# REPTTACK: Exploiting Cloud Schedulers to Guide Co-Location Attacks

Chongzhou Fang*, Han Wang*, Najmeh Nazari*, Behnam Omidi†, Avesta Sasan*,
Khaled N. Khasawneh†, Setareh Rafatirad*,and Houman Homayoun*
* University of California, Davis
Email: {czfang,hjlwang,nnazaribavarsad,asasan,srafatirad,hhomayoun}@ucdavis.edu
† George Mason University
Email: {bomidi,kkhasawn}@gmu.edu

***Abstract*—Cloud computing paradigms have emerged as a major facility to store and process the massive data produced by various business units, public organizations, Internet-of-Things (IoT), and cyber-physical systems (CPS). To meet users' performance requirements while maximizing resource utilization to achieve cost-efficiency, cloud administrators leverage schedulers to orchestrate tasks to different physical nodes and allow applications from different users to share the same physical node. On the other hand, micro-architectural attacks, e.g, side-channel attacks, transient execution attacks, and Rowhammer attacks, exploit the shared resources to compromise the confidentiality/integrity of a co-located victim application. Since co-location is an essential requirement for micro-architectural attacks, in this work, we investigate whether attackers can exploit the cloud schedulers to satisfy the co-location requirement of the micro-architectural attacks. Specifically, in this paper, we comprehensively analyze if attackers can influence the scheduling process of cloud schedulers to co-locate with specific targeted applications in the cloud. Our analysis shows that for cloud schedulers that allow users to submit application requirements, an attacker can carefully select the attacker's application requirements to influence the scheduler to co-locate it with a targeted victim application. We call such attack *Rep*lication A*ttack* (REPTTACK). Our experimental results, in both a simulated cluster environment and a real cluster, show similar trends; a single attack instance can reach up to $50\%$ co-location rate (probability of co-location) and with only $5$ instances the co-location rate can reach up to $80\%$ in a heterogeneous cloud. Furthermore, we propose and evaluate a mitigation strategy that can help defend against REPTTACK. We believe that our results highlight the fact that schedulers in multi-user clusters need to be more carefully designed with security in mind, and the process of making scheduling decisions should involve as little user-defined information as possible.**

## I. INTRODUCTION

In the last two decades, the computing paradigms have experienced a tremendous change, with vast amounts of data being amassed by various business units and public organizations. This rate of data amassing is further fueled by the advancements in the Internet-of-Things (IoT) and cyber-physical systems (CPS) equipped with networking capabilities and rapid movement towards paperless organizations [46]. For security consideration, virtual machines (VMs) [42] or containers [41] are utilized to achieve software level isolation between different users. VMs or containers are called instances in this work. To better manage the submitted workloads and improve utilization rate automatically, scheduling algorithms are proposed to help decide resources assigned to each task submitted and which physical node a task should reside. A number of schedulers, including Apollo [10], YARN [52] achieve great success in the industry. However, the current design of the scheduling algorithms focuses on enhancing performance, utilization rate, and load-balancing without the security consideration, which may bring new vulnerability as we explore in this work.

Micro-architectural attacks, such as side-channel attacks [57], covert-channel attacks [40], Rowhammer [32], and transient execution attacks [33], [36], [34], exploit shared resources to compromise the confidentiality/integrity of a co-located victim application/instances. Thus, attackers usually target clouds to launch micro-architectural attacks since clouds allow attackers' instances to share resources with other users' instances to achieve cost-efficiency. However, a major challenge in launching micro-architectural attacks in clouds is the ability of the attack instance to co-locate with a targeted victim. While prior work showed that co-located applications can be identified using network probing [43] or side-channel analysis [60], [25], the only way to increase the co-location probability of the attack instance with a targeted victim is by launching a large number of attack instances (possibly with the assistance of placement timing locality), i.e., brute-force approach, and hope that the attack instance co-locates with a targeted victim. Nonetheless, to defend against the brute-force, cloud managers can limit the number of instances a user can issue within a time frame.

In this work, we explore whether attackers can exploit the cloud schedulers to force co-locating an attack instance with a targeted victim. In particular, we target schedulers that allow users to provide specifications of the submitted instances. These specifications influence the placement decisions of the submitted instances. Our attack strategy is to camouflage the victim specifications for the attack instance to achieve co-location. Thus, allowing the attacker to achieve co-location by launching a few instances, which can't be defeated by limiting the number of instances a user can submit.

To achieve this, we analyze the parameters that schedulers take as input to decide the location of submitted workloads

Network and Distributed Systems Security (NDSS) Symposium 2022
24-28 April 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/ndss.2022.23149
www.ndss-symposium.org

and explore the possibility of manipulating the scheduling outcome for malicious instances by controlling the parameters. In particular, this work investigates the effects of certain parameters by analyzing the co-location rate of malicious instances and victim instances under different parameter settings, and validate that they contribute to vulnerabilities of schedulers. We choose open-source scheduling strategies based on the filter-score scheduling method as examples and find that such scheduling algorithms leverage a special feature called affinity that allows users to fine-tune the scheduling output makes the scheduler more vulnerable. We find that the feature exists in schedulers of cloud architecture like Kubernates [2], Openstack [5], OpenNebula [4], etc. and are used in a recent IoT system [44]. Hence we believe the risk we identify in this paper applies to real-world scenarios. Therefore, showing that security needs to be considered when designing cloud schedulers on par with performance and cost-efficacy.

Lastly, based on our findings, we provide a strategy to mitigate the vulnerability by making a small change to the scheduler and show how bringing randomness in the scheduling process can help avoid the security risks.

Our contributions in this paper are:

- We identify the security risks of manipulating schedulers to achieve co-location with victim instances with less than $5$ malicious instances requested.
- Experiments that simulate the process of deploying applications in real-world scenarios have been done in both simulation and CloudLab cluster [18] to identify the vulnerable features.
- We find that the use of affinity features in filter-score schedulers can induce security problems and propose REPTTACK, which is an attack method that utilizes such security vulnerabilities. We show that with these features used, REPTTACK can enable malicious users to achieve a relatively high co-location rate in a heterogeneous cluster.
- Based on our findings, we conclude several guidelines for attackers, non-malicious users, and scheduler designers regarding how to utilize or how to avoid the security risks.
- We propose a mitigation method and validate it in our simulation. It shows how bringing randomness to the scheduling process can help to defend against co-location attacks.

Our work focuses on achieving co-location to satisfy the prerequisite step of launching micro-architectural attacks and considers the implementation of micro-architectural attacks on a specific hardware as an orthogonal topic to this work. Attackers can utilize micro-architectural attack methods reported in literature (e.g. [17], [19], [33], [36]) to craft their own attack instances.

The remainder of this paper is organized as follows. In Section II, we will provide a brief introduction to cloud schedulers and some related background knowledge on micro-architectural attacks. In Section III, we provide the assumptions we make in this study. Section IV summarizes our attack strategy, and Section V presents corresponding results and analysis. Section VI describes our mitigation strategy and presents related data. In Section VII we provide a discussion that focuses more on the big picture of this field and points out our future direction. Related literature will be reviewed in Section VIII, and we conclude the entire paper in Section IX.

## II. BACKGROUND

This section first introduces cloud schedulers followed by an overview of micro-architectural attacks.

### A. Cloud Schedulers

A scheduler is an essential component of a distributed computing system. It orchestrates resources, assigns resources in the system, and makes placement decisions that satisfy user requirements with fairness for all users [45]. In this paper, we consider the scheduler as the component that decides how a user instance should be placed and provide resources. The scheduling algorithm receives user requirements on placement and assigns user instances to physical nodes that can satisfy users' needs. We summarize the working procedure of cloud schedulers as: First, users submit needed scheduling constraints (needed resources, requirements on nodes, etc.) to the cloud scheduler. Then, the cloud scheduler considers all available resources in the cluster as well as user constraints and assigns a node with enough resources that match the required constraints to that user's instance. Different user instances can be allocated to the same node and share the available hardware resources. A more detailed elaboration of the scheduler we target is provided in Section IV-A.

Cloud schedulers are extensively studied in academia and widely used in industry [52], [45], [10], [30], [54], [21]. Open-source cloud architectures, including schedulers as components, are also utilized in the industry, e.g., OpenStack [5] is deployed in multiple industrial companies (ELASTX, China Mobile [5], for example). However, the previous focus of academia and industry is performance, load-balancing and cost. In contrast, we focus on scheduler security in this work.

### B. Micro-architectural Attacks

Micro-architectural attacks exploit the functionality, optimization, or the physical imperfections of shared resources to compromise the confidentiality/integrity of a victim application. Furthermore, micro-architectural attacks are software-based attacks, i.e., can be launched remotely, and can be categorized into the following four categories:

**(1) Side-channel attack:** is one emerging category of techniques in the field of computer security. Side-channel attack utilizes the data-dependent effects of computation/operations/optimizations, which are also referred to as side-channels, to obtain sensitive information, like secret keys, password and etc. from target software systems [58], [19]. Researchers have shown that by utilizing side-channel information such as the access time to caches [58], [19], [17] information can be leaked from co-located applications. Compared to traditional attack techniques, side-channel attack is more stealthy and harder to defend against since it targets hardware design weak points.

