IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 August 2022, accepted 2 September 2022, date of publication 14 September 2022,
date of current version 22 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3206021

RESEARCH ARTICLE

Defending Against Co-Residence Attack in Energy-Efficient Cloud: An Optimization Based Real-Time Secure VM Allocation Strategy

LU CAO[1], RUIWEN LI[1], XIAOJUN RUAN[2], AND YUHONG LIU[1], (Senior Member, IEEE)
[1]Department of Computer Engineering, Santa Clara University, Santa Clara, CA 95053, USA
[2]Department of Computer Science, California State University, East Bay, CA 94542, USA

Corresponding author: Yuhong Liu (yhliu@scu.edu)

ABSTRACT Resource sharing among users serves as the foundation of cloud computing, which, however, may also cause vulnerabilities to diverse co-residence attacks launched by malicious virtual machines (VM) residing in the same physical server with the victim VMs. In this paper, we aim to defend against such co-residence attacks through a secure, workload-balanced, and energy-efficient VM allocation strategy. Specifically, we model the problem as an optimization problem by quantifying and minimizing three key factors: (1) the security risks, (2) the power consumption and (3) the unbalanced workloads among different physical servers. Furthermore, this work considers a realistic environmental setting by assuming a random number of VMs from different users arriving at random timings, which requires the optimization solution to be continuously evolving. As the optimization problem is NP-hard, we propose to first cluster VMs in time windows, and further adopt the Ant Colony Optimization (ACO) algorithm to identify the optimal allocation strategy for each time window. Comprehensive experimental results based on real world cloud traces validate the effectiveness of the proposed scheme.

INDEX TERMS Computer security, cloud computing, co-residence attack, ant colony optimization.

I. INTRODUCTION

Cloud computing has become popular in both business and personal services. Infrastructure as a Service (IaaS) in cloud computing is a service model that grants multiple users' access to a shared pool of physical resources in a dynamic way. Such resource sharing allows the cloud to maximize the system efficiency by fully utilizing available computing resources. On the other hand, cloud users can dramatically save costs by paying only for the resources that they are using and releasing the idle resources to other users. These advantages attract numerous businesses that want to reduce costs on intensive computational operations.

However, such infrastructure-level computing resource sharing, which is enabled through multi-tenancy (defined as ''the practice of placing multiple tenants on the same physical hardware'' [48]), also introduces new security risks. Attackers taking advantage of the co-residence opportunities may perform diverse attacks against their co-tenants [1], [2], [3], [8], [16], [20], [33], [40], [45], [49], threaten the security of cloud infrastructure and undermine users' confidence to move to the cloud [9], [39], [46], [47]. For example, a misconfigured hypervisor which hosts multiple Virtual Machines (VM) from different tenants may serve as a conduit for information leakage [11]. Chiang proposed Swiper attack with which the attacker uses a carefully designed workload to incur significant delays to the targeted co-resident application [8]. Ristenpart and Swift proposed an attack which modifies the workload of a victim VM in a way that frees up resources for the attacker's VM [49]. Particularly, such co-residence attacks have two unique characteristics: First, it is directly enabled by the resource sharing among different users, and will continuously exist unless users are isolated on different Physical Machines (PM). Second, it mainly leverages the legitimate resource requests. Therefore, conventional security techniques, such as authentication, authorization and

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

access control, can hardly detect and block co-residence attacks without preventing normal access to the shared resources [49].

There are a number of solutions proposed to defend against co-residence attacks through performance isolation which requires virtualized computing resource isolation for storage, CPU, cache, memory, and access path networks [30], [53]. However, such solutions are typically either impractical (e.g., high overhead or nonstandard hardware), application-specific, or insufficient for fully mitigating the risk. Furthermore, it requires that the resources can never be overcommitted due to the possibility of concurrent requests from multiple tenants. This requirement will inevitably leave resources idle and sacrifice cloud performance and efficiency. Due to the immaturity of virtualization technology and the absence of physical isolation, smart adversaries are still able to launch attacks that penetrate the virtual boundaries among tenants [8], [23], [24], [29], [35], [55]. At the current state of the art, there is no practical way to guarantee the unconditional security except avoiding multi-tenancy [40].

Recently, a few studies have been proposed to focus on secure VM allocation strategies, which assign VMs to available physical machines (PMs) in a secured way to prevent malicious users from achieving co-residence with normal users [4], [21], [22]. Compared to performance isolation approaches, this type of mechanisms does not require significant changes of the existing hardware/software, and is not limited to specific applications. Nevertheless, this line of research has just been initiated recently and has very limited amount of work. In addition, as the number of possible allocations increases in a factorial way when the number of available PMs/VMs becomes large, it has been verified as an NP-hard problem to search for the best allocation [21], [22]. Most of current studies resolve this issue only through heuristic solutions.

Therefore, a secure and energy-efficient VM allocation strategy to defend against the co-residence attacks is proposed in this paper. The main contributions of this research are summarized as follows.

- First, we propose to consider and quantify three key factors for secure VM allocation in energy-efficient cloud: (1) the security risks introduced by the co-residence of VMs from multiple users, (2) the overall power consumption and (3) the workload inequality among different PMs. The VM allocation problem is then modeled as an optimization problem where the objective function is to minimize these three factors at the same time.
- Second, this work assumes a realistic scenario where a random number of VMs from different users may arrive at the cloud end with random timings, which requires the optimization solution to be dynamically evolving based on both the existing allocation status and new allocation requests.
- Third, as this optimization problem is NP-hard [22], we aim to address the problem by balancing the optimization goal, the computational complexity and the allocation delay. Specifically, we propose to first introduce time windows to handle arriving VMs in clusters. Then for each time window, the Ant Colony Optimization (ACO) algorithm, an evolutionary algorithm inspired by natural ant activities, is adopted to identify the optimal allocation strategy for new VMs based on the prior VM allocation status. Although ACO has already been applied to address diverse optimization problems, we are the first one to adopt it in the secure cloud resource allocation scenario. Comprehensive understanding and analysis on the physical meanings of (1) the ACO algorithm and (2) the cloud secure VM allocation scenario have been performed to facilitate such adoption.
- Fourth, comprehensive experiments based on real-world cloud workload traces are conducted to study (1) the impact of critical parameter settings; (2) the effectiveness of the proposed scheme when compared to the state-of-the-art secure VM allocation studies.

II. BACKGROUND AND RELATED WORK

A. PERFORMANCE ISOLATION

Diverse studies have been conducted to prevent sensitive information from being transferred through converted channels (i.e. side channels) between co-resident VMs at different levels of cloud infrastructure. First, eliminating side channels from hardware level [25], [30], [52] usually provides more effective defense. However, due to the complex process of introducing new hardware into existing cloud infrastructure, the adoption of such schemes adds extra cost on hardware and administration. Second, extensive researches have been carried out at the hypervisor level. For example, XenPump proposed as a module located in hypervisor [53], monitors the hypercalls used by timing channels and adds latency to potential malicious operations, which increases the error rate in timing channels. In addition, Shacham *et al.* proposed to make the timer substantially more coarse by removing resolution clocks on Xen-virtualized x86 machines, so that malicious VMs can hardly obtain accurate time measurement [50]. The key drawback of these schemes is that they often require significant modifications on hypervisors. Third, some schemes are proposed at VM OS level [54] or application level [10]. For instance, the authors in [51] proposed to hide real power consumption information from user VMs by deploying a police VM to generate false information. Such schemes do not require substantial changes in the cloud infrastructure and are thus easy to be adopted. Nevertheless, they often suffer from the heavy overhead caused by obfuscating side channel information at the upper level of the cloud infrastructure.

B. VIRTUAL MACHINE ALLOCATION

Attackers who aim to launch co-residence attacks against a certain target have to first place their malicious VMs on the same physical host where the target VM locates.

Co-residence attacks cannot succeed if this first step fails. Therefore, researches are launched to design security aware VM allocation policies which significantly increase the difficulties for attackers to achieve co-residence.

