experiments. Note that for both experiments the VM allocation solutions used for energy consumption evaluation were obtained by using the four approaches (FF, FF-EX, MBFD, and MBFD-EX) based on simulations. For each of the VM allocation solutions generated by FF-EX and MBFD-EX, we conducted 500 evolution search iterations (with $n = 10,\ \eta = 5,\ r = 10$) to achieve a near-optimal result, which cost around 4 min. In this experiment, we do not consider the scenarios of VM rejection. In other words, all the VMs can be accommodated within the six PMs. From this table, we can find that our FF-EX and MBFD-EX approaches can achieve much better energy consumption as compared with their counterparts. For example, when comparing with MBFD method, our MBFD-EX approach can reduce the energy consumption by up to 26.8% in the real-world experiment. Although the results of simulation are not the same as the results of the real-world experiment, we can observe the same trend of energy savings by using our approaches.

6. Conclusions and future work

Service reservation is gaining popularity in cloud computing, since it can not only facilitate the management of increasing tenants' requests, but also achieve lower price compared with traditional pay-as-you-go cloud services. However, when more and more virtual machines are deployed in data centers, the utilization rate of host PMs is becoming much more challenging to control. To address this problem, we proposed an evolutionary approach that can effectively allocate and consolidate VMs among heterogeneous PMs. By fully exploring the capacity of reservation-based clouds, our approach can optimize the instruction-energy ratio of PMs to save their consumed energy. Moreover, to enable fast evaluation of VM allocation solutions during the search of optimal results, we developed a coarse-grained simulation engine and incorporated it into CloudSim. Experimental results show that our approach outperforms existing VM allocation methods from the perspectives of both energy efficiency and request acceptance ratio.

In the future, we plan to incorporate the Dynamic Voltage and Frequency Scaling (DVFS) technique into our approach to further maximize the energy efficiency for reserved cloud service requests. Due to the varying processing capacities of heterogeneous PMs along with the change of CPU frequencies, achieving an optimal VM schedule considering DVFS is more difficult. Meanwhile, the reliability of VM executions is strongly affected by the supplying voltages of host PMs. Within a DVFS-enabled cloud data center, how to optimize the energy consumption of reserved VM requests while satisfying specific reliability requirements is also an interesting topic that is worthy of further study.

References

Amazon Elastic Compute Cloud, 2016. Amazon elastic compute cloud reserved instances pricing. https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/?nc1=h_ls.

Barroso, L.A., Clidaras, J., Holzle, U., 2013. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan and Claypool Publishers.

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

Beloglazov, A., Buyya, R., 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concur. Comput Pract Exp. 24 (13), 1397–1420.

Berl, A., Gelenbe, E., Girolamo, M.D., Giuliani, G., Meer, H.D., Dang, M.Q., Pentikousis, K., 2010. Energy-efficient cloud computing. Comput. J. 53 (7), 1045–1051.

Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I., 2009. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. 25 (6), 599–616.

Calheiros, R.N., Buyya, R., 2014. Energy-efficient scheduling of urgent bag-of-tasks applications in clouds through DVFS. In: Proceedings of International Conference on Cloud Computing Technology and Science, pp. 342–349.

Calheiros, R.N., Ranjan, R., Beloglazov, A., Rose, C.A.F.D., Buyya, R., 2011. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exper. 41 (4), 23–50.

Chen, M., Huang, S., Fu, X., Liu, X., He, J., 2018. Statistical model checking-based evaluation and optimization for cloud workflow resource allocation. IEEE Transactions on Cloud Computing, accepted.

Corradi, A., Fanelli, M., Foschini, L., 2014. VM Consolidation: a real case based on openstack cloud. Future Gener. Comput. Syst. 32, 118–127.

Dai, X., Wang, J.M., Bensaou, B., 2016. Energy-efficient virtual machines scheduling in multi-tenant data centers. IEEE Trans. Cloud Comput. 4 (2), 210–221.

Fan, X., Weber, W., Barroso, L.A., 2007a. Power provisioning for a warehouse-sized computer. In: Proceedings of International Symposium on Computer Architecture (ISCA), pp. 13–23.