**(2) Transient execution attacks:** exploits flaws in micro-architecture to execute operations that should not have happened [11]. For example, in 2018, Meltdown attack is proposed [36], which utilizes out-of-order execution as a new side-channel and enables attackers to read arbitrary physical memory addresses of other processes. A more recent similar work, Spectre Attack [33], additionally utilizes branch prediction to mistrain branch predictors and lead to the execution of illegal paths. Spectre can be applied to a wider range of processors. These two attack technologies affect a wide range of processors [3] since they target speculative execution, which has been deployed in commercial processors for decades. Moreover, despite the advances in the proposed and deployed defenses against transient execution attacks [56], [55], [28], [8], [35], [49], [27], [31], they are still limited [12].

**(3) Rowhammer attack:** Current DRAM chips are designed to be physically denser to increase capacity and reduce energy consumption. Though there are benefits of such a design, it leverages smaller cells that can only hold a lower charge, reducing the noise threshold. At the same time, the smaller distance between DRAM cells causes electromagnetic coupling effects, which means that a small amount of charge is leaked when an adjacent cell is read. Kim *et. al* [32] presents that the frequent read of one DRAM cell can cause bit-flip in a neighbor row, which can be used to build a DRAM-based attack, termed as "rowhammer".

**(4) Faults attack:** employs vulnerabilities of frequency/voltage adjusting mechanisms existing in computer systems and induces faults in program execution. For example, CLKscrew [48] targets the vulnerabilities in power management in ARM devices. Plundervolt [39] targets the voltage scaling feature in Intel processors. They both abuse such flaws to induce faults that can be further utilized to reveal security keys, etc.

Since servers in the cloud use similar architectures, the same vulnerabilities of micro-architectural attacks are also a threat to cloud providers and users [43]. But to initiate side-channel attacks in the cloud, attackers must first achieve co-location with victims. This makes deploying micro-architectural attacks in the cloud an interesting problem since it involves both seizing control of cloud schedulers and exploiting weak points in cloud infrastructure hardware. Our focus in this paper is on the former step. There are off-the-shelf implementation of these attacks hence it is relatively easy to launch micro-architectural attacks once co-location is achieved.

## III. Threat Model

In this study, attackers' goal is to exploit scheduler vulnerabilities and force the scheduler to locate at least one of the attack instances to the same physical node with the victim instance. As introduced in Section I, a number of micro-architectural attacks demand co-location to compromise users' privacy [19] and system availability [29]. This work demonstrates that attackers are able to leverage cloud schedulers to co-locate with target victim applications. We target a feature in filter-score scheduler that is used in popular cloud architectures including Kubernetes [2], OpenStack [5], OpenNebula [4], etc.

Furthermore, we consider cloud providers as trusted service providers and have no additional information about their users; hence, they will consider all users as equivalent and not differentiate users according to their user groups. The schedulers will schedule users' application instances (both malicious and non-malicious) solely according to resource requirements and other specifications. No profiling or behavior analysis about the application will be initiated to detect malicious activities. To achieve the cost-efficiency requirement of the cloud provider in order to be able to generate revenue, application instances from different users can be co-located on the same physical nodes, and containers/VMs provide required isolation mechanisms. Cluster label-value choices are available for all users to provide affinity specifications.

All users in this paper are considered to have access only to their submitted instances and allocated resources, without any privilege to modify the cluster or monitor cluster information. We consider two types of users, namely normal users and malicious users. Normal users submit tasks without malicious intent to fulfill a computational need. Victim application instances are selected from instances submitted by normal users. We assume malicious users have neither collusion with cloud providers nor any privileges above other users. They do not know the scheduling results of application instances of other users. The only operation they can perform is submitting applications to the cloud like a normal user. But we assume the attackers have information about their victims and can hence generate related specifications by guesses or directly referring to the information they have. Since a lot of cloud users run existing commercial or open-source software (database, data analytics app, etc.) in the cloud to process sensitive data, accessing the target victim software and determining a suitable cloud setup for the software shouldn't be a problem. Also, because the next step of co-location is issuing micro-architectural attacks, which also require analyzing target software, this assumption is reasonable.

Deploying applications to the cloud comes with costs. We assume that users are charged according to their assigned resources and the time they run their applications. In this paper, since we focus more on the attacker side and we only evaluate the cost qualitatively, the cost can be further simplified to be proportional to the number of instances launched on the cloud.

Though there will be no performance profiling and complicated behavioral analysis for applications, obviously suspicious user behaviors like submitting a large number of jobs during a small time frame will be noticed and stopped, which renders brute-force attacks infeasible.

## IV. Methodology

In this section, we will first describe the targeted scheduler in this work. Afterward, we present the proposed co-location attack (REPTTACK) in detail. In particular, REPTTACK exploits affinity features to locate victims. Hence, co-location can be achieved more efficiently, i.e. with fewer costs.

### A. Targeted Scheduler

In this paper, we target filter-score schedulers (shown in Figure 1), which are widely deployed by various cloud orchestration systems including OpenStack [5], Kubernetes [2] and OpenNebula [4]. In general, such schedulers are composed of two phases:

1) Filtering: in this step, the scheduler excludes nodes that do not have (enough) required resources or do not match other specifications. After this phase, a list of shortlisted nodes will be passed to the scoring phase.
2) Scoring: in this step, the scheduler tries to find the most suitable node in the list of candidate nodes after the filtering step. It scores each node according to the number of resources available in the node, whether it matches preferences specified by users, and eventually picks the node with the highest score.

The corresponding pseudo-code of this scheduling process is provided in Algorithm 1.

**Algorithm 1** Scheduling.

```
Require: User specifications of the application userspecs, list
  of cluster nodes allnodes.
  candidates=FILTER(userspecs, allnodes)
  bestnode=SCORE(userspecs, candidates)
  return bestnode
  function FILTER(userspecs, allnodes)
      candidates = allnodes
      for node in allnodes do
          for spec in userspecs do
              if spec is not satisfied on node then
                  Delete node from candidates
              end if
          end for
      end for
      return candidates
  end function
  function SCORE(userspecs, candidates)
      score_node = 0 for all node in candidates
      for node in candidates do
          for spec in userspecs do
              if spec is resource specification then
                  Calculate score and add to score_node
              else if spec is satisfied on node then
                  Add a fixed value to score_node
              end if
          end for
      end for
      return bestnode = argmax_node score_node
  end function
```

We conceptually classify user specifications into two categories (as shown in Figure 2): resource specification and affinity specifications [38]. Resource specifications refer to user-specified requirements for resources, including CPU cores, memory space, disk space, etc. On the other hand, affinity specifications [20] refer to user-specified requirements or preferences on nodes in the cluster. There are two types of affinity specifications: node affinity/anti-affinity and inter-application affinity/anti-affinity. By specifying node affinity/anti-affinity, users can force the scheduler to schedule an application to/not to nodes with specific resources or features. By specifying inter-application affinity/anti-affinity, users can choose to co-locate/not co-locate with certain types of applications. In general, all affinity features rely on the label system of the cluster and are specified in the format of label-value pairs. Label-value pairs can be added to the metadata information of nodes and application instances to identify different categories of nodes and applications. For example, labels can be added to nodes to indicate the regions the servers belong to or specific hardware resources available on the nodes. In this work, we focus on the latter type of user specifications, and we argue that the extensive use of such features may result in security problems. Those features are used in some scenarios nowadays, including IOT [44]. Also, similar features appear in other commonly used schedulers like SLURM [59]. We believe such security problems naturally exist in systems with similar attributes.

### B. Attack Strategy

As we described before, there are features in scheduling algorithms that allow users to submit additional requirements and preferences of nodes and applications to be co-located, enabling users to influence the scheduling results. This exposes vulnerabilities and eases the process of achieving co-location, as attackers can have educated guesses on these feature specifications based on the targeted application and mimic these specifications to increase the probability of co-location.

The intuition behind our strategy is as follows: when two tasks share the same requirements and preferences for nodes and applications running on the nodes, replicating requirements given by victim users helps narrow down the search space in the filtering phase hence the search spaces of the scheduling algorithm will be reduced to similar lists of nodes. Replicating preferences helps to force the target node to get the highest score, increasing the co-location probability, especially in a heterogeneous cluster. As long as the number of resource requirements of attack instances can be set to a minimum, contention can be decreased to the lowest level. The attacker can issue multiple attacks to increase the co-location rate he can achieve and utilize anti-affinity features to spread all the attack instances. This is feasible because such schedulers (e.g., Kubernetes scheduler) allow user-defined specified application labels to participate in the scheduling process. Attackers can hence specify dedicated application labels that are only used by themselves to spread the attack instances.