Many VM allocation policies are studied to assign different positions to VMs. For instance, a randomization way to assign VMs has been proposed [4] to make VMs' deployment unpredictable to attackers. Han *et al.* have proposed a co-resident attack resistant VM allocation policy [22], which distributes VMs by optimizing security, workload balance and power consumption needs of cloud servers. Li and Zhang *et al.* have designed a Vickrey-Clarke-Groves (VCG) mechanism to migrate VMs periodically, so that malicious VMs cannot stay co-located with their target VM for a long time even if they can achieve co-residence [27]. Chhabra *et al.* proposed an allocation policy to reduces the probability of co-residence by classifying legal VMs and attacker VMs based on historical data, similar approaches often require significant computational analysis and previous knowledge on each incoming request [7], which can be further improved.

C. ENERGY-EFFICIENT CLOUD COMPUTING

Besides security, energy-efficient cloud computing has recently attracted great attention as data centers consume a large amount of electricity and generate giant power bills every year at companies like Google, Facebook, Amazon, etc. Data centers consumed more than 2% of the US total electricity consumption [19]. Different energy-efficient solutions have been applied at ventilation, liquid-cooling systems, and building construction [36]. However, such construction level modification will generate a large amount of cost. Furthermore, cooling systems will also consume a significant amount of electricity. Without conducting hardware level modification, a power-aware VM scheduling algorithm could significantly reduce energy consumption with minimum financial cost and little performance impact. Recent research shows that VM scheduling algorithms have great impact on overall energy consumption of a data center [6]. Therefore, energy-efficiency is used as an important factor for our scheduling algorithm to evaluate the overall performance and efficiency.

D. ANT COLONY ALGORITHM AND ITS APPLICATIONS

The Ant Colony Optimization (ACO) is a meta-heuristic algorithm for finding optimized solutions of computational problems. It is inspired by one behavior of ants, in which they leave pheromone on favorable paths for other members to follow [12]. ACO has been applied to a wide range of optimization problems which are mostly NP-hard. With the initial application to the Traveling Salesman Problem (TSP) [13], ACO has also been applied to solve other problems like sequential order problem (SOP) [18], vehicle routing problem [5], [17], resource constraint project scheduling problem [31].

In cloud computing, ant colony optimization is widely used in task scheduling [37], [38]. Li proposed a Load Balancing Ant Colony Optimization (LBACO) algorithm to achieve task scheduling in dynamic cloud system while in consideration of load balancing at the same time [26]. Feller has applied ACO in workload placement and the results show that this approach provides superior energy efficiency [15]. Similar applications can also be seen in [34] and [32] where ACO has been adopted to address cloud scheduling tasks. However, they do not consider the security aspect.

To the best of our knowledge, this is the first work to apply ACO to address the secure VM allocation issue in cloud. Based on its high efficiency and effectiveness in addressing NP-hard problems, we believe ACO is an appropriate tool to allocate cloud VMs so that the cloud's overall security, power consumption and workload balance are optimized.

E. OUR EARLIER WORK

In [28], which is the conference version of the work, we proposed the optimized energy-efficient and security-aware VM allocation strategy against co-residence attack. The preliminary results indicated that the presented research is able to achieve the balance among cloud security, energy-efficiency, and workload balance. The journal version is significantly different from our conference version in the following aspects. **First**, from the model aspect, rather than assuming all VMs arriving at the same time, this work considers a more realistic real-time scenario as a random number of VMs from different users arriving at the cloud with random timings, which requires the solution to dynamically evolve according to the existing VM allocation status and the incoming new VM requests. **Second**, from the solution aspect, to balance computational complexity, real time delay and the optimization results, we first introduce time windows to handle VMs in clusters and then apply ACO algorithm for each time window to manage VM allocation. A more in-depth understanding of the ACO algorithm, how and why it is mapped to address the proposed problem have been discussed in a more comprehensive way, which well explained the fundamental working mechanisms of the proposed scheme. **Third**, as a proof of concept, the conference version only provided basic performance evaluations. More sophisticated experiments and data analysis based on real world cloud workload traces have been conducted in this journal draft. Each of the key parameters of the proposed scheme has been tested and discussed. Additional state-of-the-art comparison scheme has been implemented and compared with the proposed scheme. The results are discussed in details. **Last but not least**, more comprehensive reviews and analysis of the state-of-the-art literature have been conducted.

III. MODELING

In this section, we will present the proposed secure VM allocation strategy in details. In particular, we would like to first discuss the system model and assumptions; then model the secure allocation issue as an optimization problem; and present how to adopt ACO algorithm to solve the optimization problem in an efficient way.

A. ASSUMPTIONS

As one of the first few works to systematically model the secure and energy-efficient VM allocation problem at IaaS level in cloud, we propose to make the following assumptions to facilitate the establishment of the optimization model later.

First, we assume the cloud receives a random number of VMs from different users at random timings. Periodically, the cloud needs to assign n_v^t VMs from n_u^t users arriving in the time duration t to a number of available PMs, so that the VM assignment can minimize security risks, overall power consumption and imbalance of workload among PMs. How frequently the cloud should perform such assignment can be determined to balance time delay, computational complexity and the optimal solutions.

Second, for each time window t, the number of PMs involved in the allocation, marked as n_s^t, is not given. As we assume that there are sufficient number of idle PMs to host VMs, n_s^t should be a value within the range $[n_{s\_min}^t, n_{s\_max}^t]$. In particular, the minimum number of PMs, $n_{s\_min}^t$, is achieved when all the VMs are squeezed into the minimal number of PMs to make the utilization as high as possible. On the other hand, the maximum number of PMs, $n_{s\_max}^t$, is achieved when each VM is assigned to a different PM. In other words, $n_{s\_max}^t = n_v^t$. This allocation achieves maximum security since all VMs are isolated on different PMs at the cost of highest power consumption and workload imbalance.

Third, we assume each VM's workload is dynamically changing during run time based on the real world cloud workload traces. Please note that such changes will lead to fluctuations of the power consumption and workload balance, and may occasionally cause overload of PMs which triggers dynamic VM migrations among PMs in cloud. These above assumptions make our model more realistic but also more challenging to address.

Fourth, regarding the security aspect, we assume that all VMs from a malicious user are malicious. The attack goal is to have the malicious VMs achieve co-residence with VMs from as many normal users as possible to facilitate later attacks. In addition, from the defender side, we also assume that according to historical data, the cloud is able to estimate the percentage of malicious users, but does not know which specific users are malicious. This assumption requires that the proposed scheme can develop the best allocation strategy based on different security context. In the case where the cloud does not have a good estimation of the malicious user percentage, this value can always be set as 100% to treat security in the most conservative way, which will result in an allocation solution that minimizes the co-locations of VMs from different users.

B. OPTIMIZATION MODEL

With the above assumptions, we model the overall VM allocation problem as an optimization problem, of which the optimization goal is to minimize (1) the security risks, which is modeled as the probability of malicious VMs co-locating with the VMs from normal users (i.e. R_{sec}), (2) the overall power consumption of used PMs to run these VMs (i.e. $f_{Power}(u)$), and (3) the workload inequality among different PMs (i.e. B_w). Therefore, we design the objective function of this optimization problem as follows.

$$c = w_S * R_{sec} + w_P * f_{power}(u) + w_W * B_w \tag{1}$$

where c represents the objective function value or cost value, w_S, w_P, and w_W represent weights for the three factors respectively.

Then the next question is how to quantify these three factors in a reasonable way. In this work, we propose the quantification of these factors as follows.

1) QUANTIFICATION OF SECURITY

Specifically, as we assume a uniform distribution of malicious users, we model the probability of malicious VMs co-locating with normal users (i.e. the security risks) as

$$R_{sec} = P_{mal} * \frac{\sum_{i=1}^{n_s}(n_{co-loc}^i - 1)}{n_s * (n_u - 1)} \tag{2}$$

where P_{mal} indicates the estimated malicious user percentage; n_s and n_u represent the number of PMs and users, respectively; and n_{co-loc}^i indicates the number of co-located users at PM i. Since we consider a PM with only one user as secure, a "-1" is introduced in both the numerator and the denominator parts for normalization purpose. We can see that (1) in the ideal case, where each PM hosts no more than one user's VMs, R_{sec} is 0; (2) in the worst case, where each PM hosts VMs from all the users, R_{sec} is 1; and (3) R_{sec} will increase when either the percentage of malicious users or the number of co-located users at each PM increases.