Fan, X., Weber, W.D., Barroso, L.A., 2007b. Power Provisioning for a warehouse-sized computer. In: Proceedings of International Symposium on Computer Architecture (ISCA), pp. 13–23.

Fuerst, C., Schmid, S., Suresh, L., Costa, P., 2016. Kraken: online and elastic resource reservations for multi-tenant datacenters. In: Proceedings of International Conference on Computer Communications (INFOCOM). 1–9

Gao, Y., Guan, H., Qi, Z., Hou, Y., Liu, L., 2013. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J. Comput. Syst. Sci. 79 (8), 1230–1242.

Goudarzi, H., Pedram, M., 2016. Hierarchical SLA-driven resource management for peak power-aware and energy-efficient operation of a cloud datacenter. IEEE Trans. Cloud Comput. 4 (2), 222–236.

Hung, N.Q., Nien, P.D., Nam, N.H., Tuong, N.H., Thoai, N., 2013. A genetic algorithm for power-aware virtual machine allocation in private cloud. In: Proceedings of International Conference on Information and Communication Technology, pp. 183–191.

Hwang, I., Pedram, M., 2013. Hierarchical virtual machine consolidation in a cloud computing system. In: Proceedings of International Conference on Cloud Computing (CLOUD), pp. 196–203.

Hwang, R.H., Lee., C.N., Chen, Y.R., Zhang-Jian, D.J., 2014. Cost optimization of elasticity cloud resource subscription policy. IEEE Trans. Serv. Comput. 7 (4), 561–574.

Kim, K.H., Beloglazov, A., Buyya, R., 2011. Power-aware provisioning of virtual machines for real-time cloud services. Concur. Comput. Pract. Exper. 23 (13), 1491–1505.

Lee, Y.C., Zomaya, A.Y., 2012. Energy efficient utilization of resources in cloud computing systems. J. Supercomput. 60 (2), 268–280.

Li, K., Tang, X., Li, K., 2014. Energy-efficient stochastic task scheduling on heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. 25 (11), 2867–2876.

Martello, S., Toth, P., 1990. Knapsack Problems: Algorithms and Computer Implementations. Wiley-Interscience.

Meisner, D., Gold, B., Wenisch, T., 2009. Powernap: eliminating server idle power. In: Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 205–216.

Paya, A., Marinescu, D.C., 2017. Energy-aware load balancing and application scaling for the cloud ecosystem. IEEE Trans. Cloud Comput. 5 (1), 15–27.

Pietri, I., Sakellariou, R., 2016. Mapping virtual machines onto physical machines in cloud computing: a survey. ACM Comput. Surv. 49 (49), 1–30.

Pricing of Alibaba Cloud ECS, 2017. https://www.aliyun.com/price/product#/ecs/detail.

Tsai, J., Fang, J., Chou, J., 2013. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. J. Comput. Operat. Res. 40 (12), 3045–3055.

Wajid, U., Cappiello, C., Plebani, P., Pernici, B., Mehandjiev, N., Vitali, M., Gienger, M., Kavoussanakis, K., Margery, D., Garcia-Perez, D., Sampaio, P., 2016. On achieving energy efficiency and reducing CO_2 footprint in cloud computing. IEEE Trans. Cloud Comput. 4 (2), 138–151.

Wang, W., Niu, D., Li, B., Liang, B., 2013. Dynamic cloud resource reservation via cloud brokerage. In: Proceedings of International Conference on Distributed Computing Systems (ICDCS). 400–409

Xu, X., Dou, W., Zhang, X., Chen, J., 2016. Enreal: an energy-aware resource allocation method for scientific workflow executions in cloud environment. IEEE Trans. Cloud Comput. 4 (2), 166–179.

Zhan, Z., Liu, X., Gong, Y., Zhang, J., Chung, H., Li, Y., 2015. Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Comput. Surv. 47 (4), 1–33.