The attack method we propose in this paper is called REPTTACK. An attacker can start by checking the document of the target scheduler, look at the scheduling requirements they can submit, and identify features similar to affinity rules [38], [20] that allow users to specify requirements or preferences on nodes or co-located applications. We assume that the attacker has enough information about the victim application hence can have relatively accurate guesses on the specifications the victim users might provide (regions of nodes, specific hardware resources on nodes, etc.). The attacker then replicates such specifications, limits the resource requirements to the minimum amount required to issue an attack (so that target nodes in the cluster won't be mistakenly filtered out because of resource limitations), and use inter-application anti-affinity (inter-pod anti-affinity in Kubernetes) to spread the multiple attack instances on different machines. The corresponding pseudo-code is shown in Algorithm 2.

Generally, suppose an attacker cannot directly obtain the user-submitted specifications. In that case, the attacker can

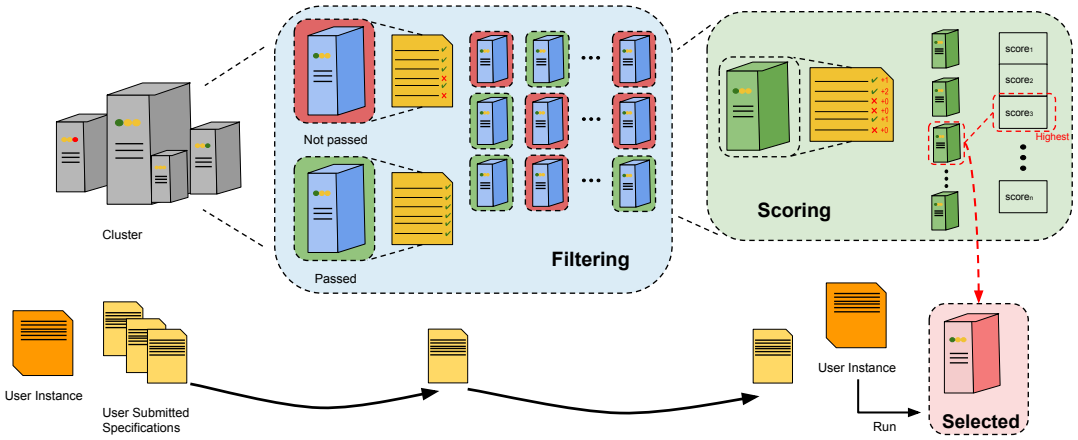



Fig. 1: The diagram of the scheduler we target.

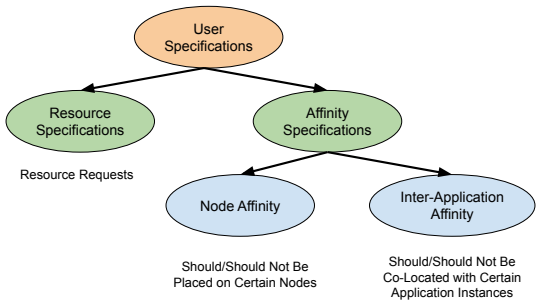



Fig. 2: User specifications.

**Algorithm 2** REPTTACK.

**Require:** User specifications of the victim application $victimspecs$.
$attackspecs$ = {}
**for** $spec$ in $victimspecs$ **do**
  **if** $spec$ is an affinity specification **then**
    $attackspecs = attackspecs \cup \{spec\}$
  **end if**
**end for**
Set all resource requirements to minimum then add to $attackspecs$.

**if** There are multiple attack instances **then**
  Add an anti-affinity spec to $attackspecs$:
    Label = $\kappa$
    Value = $\nu$
    ($\kappa$ and $\nu$ are used only by attackers)
**end if**
**return** $attackspecs$

start by studying the target victim application and determine whether there are any special requirements (e.g., victim application requires extensive GPU resources and requires not being co-located with a specific type of application because of contention, etc.). Location preferences of servers of other users can also be exploited, e.g., in [44] round-trip time information is used to specify labels of machines and used as an affinity feature in a Kubernetes-based cluster.

To increase the co-location success rate, attackers can have multiple versions of attack instances consisting of various user specifications. Compared to brute-force attacks, the cost is still controllable.

## V. EVALUATION

Our experiments consist of behavioral simulation and tests in a real cluster. Due to our limited access to large clusters, part of the experiments are conducted in a scheduler simulator implemented in Python that simulates the behaviors of filter-score schedulers. The simulator gives us the flexibility to test the influence of different parameters. The test on a real cluster is conducted on a public cloud testing environment in which we deploy a 40-node Kubernetes [2] cluster and randomly submit jobs to observe the scheduler outcome of a real cluster. In our experiments, we only consider whether or not the target applications can be co-located.

### A. Experiments in Simulation

*1) Experiment Settings:* The simulator written in Python consists of approximately 1000 LOC. In the simulation process, configurations of machines in the cluster are generated randomly. There are three phases in the simulation process: node setup, application generation, and execution simulation.

For node settings, we specify the resource capacities (number of CPU cores, memory space, storage space, network ports) of every node, as well as the labels of each node. In our simulation, the number of nodes in the cluster is 100. In the execution phase, we split the whole experiment into 1000 slots (scheduling cycles). In each scheduling cycle, we submit 10 random applications to the scheduling queue. The total number of labels used in our experiments is 11, including 5 as node labels, 5 as application labels, and 1 used as spreading labels for attackers to spread attack instances in the cluster.

Application requirements submitted to the system are randomly generated by the simulator. Besides required resources to run the application, labels of the applications and

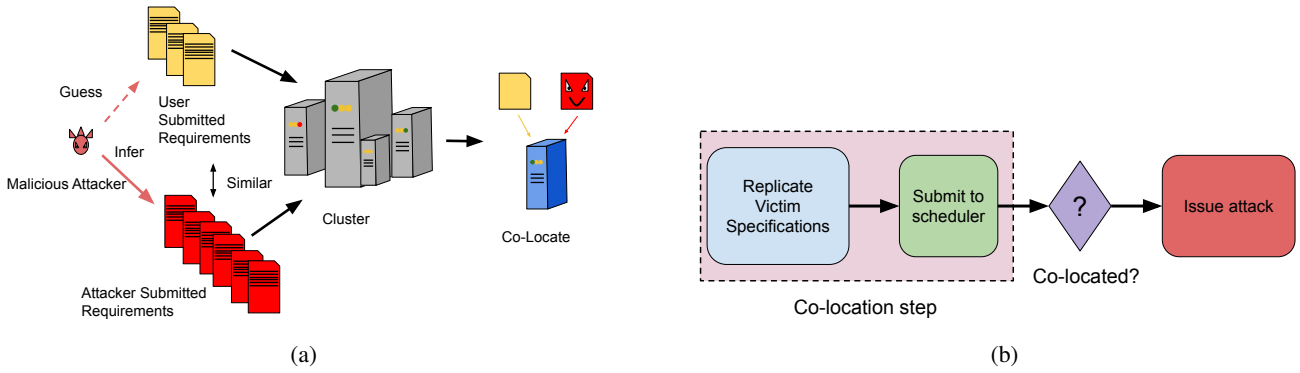



Fig. 3: Diagram of our attack strategy.

scheduling requirements and preferences options are generated randomly. When generating labels for nodes/applications, we follow the following procedure. Assume there are $n$ labels for nodes/applications. We allow some labels to be missing in the configuration, and the probability of having a single label is $p_m$. To generate all label specifications, we examine the $n$ labels one by one, and for each label, with a probability $p_m$ we randomly assign a value to that label. Otherwise, we set the label as missing. The process of generating affinity specifications is similar.

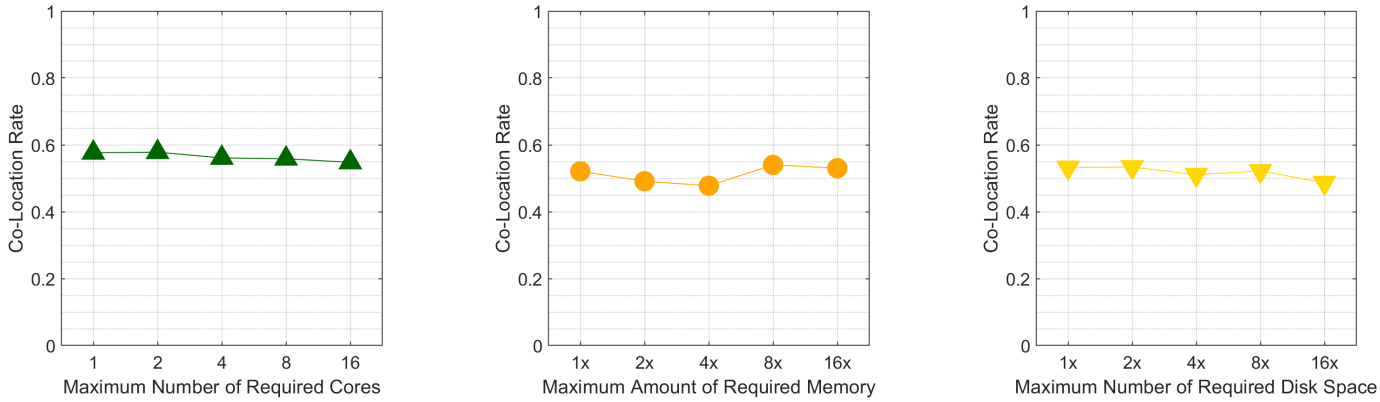
After that, we enter the execution phase, where we submit various applications, including normal tasks and malicious tasks that aim to accomplish co-location with some of the normal tasks. The execution phase is implemented synchronously, meaning that we divide the whole experiment into a large number of slots and will push several applications to the scheduling queue, make scheduling decisions and collect scheduling results in each time slot. Thousands of tasks will be provided to the scheduler, including victim tasks, attack instances, and randomly generated unrelated applications that aim to mimic the assumed unpredictable user behaviors in the system. The goal of each malicious application instance is to co-locate with its target victim instance. We repeat this attack during the execution phase and target different victims each time. After the execution phase, we calculate the co-location rate and provide corresponding results.