2) QUANTIFICATION OF POWER CONSUMPTION

The power cost evaluation is based on the power measurement of a PM at eleven CPU utilization levels at 0%, 10%, 20% ... 100% [41] since only CPU utilization is sufficient to evaluation whole computing system's power cost [14]. Since measuring power cost at all utilization levels are neither cost-effective nor practical, we use linear interpolation (Eq. 3) to estimate the corresponding power cost that is at an unmeasured utilization level U by using the measured power costs P_h and P_l at the higher utilization U_h and lower utilization U_l.

$$f_{Power}(u) = \frac{P_h - P_l}{U_h - U_l}U - \frac{P_h U_l - P_l U_h}{U_h - U_l} \tag{3}$$

The power cost is normalized as

$$P_{normalized} = \frac{\sum_{i=1}^{n_s}|P_i - P_{best}|}{P_{best} \times n_s} \tag{4}$$

where P_i represents the current power cost of the ith PM and P_{best} represents the most effective power cost [41] that has the highest performance to power ratio. We can see that the greater the difference between P_i and P_{best}, the less power efficient the current server is.

3) QUANTIFICATION OF WORKLOAD INEQUALITY

At the end, the cost of workload inequality is normalized as

$$B_w = \frac{1}{n_s}\sqrt{\sum_{i=1}^{n_s}(wl_i - \overline{wl})^2} \tag{5}$$

where wl_i represents the workload of VM i; and $\overline{wl}$ represents the average workload for all the n_s PMs. We can see that either an extremely large or extremely small workload will dramatically increase the cost of the workload inequality.

C. DYNAMIC VMs IN REAL TIME

As proved in other existing studies [21], [22], we recognize the optimization issue modeled in the above section as an NP-complete problem. However, the problem is even more complex as in reality, cloud servers continuously receive dynamic VM requests and the workload of existing VMs is also dynamically changing.

To make the model more realistic, we assume that users' requests arrive at the cloud end with random timings. Furthermore, the request from each user may be realized through a random number of VMs. In particular, we 1) adopt Poisson distribution to simulate the incoming VMs' arrival rate; 2) introduce time window concept to group incoming requests to balance optimal assignment solution, computational complexity and allocation delay; 3) use real world cloud traces for VM workload simulations.

1) ARRIVAL TIMING

We adopt Poisson distribution, a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time, to simulate the arrival time of VMs. Furthermore, the number of VMs arrives in one time interval does not affect that in any other time intervals.

In particular, VM arrival rate λ is used to tune the workload by varying the inter arrival time based on Poisson Distribution as presented in the following equation where R is a random number between 0 and 1, e is the base of natural logarithm, λ is greater than 0. Greater λ means smaller inter-arrival time between VMs which results in more intense VMs in the same period of time. Therefore, parameter λ tunes the real-world workloads to better evaluate our strategies performance.

$$IntervalTime = -log_e(R/\lambda). \tag{6}$$

2) TIME WINDOW

Since VMs are continuously arriving, we propose to handle VM requests in groups through time windows. VMs arriving in the same time window will be processed together. Such solution requires a careful design of the time window length. The search of an optimal allocation strategy can have more flexibility when more VMs are available, which may lead to better performance in achieving the optimization goals. However, waiting to gather too many VMs will cause significant delays to handle users' requests. Meanwhile, with more number of VMs handled together, the computational complexity will also increase, leading to further delays. Therefore, there is a trade-off between request delay and the optimization performance of the resulted allocation strategy.

3) DATA TRACE

We simulate the workload of each VM based on PlanetLab cloud workload which is a list of VM CPU utilization percentage values collected on March 3rd, 2011. Each workload is 24-hour long and the interval of utilization measurement is 300s. As a result, the utilization request of a VM may change from time to time, which requires the VM allocation algorithm to dynamically evaluate the workload at each PM accordingly and migrate some allocated VMs in case of server overloading.

D. ANT COLONY OPTIMIZATION

Next, we propose to adopt Ant Colony Optimization (ACO) as a solution to the proposed optimization model. Inspired by natural ant activities, ACO is an algorithm integrating both heuristic information and randomness to find the optimized solution to a problem [12]. The basic idea is that ants carry back their food to colony through different random trails initially, and meanwhile release pheromone on their trails. After a while, trails that take ants less time to travel are piled up with pheromone and become more attractive to ants traveling later. In this way, the shorter trails are reinforced again and again, so that eventually the shortest one will stand out.

A typical application of the ACO algorithm is traveling salesman problem (TSP), an NP-hard problem, of which the optimization goal is to find the shortest path to traverse all cities in a given map. In particular, given an ant k at a city i, the probability for this ant to choose the next city as j is calculated as follows.

$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{l \in N} \tau_{il}^{\alpha} * \eta_{il}^{\beta}} & j \in N \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where N is the set containing all the cities that are not visited by ant k yet; τ_{ij} and η_{ij} represent the pheromone and heuristic value of selecting city j to visit next after city i. The heuristic value η_{ij} can be calculated based on the direct distance between cities i and j (i.e. d_{ij}) as follows.

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{8}$$

From the above discussion, we can see that the heuristic value η_{ij} is determined by the direct distance between cities i and j, which is consistent with the intuitive way of determining the shortest path. In addition, the pheromone value τ_{ij} may initially represent randomness, as ants may take random trails and lay pheromone, and is gradually reinforced by later ants' choices/experiences (i.e. if the path from city i to city j is frequently selected as part of the best path). By adjusting the two parameters α and β, which range from 0 to 1, ACO can

dynamically adjust how important the pheromone and heuristic values are considered, respectively. For example, when $\alpha = 0$, the pheromone information is completely ignored; and the ACO algorithm becomes a pure greedy algorithm. On the other hand, when $\beta = 0$, the heuristic information is completely ignored; and the ACO algorithm becomes a pure random searching algorithm.

By taking both information into account, the ACO algorithm aims to identify the optimal trail by integrating ''exploitation'' (selecting ''optimal'' action based on heuristic information that is already known) and ''exploration'' (attempting to discover new possibilities by selecting a sub-optimal action with certain randomness). In addition, as different ants are independent from one another, the ACO algorithm can be naturally implemented in a distributed way to improve efficiency. Therefore, the ACO algorithm is often applied on NP-hard problems to efficiently find high quality solutions.

E. ADOPTING ANT COLONY OPTIMIZATION (ACO)

The advantages of ACO make it a promising approach to address the secure VM allocation issue in cloud. Furthermore, compared to deep learning based solutions (e.g. reinforcement learning), ACO schemes require much less training data. Therefore, we would like to adopt ACO in the proposed scheme to address the optimization problem discussed in Section III-B. However, such adoption is not trivial due to several challenges. First, how to map the VM assignment issue, which involves two parties as the VM and the PM, to a city visit problem that only considers one party (i.e. cities)? Second, different from TSP where the number of cities to visit is fixed, the secure VM allocation problem only specifies the number of VMs to assign, while leaving the number of physical servers open. How to determine the optimal number of PMs involved? Third, how to model the heuristic and pheromone values in the VM assignment scenario?

We aim to address these challenges in the following two sections. In particular, we need to handle two major steps as: (1) mapping VM allocation as a shortest path problem, and (2) designing heuristic and pheromone values in VM allocation.

1) MAPPING VM ALLOCATION

We propose to address the first two challenges through the following mapping scheme. Recall that the original VM allocation problem is to assign a list of n_v VMs (i.e. V_{list}) to n_s available physical servers, where $n_s \in [n_s^{min}, n_s^{max}]$ is not a fixed value (i.e. the second challenge mentioned above). To simplify the problem, we first divide the entire problem into $n_s^{max} - n_s^{min}$ subproblems, where each subproblem only handles one specific PM number. We will retrieve the optimal solution to the overall problem as the best solution out of the optimal solutions to each subproblem.