Zhang, S., Qian, Z., Luo, Z., Wu, J., Lu, S., 2016. Burstiness-aware resource reservation for server consolidation in computing clouds. IEEE Trans. Parallel Distrib. Syst. (TPDS) 27 (4), 964–977.

Zhao, C., Zhang, S., Liu, Q., Xie, J., Hu, J., 2009. Independent tasks scheduling based on genetic algorithm in cloud computing. In: Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing (WiCom), pp. 1–4.

Xinqian Zhang received the B.E. degree from the Computer Science and Software Engineering Institute of East China Normal University in 2015. He is currently a Ph.D. student in the Computer Science and Software Engineering Institute, East China Normal University, Shanghai, China. His research interests are in the area of cloud computing, parallel and distributed systems, design automation of parallel computing systems, and software engineering.

Tingming Wu received the B.E. degree from the Software Engineering Institute, East China Normal University, Shanghai, China, in 2015. He is currently a master student in the Institute of Computer Science and Software Engineering, East China Normal University. His research interests are in the area of cloud computing, parallel and distributed systems, design automation of embedded systems, and software engineering.

Mingsong Chen received the B.S. and M.E. degrees from Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006 respectively, and the Ph.D. degree in Computer Engineering from the University of Florida, Gainesville, in 2010. He is currently a Professor with the Computer Science and Software Engineering Institute at East China Normal University. His research interests are in the area of design automation of cyber-physical systems, cloud computing, parallel and distributed systems, and formal verification techniques. He is an Associate Editor of IET Computers & Digital Techniques, and Journal of Circuits, Systems and Computers.

Tongquan Wei received his Ph.D. degree in Electrical Engineering from Michigan Technological University in 2009. He is currently an Associate Professor in the Department of Computer Science and Technology at the East China Normal University. His research interests are in the areas of green and reliable embedded computing, cyber-physical systems, parallel and distributed systems, and cloud computing. He serves as a Regional Editor for Journal of Circuits, Systems, and Computers since 2012. He also served as Guest Editors for several special sections of IEEE TII and ACM TECS.

Junlong Zhou received his Ph.D. degree in Computer Science from East China Normal University in 2017. He is currently an Assistant Professor with the Department of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing, China. He was a Research Visitor with the University of Notre Dame, Notre Dame, IN, USA. His current research interests include real-time embedded systems, cloud computing, and cyber-physical systems. Dr. Zhou has been an Associate Editor for the Journal of Circuits, Systems, and Computers since 2017, and a Guest Editor for the ACM Transactions on Cyber-Physical Systems.

Shiyan Hu received his Ph.D. in Computer Engineering from Texas A&M University in 2008. He is an Associate Professor at Michigan Tech, and he was a Visiting Associate Professor at Stanford University from 2015 to 2016. His research interests include Cyber-Physical Systems (CPS), CPS Security, Data Analytics, and Computer-Aided Design of VLSI Circuits, where he has published more than 100 refereed papers. He is an ACM Distinguished Speaker, an IEEE Systems Council Distinguished Lecturer, an IEEE Computer Society Distinguished Visitor, and a recipient of National Science Foundation (NSF) CAREER Award. Prof. Hu is the Chair for IEEE Technical Committee on Cyber-Physical Systems. He is the Editor-In-Chief of IET Cyber-Physical Systems: Theory&Applications. He is an Associate Editor for IEEE Transactions on Computer-Aided Design, IEEE Transactions on Industrial Informatics, and IEEE Transactions on Circuits and Systems. He has held chair positions in numerous IEEE/ACM conferences. He is a Fellow of IET and a senior member of the IEEE.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He served as a Future Fellow of the Australian Research Council during 2012–2016. He has authored over 625 publications and seven text books. He is one of the highly cited authors in computer science and software engineering worldwide. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and five IEEE/ACM conferences. He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience. Dr. Buyya is recognized as a “2016 Web of Science Highly Cited Researcher” by Thomson Reuters, a Scopus Researcher of the year 2017 with Excellence in Innovative Research Award by Elsevier, and a Fellow of IEEE for his outstanding contributions to Cloud computing.