*2) Influence of User Specifications:* We first examine how resource specifications can affect the co-location rate of REPTTACK. We evaluate the co-location rate under different amounts of resources (CPU cores, memory, disk space) that users can request and provide the results in Figure 4. Figure 4a, Figure 4b and Figure 4c show the co-location rate under different maximum limitation settings for the number of CPU cores, amount of memory space, and amount of disk space respectively where 1x= 512MB for memory and 1x= 16MB for disk. The amount of user-requested resources follow a uniform distribution between 0 and the given maximum value. Though some scheduling algorithms (e.g., Kubernetes [2]) do not directly check disk space, we still do that in our simulations in case future scheduler designs take disk space into consideration. As shown in all results, these resource specifications will not influence the co-location rate, as the required resources change 16x the change in co-location rate is no more than $5\%$. Therefore we will focus on affinity features in our later experiments.

Figure 5 presents the influence of specifications of node affinity/anti-affinity as well as inter-application affinity/anti-affinity on co-location rate. In this experiment, we only issue one attack instance each time. As we mentioned before, we use the probability $p_m(0 \leq p_m \leq 1)$ to control the number of affinity features used in the experiment. When we have $n$ labels for specifying, the mean number of label specifications that will be considered during scheduling is $np_m$. We denote the $p_m$ used to generate node affinity/anti-affinity specifications as $p_{m_n}$ and the $p_m$ used to generate inter-application affinity/anti-affinity specifications as $p_{m_a}$. Increasing the number of node-affinity/anti-affinity and inter-application affinity/anti-affinity specifications submitted by users will both raise the co-location rate malicious attackers can achieve. As shown in Figure 5, within each group of bars which share the same $p_{m_n}$, increasing $p_{m_a}$ generally increases the co-location rate. Comparing bars with the same $p_{m_n}$ for node-affinity/anti-affinity features, we have the same observation regarding the influence of $p_{m_a}$. Despite some exceptions caused by randomness in the experiments, the trend well matches our expectation that adding more affinity specifications to control the output of schedulers will narrow down the search space of nodes and cause the scheduling results more foreseeable, which attackers can exploit to achieve co-location.

*3) Influence of Number of Attack Instances:* Instead of launching only one attack instance at a time, malicious users can submit multiple instances to the scheduler to improve the co-location rate. Previously, in [43] the authors proposed to use a brute force strategy to spread their attack program across the cloud and achieve co-location. Han *et.al* [23] further formalizes the problem and proves the lower bound of the number of VMs that should be submitted to achieve co-location. However, submitting a substantial number of applications to the scheduler within a small time slot will be easy to detect, and the cost will be unacceptable in some circumstances. We present in this part that, by using REPTTACK, malicious users can achieve a relatively high co-location rate among around 4000 victim instances in a 100-node cluster with affinity features used by users.

Figure 6 shows the simulation results of co-location rates that a malicious attacker can achieve by issuing different num-




(a) Varying the number of required CPU cores.

(b) Varying the amount of required memory.

(c) Varying the amount of required disk space.

Fig. 4: The influence of resource specifications.

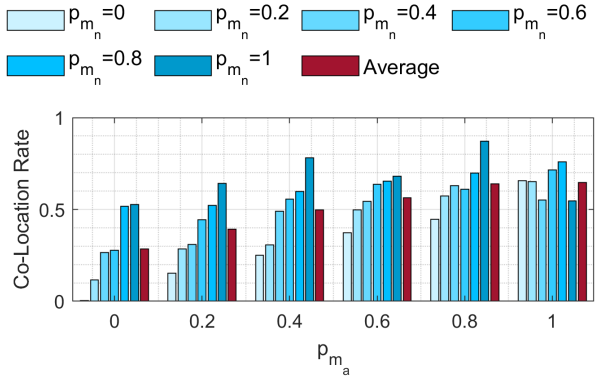



Fig. 5: Influence of node-affinity and inter-application affinity features.

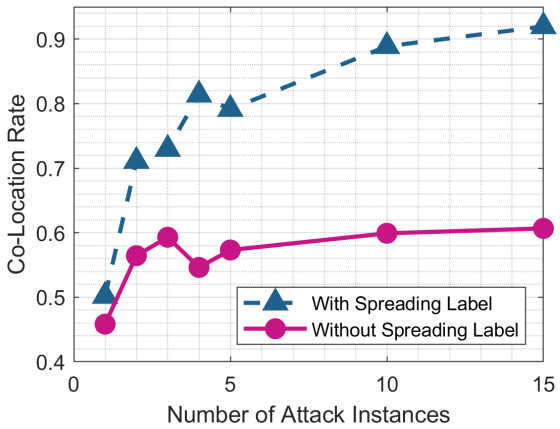



Fig. 6: The influence of the number of issued attack instance.

bers of attack instances. Firstly, we directly submit multiple attack instances to the scheduler following the strategy in Section IV but without utilizing the inter-application anti-affinity to spread all attack instances. The result (the red line in Figure 6) shows that the co-location rate does not improve much, which matches our expectation. The reason is that since multiple attack instances may be co-located on the same node that the victim does not reside in hence will not show significant advantages over the single-issue attack. Next, we demonstrate how involving anti-affinity features to spread attack instances improves the co-location rate. We reserve a special label for the use of attackers in our simulation called spreading label and specify inter-application anti-affinity to prevent attack instances to locate together. As shown in the blue line in Figure 6, the co-location rate grows as the number of attack instances increases, and with more than 1 attack instances issued, the co-location rate reaches around $20\% - 30\%$ higher than without using a spreading label.

### B. Experiments in Cluster

*1) Experiment Settings:* To validate the results of our simulator, we also conduct the experiments in a real cloud. We test related features in a Kubernetes [2] cluster deployed on CloudLab [18]. CloudLab provides cloud researchers control and access to real cluster hardware, even down to bare-metal level. Since we run experiments on top of the Kubernetes cluster setup on CloudLab's hardware and only interact with Kubernetes, we believe our settings are realistic. Also, since there are other cloud schedulers using algorithms similar to Kubernetes, and we do not rely on certain attributes of Kubernetes, the results collected on this cluster are representative.

CloudLab offers users dedicated ready-to-run clusters, with all required settings contained in a cluster profile. Kubernetes uses a filter-score scheduling strategy and is susceptible to the manipulation of scheduling results by submitting some affinity requirements or preferences. We deploy a cluster with *k8s* profile and directly interact with the cluster using Kubernetes command-line tool `kubectl`. The cluster in which we deploy our experiments consists of 40 nodes.

At first, we use the command `kubectl label nodes <nodename> <label>=<value>` to label each machine with some randomly generated configuration information. These configurations include CPU type, GPU type, disk type, and memory capacity ('high' or 'low' similar to [44]). After that, we randomly generate `.yaml` files describing different pods (in Kubernetes, user instances are encapsulated in containers and deployed in the unit of pods, which normally contain one container per pod). The container type of each non-malicious job is randomly chosen from popularly used containers from Docker Hub [1]. Node-affinity/anti-affinity as well as inter-pod affinity/anti-affinity are also generated in each `.yaml` file. After using the command `kubectl apply -f <jobname.yaml>` to deploy and schedule all pods, we run a script to test whether each target victim-attacker pod pair is co-located and report the co-location rate. We assume attackers target all the separate victims one at a time and issue corresponding attack instances. Launching attack instances to achieve co-location with a victim is defined as an attack. The testing script only considers pods that are successfully deployed and calculates the ratio of successful attacks. In this part, we aim to validate that the trends are the same as shown in simulation results.

*2) Influence of Affinity Features:* We first examine whether or not simply duplicating user specifications increases the co-location rate in the cloud. In each test of our experiment, we assume all users provide the same number of affinity specifications. We divide all affinity features into four categories: required node affinity/anti-affinity specifications, preferred node affinity/anti-affinity specifications, required pod affinity/anti-affinity specifications, preferred pod affinity/anti-affinity specifications. We use 4-digit numbers to differentiate the tests, and each digit denotes the number of labels specified in that category. For example, '2131' refers to a test where all submitted pods have: 2 required node affinity/anti-affinity specifications, 1 preferred node affinity/anti-affinity specification, 3 required pod affinity/anti-affinity specifications and 1 preferred pod affinity/anti-affinity specification. In every test of this experiment, we deploy a total number of 800 randomly generated pods, where 200 is considered victims.