Then for each subproblem with a fixed number of PMs, represented by n_s, we aim to assign each VM in the V_{list} to these n_s PMs one by one. Specifically, we create a VM assignment vector A as

$$A^{n_s} = [a_0, a_1, \ldots a_i, \ldots a_{n_v-1}] \tag{9}$$

where a_i represents the PM index that the i^{th} VM in the V_{list} is assigned to. For example, given an assignment $A^3 = [1, 0, 2, 1]$, it indicates that four VMs have been assigned to three different PM as server 1, server 0, server 2 and server 1, respectively. The first challenge mentioned above can then be addressed through this VM assignment vector A. Similar to the TSP problem, where each traversal solution contains a specific order of cities that leads to a certain overall distance; in our problem, each VM assignment A contains a specific combination of VM-PM matching pairs that leads to a certain overall cost.

2) HEURISTIC AND PHEROMONE INFORMATION IN VM ALLOCATION

Here, we address the third challenge: determining the heuristic and pheromone values in the VM allocation problem. Recall that in TSP, the heuristic value is determined as the inverse of the distance between two cities. In our problem, we design the heuristic value η_{ij} as a value related with the cost of assigning VMs to PMs. To record all the assignment costs, we introduce a two dimensional cost matrix C as

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & \ldots & c_{1,n_s} \\ \ldots\ldots\ldots & & c_{v,j} & & \ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ c_{n_v,1} & c_{n_v,2} & c_{n_v,3} & \ldots & c_{n_v,n_s} \end{bmatrix} \tag{10}$$

where $c_{v,j}$ represents the cost of assigning VM v to server j, which is calculated according to equation (1), as the extra cost increase on security risks, power consumption, and workload inequality caused by the assignment of VM v to PM j. Then the heuristic information η_{ij} can be easily obtained from this matrix as

$$\eta_{ij} = \frac{1}{c_{v,j}} \tag{11}$$

Please note that as we assign VMs according to their orders in the V_{list}, the cost $c_{v,j}$ is calculated based on the previous assignment of VM $v-1$ to server i (i.e. $c_{v-1,i}$). Therefore, different orders of the VMs in the V_{list} may lead to different assignment solutions.

Second, we design a two-dimension pheromone matrix Ph as follows.

$$Ph = \begin{bmatrix} ph_{1,1} & ph_{1,2} & ph_{1,3} & \ldots & ph_{1,n_s} \\ \ldots\ldots\ldots & & ph_{v,j} & & \ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ ph_{n_v,1} & ph_{n_v,2} & ph_{n_v,3} & \ldots & ph_{n_v,n_s} \end{bmatrix} \tag{12}$$

where $ph_{v,j}$ represents the current pheromone value of assigning VM v to server j. In the beginning of the problem, as there is no information available for possible assignments, all the values in the initial pheromone matrix are normalized as $\frac{1}{n_s}$. Once a local optimal assignment has been identified, the VM-PM matching pairs that are involved in this assignment will have their pheromone values updated.

3) ITERATIVE ACO FOR SECURE VM ALLOCATION

With all the three challenges addressed, we are now able to present the iterative ACO algorithm for secure VM allocation. In particular, there are five steps as follows.

Step 1, divide the original problem of VM assignment into $n_s^{max} - n_s^{min}$ subproblems. For each subproblem with a fixed number (i.e. n_s) of PMs, an iterative ACO will be launched to find the optimal assignment.

Step 2, for each iteration l, identify the best assignment. Specifically, n_a ants are created, where each ant will start from a V_{list} with a randomly generated order of VMs, and work on constructing its own assignment $A^{n_s,l}$ by considering both the heuristic and pheromone information. Once all the n_a ants have completed their assignments, the total cost of each assignment is stored as an element in a $1 \times n_a$ vector $C^{t,l}$. The assignment with the lowest cost $min(C^{t,l})$ will be identified as the best assignment (i.e. $A_{opt}^{n_s,l}$) for iteration l.

Step 3, update information. If the minimum cost at iteration l is smaller than the optimal cost for the current subproblem (i.e. $min(C^{t,l}) < c_{opt}^{n_s}$), we will have

$$c_{opt} = min(C^{t,l}) \tag{13}$$

$$A_{opt}^{n_s} = A_{opt}^{n_s,l} \tag{14}$$

Consequently, the pheromone information for the next iteration $l+1$ will be updated as follows.

$$ph_{v,j}^{(l+1)} = (1-\varphi)ph_{v,j}^{l} + \varphi \Delta ph_{v,j}^{l}, \tag{15}$$

where $\Delta ph_{v,j}^{l} = \frac{1}{c_{opt}^{n_s}}$. In addition, φ represents how fast the pheromone information is updated. A higher φ value represents a faster speed to forget the out-of-date pheromone information.

Step 4, repeat the above process for L iterations in total. In particular, each new iteration will be performed by another n_a ants with the updated pheromone information, which may lead to some new better assignments. After L iterations, the final $A_{opt}^{n_s}$ will be determined as the best solution for this subproblem.

Step 5, once all the subproblms are addressed, the global optimal assignment is determined as below.

$$A_{opt}^{Global} = \text{optimal}(A_{opt}^{n_s}), \text{ where } n_s \in [n_s^{min}, n_s^{max}] \tag{16}$$

We summarize the proposed scheme in Algorithm 1. For clarity purpose, we use bold notations to represent matrices, capital notations to represent vectors and lower case notations to represent scalar variables. The time complexity of Algorithm 1 is affected by the max and min number of servers, the number of iterations, the number of ants, and the number of VMs. Since the number of iterations and ants are fixed for each execution, the time complexity is roughly $O(n^2)$.

IV. PERFORMANCE EVALUATION

To demonstrate the effectiveness and efficiency of the presented solution, we evaluate our strategy under different amount of users and workloads. Then we compare the performance with two other state-of-the-art VM allocation strategies. The results of simulated experiments indicate that the proposed strategy outperforms the baseline strategies considering workload balance, security, and power cost. Table 2 presents the standard VM configuration that is used in the experiments. Bandwidth and VM size decide the migration time cost. MIPS (Millions of Instructions per Second), Processing Elements, and VM utilization will be used to convert the VM utilization to server utilization. In particular, the real time VM utilization rate traces are generated based on data collected from a real data center. Table 3 presents the power and performance data of the servers used in the simulation.

TABLE 1. Table of notations.

Notation	Description
U_{list}	A list with all users
V_{list}	A list with all VMs
n_s	Number of servers
n_u	Number of users
n_v	Number of VMs
n_a	Number of ants
L	Number of iterations
Ph	Pheromone Matrix
C	Cost matrix for assigning VMs to different servers
C_a	A vector of costs for each ant's assignment
A	VM assignment vector with dimension as $n_v * 1$
$A_{opt}^{n_s}$	optimal VM assignment vector for n_s servers
A_{opt}^{Global}	The global optimal VM assignment

Algorithm 1 ACO Cloud VM Assignment Algorithm

$U_{list} \leftarrow$ All Users
$V_{list} \leftarrow$ All VMs
$n_s^{min}, n_s^{max} \leftarrow$ The max/min number of servers
for $n_s = n_s^{min}$ to $n_s = n_s^{max}$ **do**
  Initialize pheromone matrix Ph, $c_{opt}^{n_s} = Inf$ and $A_{opt}^{n_s} = NULL$
  for $l = 0$ to $l = L-1$ **do**
    for $m = 0$ to $m = n_a - 1$ **do**
      for $v = 0$ to $v = n_v - 1$ **do**
        $C_v[v] = getCost(v, V_{list})$
        $Pr_v[v] = getPro(C_v[v], Ph, \alpha, \beta)$
        $A^{n_s,l}[q] = randomGen(Pr_v[v])$
      end for
      $C_a^l[m] = \sum_{v=1}^{n_v} C_v[v][A^{n_s,l}[v]]$
      if $c_{opt}^{n_s} > C_a^l[m]$ **then**
        $c_{opt}^{n_s} = C_a^l[m]$ and $A_{opt}^{n_s} = A_{opt}^{n_s,l}$
      end if
    end for
    $Ph = pheUpdate(c_{opt}^{n_s})$
  end for
end for
return $A_{opt}^{Global} = optimal(A_{opt}^{n_s})$, where $n_s \in [n_s^{min}, n_s^{max}]$

A. KEY PARAMETERS TESTING

1) ACO PARAMETERS

In this section, we mainly investigate the impact of three key parameters from the ACO algorithm: α and β. Specifically,

TABLE 2. Virtual Machine Configuration.

Configuration Parameters	Default Value
MIPS	2000
Processing Element	2
Memory	1GB
Bandwidth	100 Mbit/s
VM Size	2.5 GB
VM CPU Utilization	0% - 100%

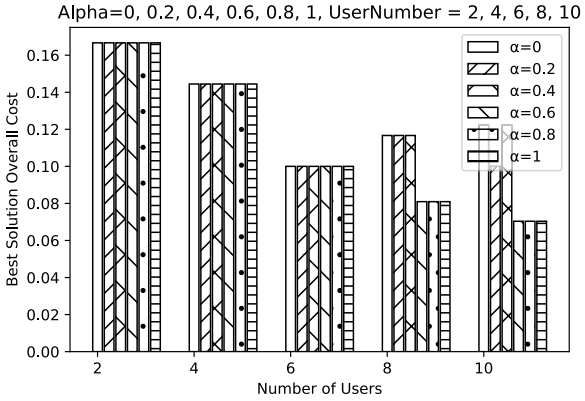


FIGURE 1. Impact of alpha on overall costs.

α and β are ranging from 0 to 1, representing the weights of the pheromone and the heuristic values to be considered in the optimization, respectively.

a: IMPACT OF α

To validate the impact of α on the overall costs, we fix $\beta = 0.9$, $\varphi = 0.8$ and change α from 0 to 1. The results are shown in Fig. 1. In particular, we can observe that the overall costs are not sensitive to α values when the number of users is small. When there are more users (> 6), greater α values will result in smaller overall costs in general. This is reasonable. Recall that the α value indicates the weight of pheromone information to be considered, which starts from an identical value for all possible VM-PM matching pairs and needs to be accumulated over time. When there are not many users/VMs to assign, there is not sufficient accumulation for the pheromone value to represent better matching pairs. As a result, a higher or lower weight (i.e. α) of the pheromone values will not influence the overall costs much. However, when more users/VMs are available, the pheromone information can be accumulated more to represent better matching pairs. Therefore, a higher weight (i.e. greater α value) will effectively help to reduce the overall costs.

b: IMPACT OF φ

Similarly, to evaluate the impact of φ on the optimized overall costs, we set $\alpha = 0.9$, $\beta = 0.89$ and change φ from 0 to 1. The similar trend is observed in Fig. 2, which indicates that although the overall costs can yield lower values when there are more users/VMs, they are not sensitive to the change of φ values for a fixed number of users, especially when the user

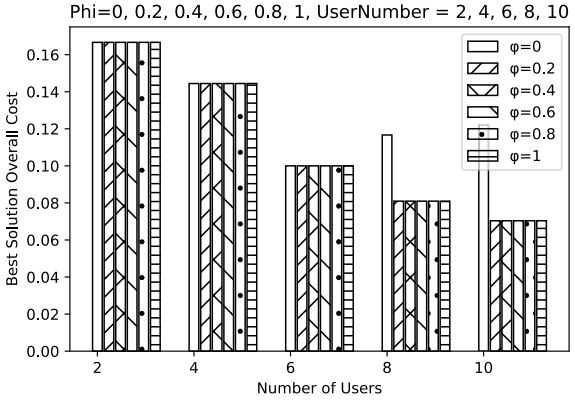


FIGURE 2. Impact of phi on overall costs.

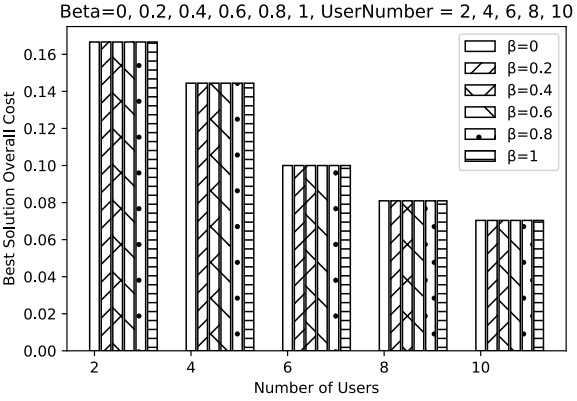


FIGURE 3. Impact of beta on overall costs.

number is small (< 8). However, when more users/VMs need to be allocated, $\varphi = 0$ will not lead to good results. The reason is as follows. Recall that a larger φ value indicates a faster update of the pheromone value, $\varphi = 0$ leads to no pheromone updates at all. It indicates that the pheromone values in all the later iterations will be exactly the same as the initial pheromone values, which are set as an identical value for all possible VM-PM matching pairs. It actually mitigates the impact of pheromone as it cannot be used to help differentiate different matching pairs. In other words, when only considering heuristic information, the overall cost will not yield its optimal value. But except the case $\varphi = 0$, other φ values will yield the same optimal overall costs, indicating that regardless of the pheromone updating speed, as long as it is not zero, the optimal overall costs can always be achieved.

c: IMPACT OF β

To validate the impact of β on the overall costs, we set $\alpha = 0.9$, $\varphi = 0.8$ and change β from 0 to 1. The results are shown in Fig. 3. Similar trend can be observed as that although the overall costs can yield lower values when there are more users/VMs (i.e. easier to tune and balance), they are not sensitive to the change of β values when the number of servers is fixed. It indicates that a wide range of β values

TABLE 3. Power Consumption of Hosts [41], [42], [43], [44].

Host Model	Average Active Power (Watt)										
Utilization:	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Fujitsu Primergy RX1330 M1	13.8	20.8	23.9	26.3	29.1	32.6	36.2	42.0	48.6	55.9	63.7
Inspur NF5280M4	44.4	83.3	101	118	135	146	161	190	218	255	301
Dell PowerEdge R820	71.8	135	156	176	198	219	243	269	297	318	374
IBM NeXtScale nx360 M4	497	814	947	1079	1211	1344	1493	1648	1863	2108	2414

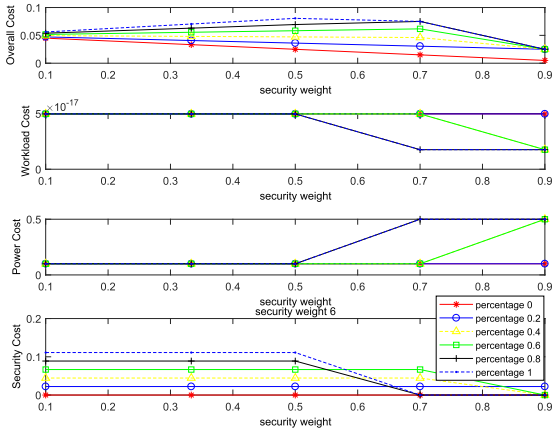


FIGURE 4. Impact of Malicious User Percentage.

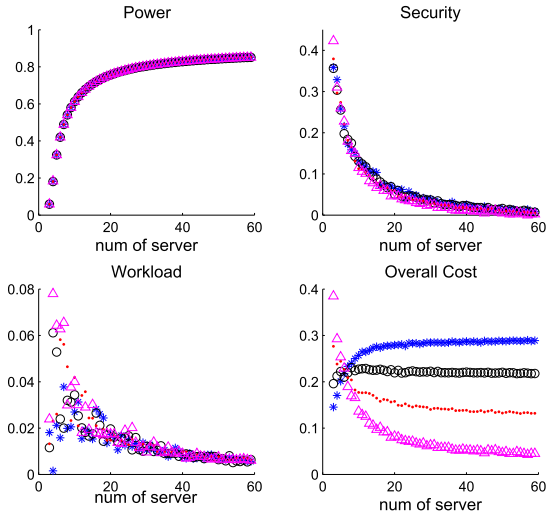


FIGURE 5. Impact of Security Weights Per Server Num.

can be chosen for the ACO algorithm and will not lead to a dramatic performance change.