As shown in Figure 7, the co-location rate increases as the users use more affinity features. By only issuing a single attack instance, the co-location rate can achieve as high as 60%. This matches our results in our simulator, showing that the risk indeed exists in real clusters. This result again showcases how affinity features can be dangerous if attackers exploit them to achieve co-location.

*3) Issuing Multiple Attack Instances:* After constructing a Kubernetes cloud in CloudLab platform, we also test the influence of the number of attack instances issued by malicious users. The results are shown in Figure 8.

In this experiment, we vary the number of attack instances a malicious attacker can issue as well as several other settings and deploy all the non-malicious applications and malicious attack instances to the cloud in a single experiment. We assume attackers utilize a special label-value pair to mark their malicious instances and exploit the pod anti-affinity feature to avoid the same attack instances to locate on the same machine. From all results shown in Figure 8, we can see that as the number of attack instances the attacker launches increases, at first, there is a significant gain in co-location rate an attacker can achieve. However, after the number of attack instances reaches a certain value, increasing the number of attack instances does not provide an obvious gain in co-location rate anymore. The curve of the co-location rate shows an interesting roofline pattern.

To further explore what parameters may affect the roofline model, we vary different parameters and show how the co-location rate curve is affected. In Figure 8a, we vary the number of affinity features used in the way similar to Section V-B2. As shown in Figure 8a, with more affinity features used by users,

1) the initial co-location rate is higher;
2) the growth of co-location rate as the number of attack instances increases is more rapid;
3) the saturation point is reached earlier.

It is worth noting that the curve '0000' can be considered as the results under a brute-force attack setting, since no affinity features are utilized here. As a result, the initial co-location rate is the lowest, and it costs more instances to reach a similar co-location rate.

In the following 3 experiments, affinity features are fixed to 3020 pattern to provide suitable lengths of growing stage and steady stage. In the experiment of Figure 8b, we test the co-location rate on two sets of containers. Set 1 consists of Traefik, Ngnix, Tomcat, Redis, Mongo, and Wordpress. Set 2 consists of Alpine, Busybox, Python, Registry, Httpd, and Memcached. In the experiments of Figure 8c and 8d we vary the application types, the number of targets in the experiment, and the total number of applications, respectively. As shown in the results, varying these parameters lead to similar co-location rates under different attack instance numbers. These parameters are hence considered not to influence co-location rate.

The trend can be explained as follows. Since attackers specify anti-affinity requirements when they submit their tasks in order to spread their attack instances across the entire cluster, the increase in co-location rate at first is because more qualified nodes after the filtering step are occupied when the number of attack instances increases. When victims use more affinity features, the search space narrows down; hence the co-location rate will grow faster as there are fewer qualified nodes. The co-location rates reach the saturation stage after the number of attack instances passes a specific number because nearly all possible nodes are occupied. However, the later steady stage also demonstrates that increasing the number of attack instances cannot cover all possible nodes, and there will still be victims escaping from such attacks. This phenomenon can be possibly caused by the variations of cluster states (resource usage in each node, etc.) during different scheduling processes since all specifications of the attack instances are the same as victims. Therefore, when the number of attack instances exceeds a certain number, the co-location rate stays at a relatively high value but will not reach 100%.

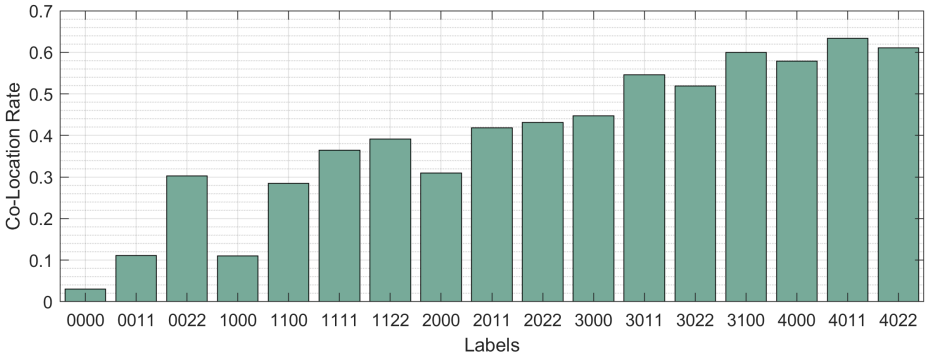
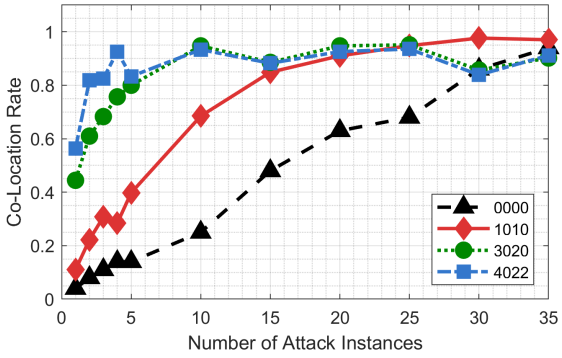
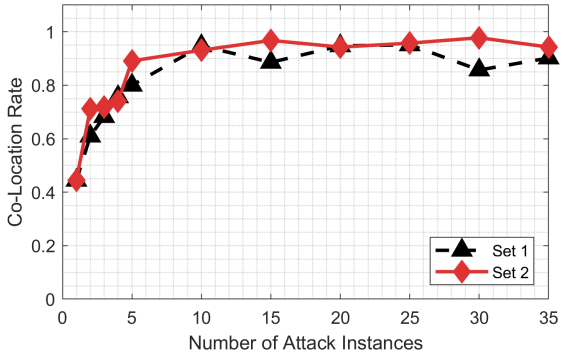


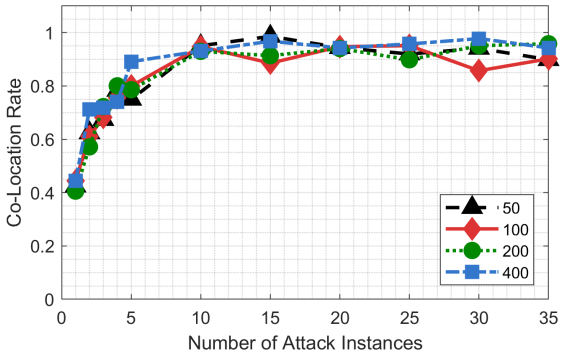

Fig. 7: The influence of affinity features on co-location rate.



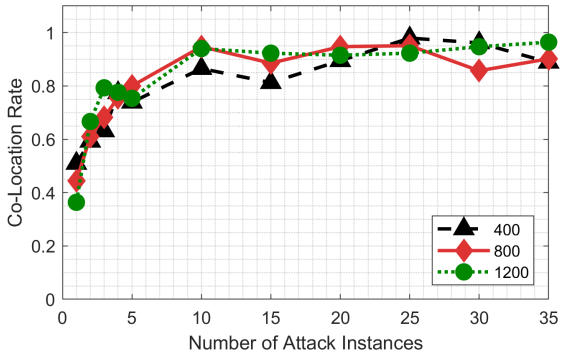

(a) Varying number of affinity features used.




(b) Varying application types.




(c) Varying the number of targets.




(d) Varying the total number of deployed applications.

Fig. 8: Influence of number of issued attack instances.

## VI. Mitigation

### A. Can Migration Defend Against Co-Location Attacks?

In some previous research works in this area [62], migration of application instances in the cloud is used as a defense method. By moving application instances occasionally, the difficulty of achieving co-location will increase for attackers. However, in this part, we argue that under the assumption that migration still respects user requirements, the risk of being co-located with malicious instances is potentially higher.

Co-location rate in our experiment is defined as

$$\text{Co-Location Rate} = \frac{\text{Successfully Co-Located Attacks}}{\text{Total Number of Attacks}}.$$

The count of successful attacks, however, only considers the initial placement. When migration is introduced, attack instances that are previously considered successful may not co-locate with victim instances anymore, but new co-located attack-victim pairs may appear as well. We hence modify our definition of a successful attack. An attack attempt is considered successful when the target victim application is located on nodes with attack instances in $t\%$ of its lifetime, where $t$ is

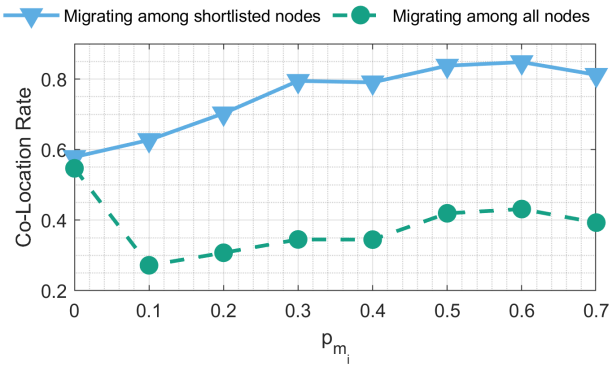



Fig. 9: Co-location rate results in our simulator after introducing migration.