2) PERCENTAGE OF MALICIOUS USERS

Besides the ACO parameters, we also want to evaluate how the percentage of malicious users will influence the overall costs. In particular, we have changed the malicious user percentage from 0 to 1, and the security weight from 0.1 to 0.9. The resulted overall costs are shown in Fig. 4. In Fig. 4, the x-axis and y-axis represent the malicious user percentage and overall costs, respectively. The five curves represent different weights of the security factor in our objective function (i.e. Equation 1). There are several observations. First, when the malicious user percentage increases, smaller security weights often lead to slower increase of the overall costs. This is because when security is less cared (i.e. lower weights), the overall costs are less sensitive to the malicious user percentage changes. Second, when security weights are set high (e.g. 0.7 or 0.9), we can observe that the cost curves achieve their peek values at a certain point and will not continuously increase when the malicious user percentage increases. This is because higher security weights make the optimization process tilted towards the security aspect, which can effectively guarantee no raises of the security risks while malicious user percentage continuously growing. It also shows the effectiveness of the proposed optimization scheme.

3) THE WEIGHT OF SECURITY

As the weight of security is also a key parameter to be determined by the cloud service provider, we aim to study its impact on the overall costs, which consists of the costs for power, workload inequality, and security. Specifically, the experiments are conducted by simulating 30 users, where each user has two VMs with random utilization requests. Recall that our proposed algorithm actually divides the entire optimization problem into smaller subproblems, with each one of which handles only a fixed number of servers. So we first present the impact of security weights for each specific number of servers (i.e. each subproblem) in Fig. 5. Please note that each of the data points here represents the optimal costs for a specific subproblem, not the global optimal costs.

In particular, there are four subplots, showing the overall costs, workload inequality costs, power costs, and security costs. In each subplot, there are four curves representing different security weights as 0.3, 0.5, 0.7 and 0.9, respectively. The corresponding weights of power consumption and workload inequality are set as $w_W = w_P = (1-w_S)/2$. In addition, the x-axis of each subplot represents the number of occupied physical servers, and the y-axis represents the corresponding costs.

From this figure, we can make several observations. First, as shown in the upper left subplot, the power costs are not sensitive to security weights, but mainly dominated by the number of PMs. Even if different security weights lead to different optimal assignment solutions, as long as the solutions are occupying the same amount of PMs, the power costs for these assignments will be roughly the same.

Second, from the upper right subplot, we can observe that regardless of the security weight, when the number of

occupied PMs increases, the security costs (i.e. security risks) are decreasing. This is because when the number of occupied PMs increases, the VMs are more spread out, indicating a higher possibility for VMs from different users to be allocated on different PMs, leading to lower security risks/costs.

Third, from the lower left subplot, we can observe that for all different security weights, the workload inequality costs will always increase first and drop later, when the number of PMs is increasing. The reason is as follows. When the number of PMs is small, most of the VMs are squeezed in the PMs, leading to very limited extra capacity for each PM. Therefore, the workload is roughly balanced among different PMs. When the number of occupied PMs starts to increase, VMs are allocated more flexibly to different PMs, which easily makes more PMs have different capacity left, leading to more imbalanced workload. However, as the number of occupied PMs continues to grow, VMs are spread out, leading to very few VMs sharing the same PM. As a result, it becomes easier again to balance the workload among different PMs.

At the end, as shown in the lower right plot, the overall cost is a trade-off of the three factors: power consumption, workload inequality and security risks, and therefore can be significantly influenced by the security weight. Specifically, when the number of occupied PMs is small, the costs of power and workload inequality can be small. However, as VMs are squeezed to reach the maximum capacity of each server, the security risk is greatly increased. As a result, the local optimal solution will yield higher overall costs if the security factor is the dominated factor (i.e. high security weight), and lower overall costs if power and workload inequality are dominated factors (i.e. low security weight). On the other hand, when the number of occupied PMs is large, the power cost goes up. However, as VMs are spread out on different PMs, the security risks and workload inequality can be low. As a result, the local optimal solution will yield lower overall costs if the security factor is the dominated factor (i.e. high security weight), and higher overall costs if power is the dominated factor (i.e. low security weight).

Next, we aim to study the impact of security weights on the global optimal solution in Fig. 6. Different from Fig. 5, where the local optimal costs for each subproblem are analyzed, here we only examine the global optimal solution with the best number of PMs for each specific security weight. Specifically, the five subplots represent the influence of security weights on the optimal costs for the overall solution, workload inequality, power, and security, as well as the optimal number of servers, respectively.

From Fig. 6, we can observe that when security weight gradually increases, the corresponding optimal solutions tend to make more efforts on lowering the security costs, which will lead to more number of occupied PMs and higher power costs, but lower workload inequality and overall costs.

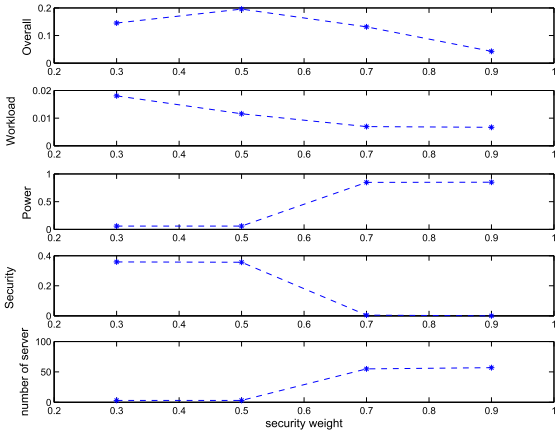


FIGURE 6. Impact of Security Weights on Optimal Costs.

B. PERFORMANCE COMPARISON

In this section, we compare the proposed scheme with two other existing allocation strategies. The first one is Round-Robin, a classic algorithm that allocates resources, e.g. physical machine (PM), in equal portions and in circular order, handling all VMs without priority. In particular, as the original Round-Robin algorithm cannot specify the number of physical PMs to be involved for allocation, we implement the algorithm in a way that the algorithm spawns a group of PMs each time. We set the number of PMs in a group as 2, 4, 6 and 8 each time a new spawning process is needed. The modified Round-Robin algorithms are named RR2, RR4, RR6, RR8 respectively and we examine their performance at different choices of the number of spawning PMs.

The second one is the Previously-Selected-Servers-First (PSSF) scheme proposed in [22], a representative study of the state-of-the-art VM allocation schemes that optimizes security, workload balance and power consumption through a heuristic strategy. In particular, PSSF tends to select from two strategies: stacking or spreading. Each new VM will be first stacked to the same PM to which other VMs from the same user has been allocated. If the PM has reached its capacity, the new VM will be spread to a new PM. Similar to Round-Robin, PSSF cannot explicitly determine the number of physical PMs to involve. Instead, it involves a group of physical PMs each time, and the new group of PMs will not be involved until the existing PMs reach their capacity. Therefore, a key parameter for PSSF is the number of PMs in each group. In our experiments, we set the number as 2, 4, 6 and 8, respectively to examine its performance and name the algorithms as PSSF2, PSSF4, PSSF6, and PSSF8, respectively.

As discussed in Section III-C1, we use Poisson distribution to simulate different scenarios where the number of users and VMs increases as the parameter λ increases in a given amount of time. In particular, λ varies from 0.001 to 0.01, which indicates the number of incoming VMs ranging from 10 to 100 during the total experiment duration. We generate 100 sets of data from Poisson distribution at each λ values and take the average performance to make a fair comparison. The representative results are presented in Fig. 7.

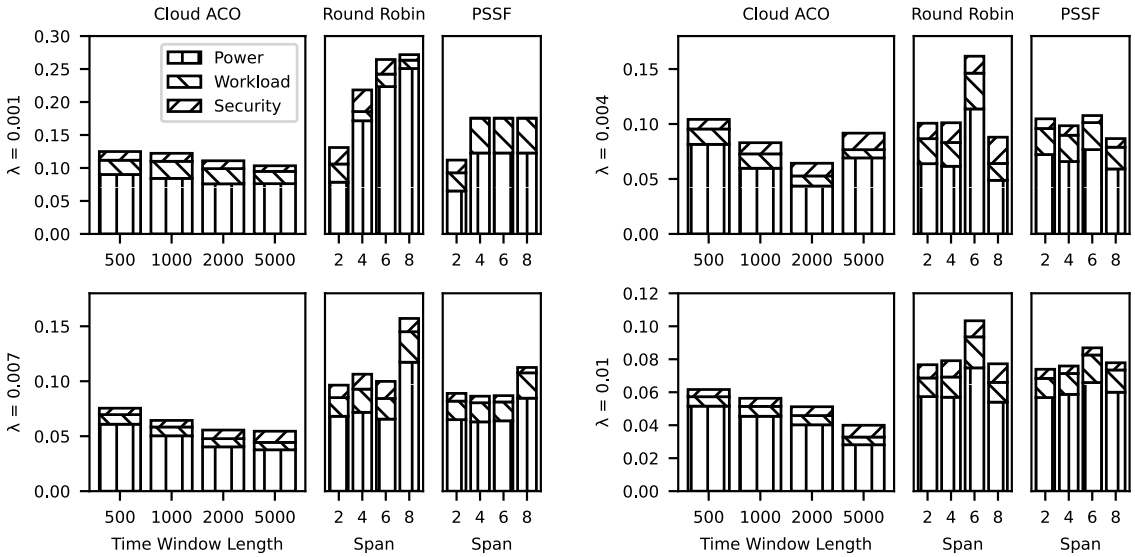


FIGURE 7. Performance Comparison for Scenario with different λ.

Fig. 7 contains 4 subplots with λ set to 0.001, 0.004, 0.007 and 0.01, representing the scenarios with the VM numbers as 10, 40, 70 and 100 respectively to demonstrate the performance of the proposed scheme, Round-Robin and PSSF. In addition, the malicious user percentage is set as 20%, a reasonable estimation on the malicious context, and the weights for security, power, and workload inequality are all set as 1/3. The bars from left to right represent the proposed scheme with its key parameter (i.e. time window length) equals 500, 1000, 2000 and 5000; Round-Robin with span set to 2, 4, 6, and 8; and lastly PSSF with span set to 2, 4, 6, and 8.

As shown in Fig. 7, the total cost of the proposed scheme on the top right subplot, where λ equals 0.004, drops at time window length 200 and then increases along with the increase of time window length. This is because when the incoming number of VMs are relatively small, the number of VMs falling in one time window may vary, leading to some inconsistencies in the performance. As the number of VMs increases where λ equals 0.007 and 0.01 on the bottom two subplots, the proposed schemes tends to be more stable with a downward slope as time window length increases. This is because when time window length increases, a larger portion of incoming VMs are considered by the algorithm at each calculation, thus results in a lower overall cost. On the other hand, larger time window length means longer average wait time for each incoming VM before assigning to a actual PM. The trade off here has to be considered in a real world scenario to accomplish an optimized configuration.

Among these algorithms, the proposed scheme always achieves the best performance in terms of the optimal total costs, which validates its effectiveness. Moreover, the detailed subcosts for security, power, and workload balance are also shown for each allocation scheme. There are several observations. First, at λ=0.001, the proposed scheme with time window length set to 100 achieves the best overall score. PSSF4, PSSF6 and PSSF8 achieves the exact same score with zero cost on security. This is because when the number of incoming VMs is relatively small (10 VMs in this case), according to the behaviour of PSSF, VMs will simply span to a new server when there are enough PMs allocated. This results in zero security cost. However, because of the usage of new PMs, the power costs will significantly increase, leading to much higher overall costs. Secondly, when λ=0.004, the proposed scheme with time window length 2000 achieves the minimum overall costs. Even though PSSF6 has the lowest security cost, the proposed scheme has far less power cost compared to PSSF6, which results in an overall lowest cost. RR2, RR4, RR6 and RR8 behave similarly to PSSF, but with higher security cost since they tend to stack VMs from different users into the same PM. When λ=0.007, there are total 70 incoming VMs, which makes it a more realistic scenario. The proposed scheme outperforms both RR and PSSF, and the worst score of the proposed scheme is nearly the same as the best score of both RR and PSSF. Both RR and PSSF have significantly larger power cost and workload balance cost than the proposed scheme because the way the stack up VMs and spawning PMs, and the choices of span 2, 4, 6, 8 of the two schemes among different λ does not always generate the same results, which make it hard to detect the optimized choice of span in a complex, realistic scenario. Similar observation can be concluded on the last subplot when λ is set to 0.01 with a total of 100 incoming VMs, this confirms that the proposed scheme is robust and scalable with better performance among all these schemes.

V. CONCLUSION

Co-residence attack has raised significant concerns as the increassing popularity of cloud computing. Attackers are able to take advantage of the resource sharing in multi-tenant cloud to perform diverse attacks against their co-residents on the same physical server. We proposed to defend against such

co-residence attacks through a secure, workload-balanced, and energy-efficient VM allocation strategy. and modeled the VM allocation problem as an optimization problem. As this optimization problem is NP-hard, we further applied the Ant Colony Optimization (ACO) algorithm, an evolutionary algorithm inspired by natural ant activities, to identify the optimal allocation strategy. Experiment results demonstrated that the proposed scheme can make the multi-tenant cloud secure and power efficient.