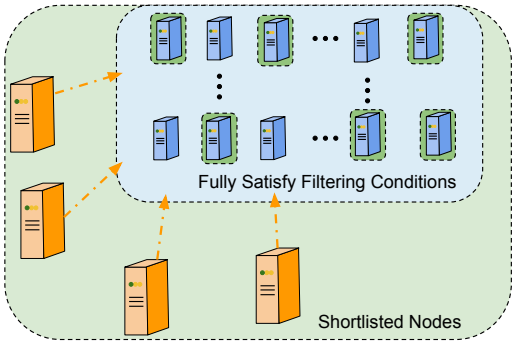



Fig. 10: The diagram of our mitigation strategy.

a predefined threshold percentage. In our experiments, we set it to 80, so when an application is co-located with an attack instance targeting itself for more than $80\%$ of its lifetime, we consider the corresponding attack a successful one. Attacks like performance degradation attack [7], [50] can benefit from longer co-location time.

The migration process in our experiment is implemented as follows. At the end of each time slot, we examine every application, and with a probability $p_{m_i}(0 \leq p_{m_i} \leq 1)$, we remove it from its current node and place it on another node. There are two methods to select migration destinations:

1) Randomly selects one node from the shortlist of nodes after the filtering step.
2) Randomly selects one node from the cluster.

The results are presented in Figure 9. In these experiments, we vary $p_{m_i}$ to control how frequently instances in the system are being migrated. The first selection method respects user-specified requirements since the migration destination is chosen from the filtered shortlist. The second selection method brings more randomness, and as a trade-off, the user requirements can be violated shortly after its initial placement.

When the migrating destination selection is based on the shortlist of nodes after the filtering step, as we migrate more frequently, the co-location rate increases, which contradicts our intuition. This can be explained as the result of more collisions between the attack and victim instances in our experiment. This results in the victim instance being co-located with the attacker longer in time, giving rise to more successful attack attempts. When we expand the selection to all nodes in the cluster, there is indeed a drop in the co-location rate. However, we can observe a decrease in defense effect as $p_{mi}$ approaches $50\%$ because of the increase in the probability of collisions between victims and attack instances after adding migrations.

### B. Mitigation Strategy

The affinity features make schedulers' outcomes more predictable and can hence be manipulated by malicious attackers. To defend against REPTTACK, a straightforward method is to re-introduce randomness into the scheduling process. To this end, we manually insert nodes that do not pass the filtering phase of the scheduling process to the scoring candidate lists. This comes with violations of user requirements, but according to our later results, the violation is not severe.

The proposed mitigation method is to randomly skip label check in the filtering step of scheduling with a specified probability $p_s(0 \leq p_s \leq 1)$. This can be considered loosening the filtering conditions and manually increasing the number of available nodes to schedule user applications. This intentional violation of user specifications comes with the risk of violating user-specified requirements and can possibly cause performance degradation. Still, according to our results in this section the sacrifice is relatively acceptable compared to the benefits it brings in defending against REPTTACK. We will present in later that with only a relatively small sacrifice of violating user requirements, the co-location rate of attacks is significantly reduced. The performance degradation can be interpreted as the cost for security.

The detailed pseudo-code of our mitigation method is shown in Algorithm 3. We introduce a simple SKIP function to help determine whether or not a label checking step should be omitted, and use SKIP function to modify the FILTER function in Algorithm 1. As can be seen from Algorithm 3, this mitigation method is relatively easy to integrate into existing systems, and there is a parameter $(p_s)$ that can be adjusted to control the level of defense in the system.

It is worth noting that scheduler designers should be careful about skipping user-specified label checks since simply ignoring some of them will cause the application to be scheduled to nodes without necessary hardware or system settings. The choices of labels to skip ought to be limited to those only affecting performance to guarantee correct execution. One example is if the application requires a special type of hardware architecture, this specification cannot be ignored.

### C. Effects of Mitigation Strategy

We test our mitigation strategy in simulation and present the results in Figure 11. Since it is hard to model the actual performance as it varies according to cluster settings and it is highly related to the heterogeneity of the target cluster, we use a metric called affinity satisfaction to quantify our sacrifice in performance. Since a lot of these affinity specifications are provided by users for the sake of performance, and our strategy is to ignore part of these specifications randomly, this metric can reflect the trade-off between performance and security.

**Algorithm 3** Mitigation strategy.

```
function SKIP(p_s)
    Generate a 1 bit flag. flag is 1 with probability p_s.
    if flag = 1 then
        return True
    else
        return False
    end if
end function
function NEWFILTER(userspecs, allnodes, p_s)
    candidates = allnodes
    for node in allnodes do
        for spec in userspecs do
            if spec is a resource specification
            or SKIP(p_s) == False then
                if spec is not satisfied on node then
                    Delete node from candidates
                end if
            end if
        end for
    end for
    return candidates
end function
```

After making a scheduling decision, we re-run the filtering check on the node the task is assigned to and record whether all affinity requirements are met. After the entire experiment finishes, we calculate the percentage of applications assigned to a node that fully satisfies all their required affinity and use that as our metric.

Figure 11a and Figure 11b show how our mitigation strategy reduces the co-location risk and the sacrifice it brings. We vary $p_s$, the probability of skipping affinity check of a label and re-run the simulation in Section V-A3. In Figure 11a, with $p_s = 5\%$ the co-location rate drops from $50\%$ to around $10\%$, while still keeping approximately $55\%$ of scheduling results that fully satisfy user needs. Please note that our metric only measures whether all requirements that aim to enhance performance are fully satisfied. As shown in Table I, in a lot of the cases, there may be only a limited number of affinity specifications violated; hence the applications will still be able to have most of their requirements met. In Figure 11b we simulate an environment that users use affinity features more extensively. Comparing to Figure 11a, we conclude that when users utilize more such features, the co-location rate and the affinity satisfaction metric both become more sensitive to our mitigation strategy. This is easy to understand as well. The scheduler outputs rely more on user specifications as users use those features more extensively, but our strategy randomly forces the scheduler to give up user specifications in the filtering step. Therefore more user specifications will be violated as $p_s$ increases, resulting in a quicker decrease in co-location rate as well as affinity satisfaction metric.

In Figure 11c and Figure 11d, we compare co-location rates before and after involving our mitigation method on attacks with different numbers of attack instances and provide the affinity satisfaction scores, respectively. Here, all attack instances are submitted with the spreading label to increase the co-location rate. As shown in Figure 11c, with $p_s = 2\%$, our mitigation strategy reduces the co-location rate by at least around $15\%$. However, as the number of attack instances increases, the reduction becomes less significant. According to the result of Figure 11d, the affinity satisfaction metric does not change much as the number of attack instances increases. This proves that the metric of affinity satisfaction is only the function of $p_s$. With the results in Figure 11, we conclude that bringing randomness to the scheduling process helps resist co-location attacks. But to defend against malicious attackers who issue multiple attack instances at a time, we will need to raise up $p_s$ to achieve the same reduction in co-location rate.

According to the experimental results, the sacrifice on affinity matching is not significant. Also, since all basic resource requirements are guaranteed to meet because we don't change the CPU and memory allocation strategy, and our mitigation strategy only violates a relatively small number of affinity requirements, we believe the performance degradation is not much.

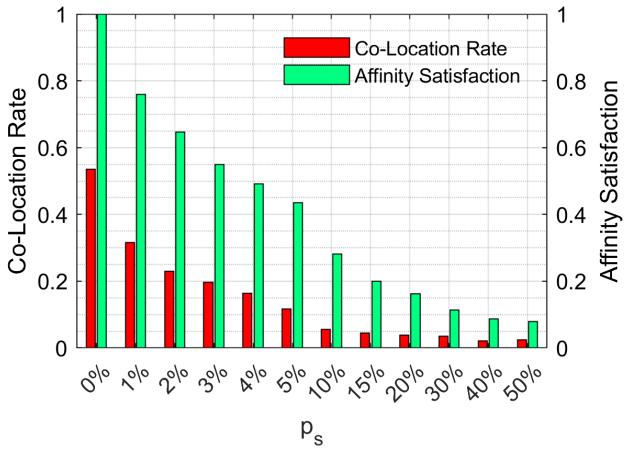
## VII. Discussion

According to our experimental results in Section V, security problems mainly arise from user specifications that aim to fine-tune the scheduling decisions by specifying preferences or requirements on nodes and accompanying applications. In heterogeneous clusters, giving users access to control the output of the schedulers is possible to expose such security flaws to co-location attacks. These specifications help narrow down the search space of available nodes in the filtering step and force the scheduler to score the target machine with higher scores. In previous research works, co-location is achieved via issuing a large number of attacks, which is relatively easy to detect and prevent. When enabling user specifications in scheduling, by only issuing a few attack instances, the co-location rate is relatively high. From the results shown in Section V, we have summarized guidelines for attackers as well as regular users and cluster managers as detailed below.