REFERENCES

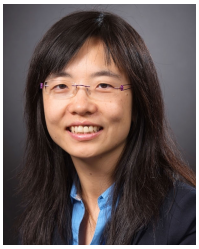
[1] O. Aciiçmez, "Yet another microarchitectural attack: Exploiting I-cache," in *Proc. ACM Workshop Comput. Secur. Archit.*, 2007, pp. 11–18.
[2] O. Aciiçmez, O. K. Koç, and J.-P. Seifert, "Predicting secret keys via branch prediction," in *Topics Cryptology–CT-RSA 2007*. Cham, Switzerland: Springer, 2006, pp. 225–242.
[3] O. Aciiçmez, O. K. Koç, and J.-P. Seifert, "On the power of simple branch prediction analysis," in *Proc. 2nd ACM Symp. Inf., Comput. Commun. Secur.*, 2007, pp. 312–320.
[4] Y. Azar, S. Kamara, I. Menache, M. Raykova, and B. Shepard, "Co-location-resistant clouds," in *Proc. 6th Ed., ACM Workshop Cloud Comput. Secur.*, 2014, pp. 9–20.
[5] J. E. Bell and P. R. McMullen, "Ant colony optimization techniques for the vehicle routing problem," *Adv. Eng. Inform.*, vol. 18, no. 1, pp. 41–48, 2004.
[6] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency Comput., Pract. Exper.*, vol. 24, no. 13, pp. 1397–1420, Sep. 2012.
[7] S. Chhabra and A. K. Singh, "A secure vm allocation scheme to preserve against co-resident threat," *Int. J. Web Eng. Technol.*, vol. 15, no. 1, pp. 96–115, 2020.
[8] R. C. Chiang, S. Rajasekaran, N. Zhang, and H. H. Huang, "Swiper: Exploiting virtual machine vulnerability in third-party clouds with competition for I/O resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1732–1742, 2014.
[9] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: Outsourcing computation without outsourcing control," in *Proc. ACM workshop Cloud Comput. Secur.*, 2009, pp. 85–90.
[10] B. Coppens, I. Verbauwhede, K. D. Bosschere, and B. D. Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *Proc. 30th IEEE Symp. Secur. Privacy*, May 2009, pp. 45–60.
[11] B. R. Williams and A. Chuvakin, *PCI Compliance: Understand and Implement Effective PCI Data Security Standard Compliance*. Syngress, 2014.
[12] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.
[13] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.
[14] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 13–23, 2007.
[15] E. Feller, L. Rilling, and C. Morin, "Energy-aware ant colony based workload placement in clouds," in *Proc. IEEE/ACM 12th Int. Conf. Grid Comput.*, Sep. 2011, pp. 26–33.
[16] Z. Feng, B. Bai, B. Zhao, and J. Su, "Shrew attack in cloud data center networks," in *Proc. 7th Int. Conf. Mobile Ad-Hoc Sensor Netw.*, Dec. 2011, pp. 441–445.
[17] N. E. Toklu, L. M. Gambardella, and R. Montemanni, "A multiple ant colony system for a vehicle routing problem with time windows and uncertain travel times," *J. Traffic Logistics Eng.*, vol. 2, no. 1, 2014.
[18] L. M. Gambardella and M. Dorigo, "An ant colony system hybridized with a new local search for the sequential ordering problem," *INFORMS J. Comput.*, vol. 12, no. 3, pp. 237–255, 2000.
[19] J. Glanz, *The Cloud Factories Power, Pollution and the Internet*. New York, NY, USA: New York Times, 2012.
[20] D. Grunwald and S. Ghiasi, "Microarchitectural denial of service: Insuring microarchitectural fairness," in *Proc. 35th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Nov. 2002, pp. 409–418.
[21] Y. Han, T. Alpcan, J. Chan, and C. Leckie, "Security games for virtual machine allocation in cloud computing," in *Proc. Int. Conf. Decis. Game Theory Secur.* Cham, Switzerland: Springer, 2013, pp. 99–118.
[22] Y. Han, J. Chan, T. Alpcan, and C. Leckie, "Using virtual machine allocation policies to defend against co-resident attacks in cloud computing," *IEEE Trans. Depend. Sec. Comput.*, vol. 14, no. 1, pp. 95–108, Jan./Feb. 2017.
[23] D. Hyde. (2009). *A Survey on the Security of Virtual Machines*. [Online]. Available: Www1.cse.wustl.edu/~jain/cse571-09/ftp/vmsec/index.html
[24] A. Jasti, P. Shah, R. Nagaraj, and R. Pendse, "Security in multi-tenancy cloud," in *Proc. 44th Annu. IEEE Int. Carnahan Conf. Secur. Technol.*, Oct. 2010, pp. 35–41.
[25] G. Keramidas, A. Antonopoulos, D. N. Serpanos, and S. Kaxiras, "Non deterministic caches: A simple and effective defense against side channel attacks," *Design Autom. Embedded Syst.*, vol. 12, no. 3, pp. 221–230, 2008.
[26] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, "Cloud task scheduling based on load balancing ant colony optimization," in *Proc. 6th Annu. Chinagrid Conf.*, Aug. 2011, pp. 3–9.
[27] M. Li, Y. Zhang, K. Bai, W. Zang, M. Yu, and X. He, "Improving cloud survivability through dependency based virtual machine placement," in *SECRYPT*, pp. 321–326, 2012.
[28] Y. Liu, X. Ruan, S. Cai, R. Li, and H. He, "An optimized VM allocation strategy to make a secure and energy-efficient cloud against co-residence attack," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Mar. 2018, pp. 349–353.
[29] X. Luo, L. Yang, L. Ma, S. Chu, and H. Dai, "Virtualization security risks and solutions of cloud computing via divide-conquer strategy," in *Proc. 3rd Int. Conf. Multimedia Inf. Netw. Secur.*, Nov. 2011, pp. 637–641.
[30] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 3, pp. 118–129, 2012.
[31] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333–346, Aug. 2002.
[32] C. Ming, Y. Bingjie, and L. Xiantong, "Multi-tenant saas deployment optimisation algorithm for cloud computing environment," *Int. J. Internet Protocol Technol.*, vol. 11, no. 3, pp. 152–158, 2018.
[33] C. Momm and W. Theilmann, "A combined workload planning approach for multi-tenant business applications," in *Proc. IEEE 35th Annu. Comput. Softw. Appl. Conf. Workshops*, Jul. 2011, pp. 255–260.
[34] Y. Natarajan, S. Kannan, and G. Dhiman, "Task scheduling in cloud using ACO," *Recent Adv. Comput. Sci. Commun.*, vol. 13, pp. 1–6, Mar. 2022.
[35] K. Owens, *Securing Virtual Computer Infrastructure in the Cloud*. Jefferson City, MI, USA: SavvisCorp, 2009.
[36] C. D. Patel, C. E. Bash, R. Sharma, M. Beitelmal, and R. Friedrich, "Smart cooling of data centers," U.S. Patent 6 574 104 B2, Oct. 5, 2001.
[37] A. Ragmani, A. E. Omri, N. Abghour, K. Moussaid, and M. Rida, "A performed load balancing algorithm for public cloud computing using ant colony optimization," *Recent Patents Comput. Sci.*, vol. 11, no. 3, pp. 179–195, 2018.
[38] A. Ragmani, A. Elomri, N. Abghour, K. Moussaid, and M. Rida, "FACO: A hybrid fuzzy ant colony optimization algorithm for virtual machine scheduling in high-performance cloud computing," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 10, pp. 3975–3987, 2019.
[39] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Comput.*, vol. 16, no. 1, pp. 69–73, Jan. 2012.
[40] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 199–212.
[41] SPEC. *Dell Inc. PowerEdge R820 (Intel Xeon E5-4650 v2 2.40 GHz)*. Accessed: 2008. [Online]. Available: https://www.spec.org/power_ssj2008/results/res2014q2/power_ssj2008-20140401-00654.html
[42] SPEC. *Fujitsu FUJITSU Server PRIMERGY RX1330 M1*. [Online]. Available: https://www.spec.org/power_ssj2008/results/res2014q3/power_ssj2008-20140804-00662.html
[43] SPEC. *IBM Corporation IBM NeXtScale nx360 M4 (Intel Xeon E5-2660 v2)*. Accessed: 2008. [Online]. Available: https://www.spec.org/power_ssj2008/results/res2014q2/power_ssj2008-20140421-00657.html
[44] SPEC. *Inspur Corporation NF5280M4*. Accessed: 2008. [Online]. Available: https://www.spec.org/power_ssj2008/results/res2014q4/power_ssj2008-20140905-00673.html

[45] F.-X. Standaert, "Introduction to side-channel attacks," in *Secure Integrated Circuits and Systems*. Cham, Switzerland: Springer, 2010, pp. 27–42.
[46] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 1–11, 2011.
[47] H. Takabi, J. B. D. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security Privacy*, vol. 8, no. 6, pp. 24–31, Nov./Dec. 2010.
[48] W. J. Brown, V. Anderson, and Q. Tan, "Multitenancy-security risks and countermeasures," in *Proc. 15th Int. Conf. Netw.-Based Inf. Syst.*, 2012, pp. 7–13.
[49] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, "Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense)," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 281–292.
[50] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in Xen," in *Proc. 3rd ACM Workshop Cloud Comput. Secur. Workshop*, 2011, pp. 41–46.
[51] S. Waheed, B. P. C. N. Islam, and S. H. Bhuiyan, "Security of side channel power analysis attack in cloud computing," *Global J. Comput. Sci. Technol.*, vol. 14, no. 4, pp. 1–8, 2015.
[52] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 494–505, 2007.
[53] J. Wu, L. Ding, Y. Lin, N. Min-Allah, and Y. Wang, "XenPump: A new method to mitigate timing channel in cloud computing," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 678–685.
[54] Y. Zhang and M. K. Reiter, "Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 827–838.
[55] M. Zheng. *Virtualization Security in Data Centers and Cloud*. Accessed: 2011. [Online]. Available: http://www.cse.wustl.edu/~jain/cse571-11/ftp/virtual/

LU CAO received the B.S. and M.S. degrees from Santa Clara University, in 2014 and 2017, respectively, where he is currently pursuing the Ph.D. degree under the supervision of Dr. Yuhong Liu with the Department of Computer Engineering. His research interests include trustworthy computing and cloud computing.

RUIWEN LI received the bachelor's and master's degrees from Santa Clara University. Her research interest includes cloud computing security.

XIAOJUN RUAN received the B.E. degree in computer science and technology from Shandong University, in 2005, and the Ph.D. degree from Auburn University, 2011. He is an Associate Professor with the Department of Computer Science, California State University, East Bay. His research interests include cloud computing, data storage systems, parallel and distributed computing, and computer security.



YUHONG LIU (Senior Member, IEEE) received the B.S. and M.S. degrees the from Beijing University of Posts and Telecommunications, in 2004 and 2007, respectively, and the Ph.D. degree from the University of Rhode Island, in 2012. She is an Associate Professor with the Department of Computer Engineering, Santa Clara University. Her research interests include trustworthy computing on the Internet-of-Things, cloud computing, blockchain, and online social media. She has been a Distinguished Visitor with IEEE Computer Society, since 2022, and a Distinguished Lecturer with the Asia-Pacific Signal and Information Processing Association (APSIPA), since 2021.