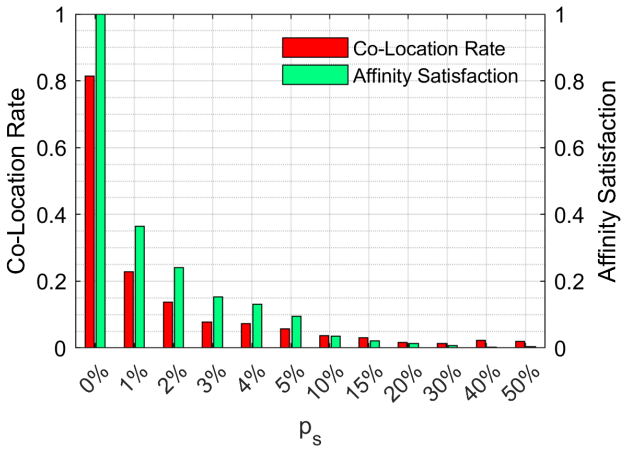
For attackers, the first thing they should do is to study the target applications and have a guess on the type of requirements the victim user will specify. To increase coverage, a malicious user can issue multiple versions of their attack applications, which cover different possible settings that users might provide. As for the number of attack instances they should launch, according to our experimental results in Section V there is a best trade-off point between co-location accuracy and cost (as shown in Figure 12). Before reaching this point, increasing the number of attack instances will raise the co-location rate. In contrast, beyond that point, issuing more attack instances (more cost) for a single target brings a relatively low gain in co-location rate.

For users, it is highly recommended that they use as few affinity features as possible to avoid predictable, mimicable scheduling results. Also, their specifications on their applications should be kept secret and avoid being leaked to malicious attackers. The co-location rate would be very low if few such specifications are copied by attackers.

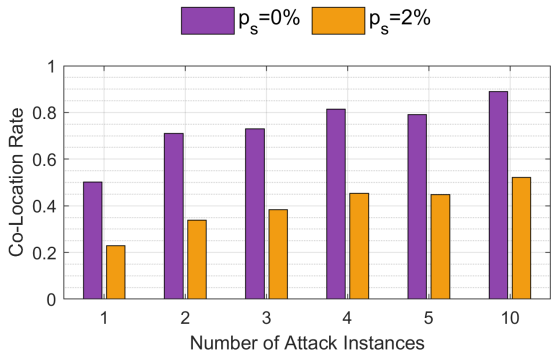
For cluster managers, the key to defend against co-location attacks issued by malicious users is to bring randomness into the scheduling process and provide users with limited control over the scheduling results. Randomness may violate service



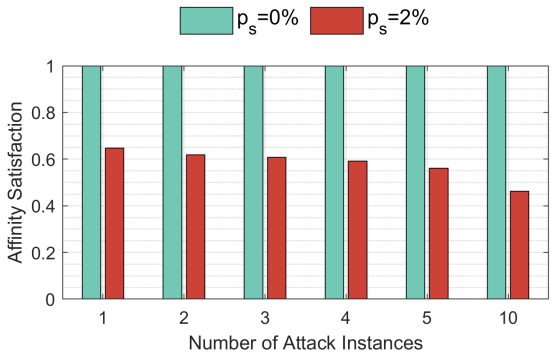

(a) $p_{mn} = 0.5$, $p_{ma} = 0.5$.




(b) $p_{mn} = 0.9$, $p_{ma} = 0.9$.




(c) Co-location rates before and after our mitigation strategy is applied to attacks with different number of attack instances. $p_s$ is set to $2\%$.




(d) Affinity satisfaction scores in the experiments.

Fig. 11: Co-location results after introducing migration.

TABLE I: The average number of violated affinity specifications. In our experiments, since 10 effective labels are used (excluding the one used as spreading label for attackers), the average number of affinity specifications for $p_{m_n} = p_{m_a} = 0.5$ is 5 and the average number of affinity specifications for $p_{m_n} = p_{m_a} = 0.9$ is 9.

| $p_{m_n}, p_{m_a}$ | $p_s = 0\%$ | $p_s = 1\%$ | $p_s = 2\%$ | $p_s = 3\%$ | $p_s = 4\%$ | $p_s = 5\%$ | $p_s = 10\%$ | $p_s = 15\%$ | $p_s = 20\%$ |
|---|---|---|---|---|---|---|---|---|---|
| 0.5 | 0.00 | 0.45 | 0.68 | 0.88 | 1.07 | 1.19 | 1.68 | 2.00 | 2.17 |
| 0.9 | 0.00 | 1.65 | 2.29 | 2.78 | 3.02 | 3.33 | 4.04 | 4.40 | 4.57 |

level agreement (SLA); hence, it is crucial for cloud managers to make the trade-off between security and user specification satisfaction. Also, when deploying their clusters and labeling all servers in the cluster, heterogeneity should be exposed to users as little as possible, making it harder for users to identify the nodes their applications are running on.

In some recent studies [15], [6], [53], instead of directly giving users the ability to control schedulers' outputs, the schedulers try to profile the applications users submit and use machine learning techniques to learn and find the best resource configuration and determine the best node for the application to run. The use of such adaptive techniques enhances the quality of scheduling, as the deployment of user applications is specially optimized. This, however, may also trigger some security problems since there are also methods to generate fake traces and trick the scheduler. By simply mimicking the behaviors of victim applications, the malicious attackers are also able to trick the scheduler of a heterogeneous cluster and get co-located with the victim application, according to some simulation results. Similar to our study, this problem occurs when schedulers try to find the most suitable machine according to the execution patterns, which inevitably narrows down the search space and greatly increases the chance of co-location.

The conflict between performance and the danger of co-location attack reveals a trade-off in scheduler design. Accord-

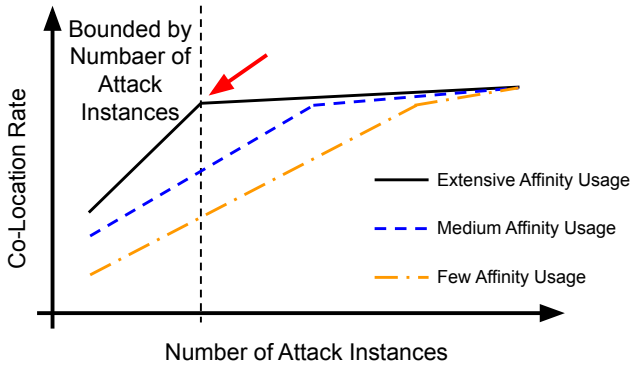



Fig. 12: The roofline model of the influence of number of attack instances.

ing to the results in this paper, to provide security, randomness is a necessity; otherwise, once attackers obtain enough information about victims, co-location can be achieved with a high success rate. Randomness can be introduced explicitly (e.g., our method in Section VI) or inexplicitly (e.g., by limiting the scheduler to consider only the current capacity of machines in the cluster because this information is not predictable by users). Our next step is to construct and test a scheduler with learning capabilities in a real large cluster and try to develop some mechanisms to provide secure scheduling results that also guarantee the performance of user applications.

## VIII. Related Work

### A. Affinity Policies

Affinity policies have been utilized in cloud systems to constrain instance placement and improve the scheduling quality. In 2012, a system called VMware's Distributed Resource Scheduler (DRS) [20] was proposed. DRS utilizes affinity policies to constrain placement of VMs and provide appropriate recommendations of nodes to users. The affinity policies in [20] are called VM-to-Host rules and VM-to-VM rules, which correspond to the node affinity rules and inter-application affinity rules in this paper. In 2018, Moreno-Vozmediano *et.al* [38] implement a cloud orchestration system with new affinity policies like role-to-role rules and VM-to-location rules to address challenges in deploying high availability services. According to the analysis in this paper REPTTACK can also exploit such features. Affinity rules widely exist in nowadays popular cloud architectures like Kubernetes [2], OpenStack [5] and OpenNebula [4].

### B. Co-Location Attack

The concept of co-location attack in the cloud was proposed and discussed in [43]. The authors analyze the risks of side-channel attacks by being co-located with malicious VMs in the public cloud. The paper also introduces the technique of network probing to determine whether the attack instances are co-located with victim instances, and the method the authors use to achieve co-location is issuing multiple attack VMs by brute force or exploit the scheduling locality regarding timing. Following this research, there are various works on the side-channel attack part [26], [61], [13] as well as the co-location detection part [60], [25], [47], [24]. Regarding the co-location step, researchers normally utilize brute-force issue of attack instances [23], [43] or exploit the scheduling locality in time to spread attack instances in the cloud. The paper [23] formalizes this question and provides the expression of minimum VMs the attacker needs to issue under different scheduling patterns. The results shown in the paper [51] is relevant to ours, where the authors examine the effects of different factors on co-location rates on commercial cloud providers.

Researchers have studied mitigation methods to prevent co-location. [9] provides a simple algorithm that brings randomness to the scheduling process and mathematically proves the reduction in security risk. [22] employs a machine-learning-based method to identify malicious user patterns. [62] presents the usage of migration of VMs to reduce co-location risk. [16] proposed to add perturbations that prevent co-location detection attempts.

A concurrent work proposed Cloak & Co-locate attack [37], which targets profiling-based schedulers to achieve co-location by providing misleading execution traces. However, to the best of our knowledge, such schedulers were only studied in the research community [15], [14] and is not currently deployed by the industry. Furthermore, our work targets the widely deployed filter-score based schedulers and achieves co-location by replicating user specified constraints.

To the best of our knowledge, we are the first to explore the security risks of allowing users to submit scheduling specifications along with the instances to the cloud. Also, when these features are utilized, the mitigation methods may not be as effective since the scheduler still obeys the rules the users provide.

## IX. Conclusion

In this paper, we present that the filter-score scheduler is prone to suffer from security problems because of the use of the existing affinity features. We find that, if the affinity features are utilized by users in the system, by some simple guesses according to the application information or obtaining the submitted configuration files, the attacker can relatively accurately locate the victims to achieve co-location and proceed to issue side-channel attacks. We propose a simple attack strategy called REPTTACK and a corresponding mitigation method to defend against it. The effectiveness of REPTTACK, as well as the proposed mitigation method, are supported by our experimental results in either the CloudLab cluster or simulation. The findings in this paper provide several guidelines regarding security for attackers, users, and, most importantly, cluster managers or designers. From the perspective of cluster scheduler designers, for safety concerns, we advise that the cluster information should be kept secret, and scheduling decisions should be made with user interference at the least level, where there is a trade-off between security and performance. Profiling applications and utilize machine learning techniques to determine the best configuration for an application can be one method to hide details of clusters yet still keep performance, but risks still exist since the behaviors of an application can be mimicked, and the scheduler can be tricked. Future work will be dedicated to how micro-architectural attacks can be initiated on the cloud and how to design schedulers that can reach a well-balanced point.

## ACKNOWLEDGMENT

The authors would like to thank the support of High Performance Computing (HPC) Center of University of California, Davis, and Mr. Sergey Buduchin, the technical director of HPC Center. We are also grateful to our shepherd Mauro Conti and all anonymous reviewers for their constructive feedback.

## REFERENCES

[1] "Docker Hub," https://hub.docker.com/, 2021, [Online; accessed 1 May 2021].

[2] "Kubernetes," https://kubernetes.io/, 2021, [Online; accessed 1 May 2021].

[3] "Meltdown and Spectre," https://meltdownattack.com/, 2021, [Online; accessed 1 May 2021].

[4] "OpenNebula," https://opennebula.io/, 2021, [Online; accessed 1 May 2021].

[5] "OpenStack," https://www.openstack.org/, 2021, [Online; accessed 1 May 2021].

[6] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 469–482.

[7] T. Allan, B. B. Brumley, K. Falkner, J. Van de Pol, and Y. Yarom, "Amplifying side channels through performance degradation," in *Proceedings of the Annual Conference on Computer Security Applications (ACSAC)*, 2016, pp. 422–435.

[8] ARM, "Cache Speculation Side-channels," 2018.

[9] Y. Azar, S. Kamara, I. Menache, M. Raykova, and B. Shepard, "Co-location-resistant clouds," in *Proceedings of the ACM Workshop on Cloud Computing Security (CCSW)*, 2014, pp. 9–20.

[10] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proceedings of theUSENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014, pp. 285–300.

[11] C. Canella, K. N. Khasawneh, and D. Gruss, "The evolution of transient-execution attacks," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2020, pp. 163–168.

[12] C. Canella, S. M. Pudukotai Dinakarrao, D. Gruss, and K. N. Khasawneh, "Evolution of defenses against transient-execution attacks," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2020, pp. 169–174.

[13] C. Cardenas and R. V. Boppana, "Detection and mitigation of performance attacks in multi-tenant cloud computing," in *Proceedings of the International IBM Cloud Academy Conference (ICACON)*, 2012, p. 48.

[14] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 77–88, 2013.

[15] ——, "Quasar: Resource-efficient and qos-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, 2014.

[16] A. Dhavlle, S. Rafatirad, K. Khasawneh, H. Homayoun, and S. M. P. Dinakarrao, "Imitating functional operations for mitigating side-channel leakage," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[17] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+ abort: A timer-free high-precision l3 cache attack using intel TSX," in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2017, pp. 51–67.

[18] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19

[19] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: a fast and stealthy cache attack," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 2016, pp. 279–299.

[20] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "Vmware distributed resource management: Design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, 2012.

[21] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich *et al.*, "Protean:VM allocation service at scale," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020, pp. 845–861.

[22] J. Han, W. Zang, M. Yu, and R. Sandhu, "Quantify co-residency risks in the cloud through deep learning," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[23] Y. Han, J. Chan, T. Alpcan, and C. Leckie, "Using virtual machine allocation policies to defend against co-resident attacks in cloud computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 95–108, 2015.

[24] M. S. Inci, B. Gülmezoglu, G. I. Apecechea, T. Eisenbarth, and B. Sunar, "Seriously, get off my cloud! cross-vm rsa key recovery in a public cloud." *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 898, 2015.

[25] M. S. Inci, B. Gulmezoglu, T. Eisenbarth, and B. Sunar, "Co-location detection on the cloud," in *Proceedings of the International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*. Springer, 2016, pp. 19–34.

[26] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 2016, pp. 368–388.

[27] Intel, "Retpoline: A Branch Target Injection Mitigation. Revision," 2018.

[28] M. S. Islam, A. P. Kuruvila, K. Basu, and K. N. Khasawneh, "Ndhmds: Non-differentiable hardware malware detectors against evasive transient execution attacks," in *Proceedings of the IEEE International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 537–544.

[29] Y. Jang, J. Lee, S. Lee, and T. Kim, "Sgx-bomb: Locking down the processor via rowhammer attack," in *Proceedings of the Workshop on System Software for Trusted Execution*, 2017, pp. 1–6.

[30] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga, "Mercury: Hybrid centralized and distributed scheduling in large shared clusters," in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, 2015, pp. 485–497.

[31] K. N. Khasawneh, E. M. Koruyeh, C. Song, D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Safespec: Banishing the spectre of a meltdown with leakage-free speculation," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[32] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[33] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.

[34] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "Spectre returns! speculation attacks using the return stack buffer," in *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, 2018.

[35] E. M. Koruyeh, S. H. A. Shirazi, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "Speccfi: Mitigating spectre attacks using cfi informed speculation," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 39–53.

[36] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin *et al.*, "Meltdown: Reading kernel memory from user space," in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2018, pp. 973–990.

[37] H. M. Makrani, H. Sayadi, N. Nazari, A. Sasan, K. N. Khasawneh, S. Rafatirad, and H. Homayoun, "Cloak & co-locate: Adversarial railroading of resource sharing-based attacks on the cloud," in *Proceed-*

*ings of the International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2021, p. 1.

[38] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Orchestrating the deployment of high availability services on multi-zone and multi-cloud scenarios," *Journal of Grid Computing*, vol. 16, no. 1, pp. 39–53, 2018.

[39] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1466–1482.

[40] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on gpgpus," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 354–366.

[41] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: a state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2017.

[42] H. Raj, R. Nathuji, A. Singh, and P. England, "Resource management for isolation enhanced cloud services," in *Proceedings of the ACM Workshop on Cloud Computing Security (CCSW)*, 2009, pp. 77–84.

[43] 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 *Proceedings of the ACM conference on Computer and communications security*, 2009, pp. 199–212.

[44] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 351–359.

[45] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2013, pp. 351–364.

[46] N. Subramanian and A. Jeyaraj, "Recent security challenges in cloud computing," *Computers & Electrical Engineering*, vol. 71, pp. 28–42, 2018.

[47] D. Sullivan, O. Arias, T. Meade, and Y. Jin, "Microarchitectural minefields: 4k-aliasing covert channel and multi-tenant detection in iaas clouds." in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[48] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: exposing the perils of security-oblivious energy management," in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2017, pp. 1057–1074.

[49] P. Turner, "Retpoline: a software construct for preventing branch-target-injection," 2018.

[50] 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 *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 281–292.

[51] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. Swift, "A placement vulnerability study in multi-tenant public clouds," in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2015, pp. 913–928.

[52] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the Annual Symposium on Cloud Computing (SoCC)*, 2013, pp. 1–16.

[53] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016, pp. 363–378.

[54] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2015, pp. 1–17.

[55] W. Xiong and J. Szefer, "Survey of transient execution attacks," *arXiv preprint arXiv:2005.13435*, 2020.

[56] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. Fletcher, and J. Torrellas, "Invisispec: Making speculative execution invisible in the cache hierarchy," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 428–441.

[57] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," http://cs.adelaide.edu.au/yval/Mastik, 2016, [Online; accessed 1 May 2021].

[58] Y. Yarom and K. Falkner, "Flush+ reload: A high resolution, low noise, l3 cache side-channel attack," in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2014, pp. 719–732.

[59] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. Springer, 2003, pp. 44–60.

[60] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, "Homealone: Co-residency detection in the cloud via side-channel analysis," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 2011, pp. 313–328.

[61] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant side-channel attacks in paas clouds," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014, pp. 990–1003.

[62] Y. Zhang, M. Li, K. Bai, M. Yu, and W. Zang, "Incentive compatible moving target defense against vm-colocation attacks in clouds," in *Proceedings of the IFIP International Information Security Conference (SEC)*. Springer, 2012, pp. 388–399.