HeteroScore: Evaluating and Mitigating Cloud Security Threats Brought by Heterogeneity

Chongzhou Fang*, Najmeh Nazari*, Behnam Omidi†, Han Wang‡, Aditya Puri§, Manish Arora¶, Setareh Rafatirad*, Houman Homayoun*, and Khaled N. Khasawneh†
* University of California, Davis, {czfang,nnazari,srafatirad,hhomayoun}@ucdavis.edu
† George Mason University, bomidi,kkhasawn@gmu.edu
‡ Temple University, han.wang.hw@temple.edu
§ Foothill High School, ap1016@pleasantonusd.net
¶ LearnDesk, Inc., manish@learndesk.us

***Abstract*—Cloud computing has emerged as a critical part of commercial computing infrastructure due to its computing power, data storage capabilities, scalability, software/API integration, and convenient billing features. At the early stage of cloud computing, the majority of clouds are homogeneous, i.e., most machines are identical. It has been proven that heterogeneity in the cloud, where a variety of machine configurations exist, provides higher performance and power efficiency for applications. This is because heterogeneity enables applications to run in more suitable hardware/software environments. In recent years, the adoption of heterogeneous cloud has increased with the integration of a variety of hardware into cloud systems to serve the requirements of increasingly diversified user applications.**

At the same time, the emergence of security threats, such as micro-architectural attacks, is becoming a more critical problem for cloud users and providers. It has been demonstrated (e.g., Repttack and Cloak & Co-locate) that the prerequisite of micro-architectural attacks, the co-location of attack and victim instances, is easier to achieve in the heterogeneous cloud. This also means that the ease of attack is not just related to the heterogeneity of the cloud but increases with the degree of heterogeneity. However, there is a lack of numerical metrics to define, quantify or compare the heterogeneity of one cloud environment with another. In this paper, we propose a novel metric called Heterogeneity Score (HeteroScore), which quantitatively evaluates the heterogeneity of a cluster. We demonstrate that HeteroScore is closely connected to security against co-location attacks. Furthermore, we propose mitigation techniques to trade-off heterogeneity offered with security. We believe this is the first quantitative study that evaluates cloud heterogeneity and links heterogeneity to infrastructure security.

I. Introduction

The last decade has seen a tremendous rise in cloud computing deployment, usage, and interest from both industry and academia. Powered by the availability of nearly unlimited compute, massive storage capacity, software and, API integration, scalability, and attractive billing models, there are an increasing number of workloads being offloaded to the cloud. Public cloud providers like Amazon AWS [1], Microsoft Azure [5] and Google Cloud [3] provide infrastructure as a service (IaaS) where resources are provisioned and made available in a pay-as-you-go manner. These services have gained tremendous popularity as they increase the speed of deployment and eliminate the need for users to purchase and configure their own infrastructure, thus reducing upfront capital expenditure and increasing the speed of deployments.

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24996
www.ndss-symposium.org

As more compute applications migrate to the cloud, there is a rise in the diversity of user applications running within the cloud infrastructure, leading to an increase in the diversity of hardware resources to satisfy application needs [17]. Instead of having similar machines in the cloud infrastructure, public clouds now offer customized hardware/software environments to meet user requirements, boost performance, and reduce costs. This heterogeneity in hardware and execution environments is also exposed to the user where they can make specific demands on the allocated hardware. For example, Google Cloud [3] allows users to specify a variety of parameters related to the hardware/execution environment when submitting their instances, including CPU type, number of cores, memory, availability of local SSDs and GPU resources, etc. This has also led to the development of various heterogeneity-aware scheduling algorithms to map application demands to hardware [18], [19].

At the same time, the emergence of security threats is becoming a critical problem for cloud users and providers. Specifically, malicious users can deploy micro-architectural attacks since applications from different users share resources when assigned to the same node. Micro-architectural attacks are a class of attacks that exploit hardware vulnerabilities in shared resources, e.g., side-channel attacks [52], [53], [42], rowhammer attacks [47], and transient execution attacks [34], [37], [14], [34]. Previous research has shown the potential for these attacks in a cloud environment [28].

The threat of micro-architectural attacks becomes more significant in heterogeneous cloud environments. The prerequisite of successful micro-architectural attacks is the co-location of attack instances with the targeted victim instance. It is easier to satisfy this prerequisite in heterogeneous clouds compared to homogeneous clouds as each node type may have fewer machines in a heterogeneous cloud environment. Considering heterogeneity during the scheduling processes narrows down the search space of nodes that application instances can be possibly assigned to and renders scheduling decisions more predictable. This makes it easier for attackers to force the scheduler to co-locate their attack applications with victim applications [22].

Previous works [22], [39] have shown the susceptibility of profiling-based schedulers [9], [18], [19] and policy-based schedulers [4], [7] to such attacks. For profiling-based schedulers where the scheduler samples the execution of target program and go through a learning process to achieve a near-optimal placement arrangement [19], by simply mimicking victim application traces [39], attackers can force schedulers to generate the same node-assignment decision, achieve co-location with victim applications and issue micro-architectural attacks. Similarly, for policy-based schedulers where heterogeneity-aware scheduling decisions are made based on user submitted information, replicating scheduling constraints [22] suffices to greatly increase the success rate an attacker might achieve.

Unfortunately, with system integration, continuous development of new architectural features, and increased costs of testing hardware designs, new micro-architectural attacks continue to be discovered. Since these attacks target design flaws of hardware, protecting against them is a difficult process requiring expensive and time-consuming hardware changes. In addition, adding security solutions for each micro-architecture attack may decrease the performance of running workloads in the cloud [38]. Therefore, from a cost, performance, and security perspective, there is a need to defend against micro-architectural attacks using a variety of techniques. One easy way to do so is to simply reduce the chances of co-location.

Since co-location of attack with target applications is the prerequisite step of all micro-architectural attacks, reducing the probability of co-location improves security. This is orthogonal to other defenses and they can be deployed immediately as hardware changes or patches are developed for each new hardware vulnerability. Lastly, reducing the probability of co-location can defend against unknown future micro-architectural attacks that are yet to be discovered.

The mitigation strategy of reducing co-location is crucial to heterogeneous clouds, where co-location is easier to achieve. As heterogeneous cloud deployments continue to rise, it is important to understand how heterogeneity impacts co-location and the infrastructure's security against micro-architectural attacks. However, there is a lack of numerical metrics to define, quantify or compare the heterogeneity of cloud environments. In this paper, we propose a novel metric called Heterogeneity Score (**HeteroScore**). HeteroScore provides a numerical measure of the heterogeneity of cloud infrastructure and helps cloud managers guide the deployment of mitigation strategies. We derive mitigation techniques based on HeteroScore to help prevent co-location forced by attackers. We evaluate our strategies in a real cloud to show the co-location success rate an attacker can achieve and do not make assumptions regarding the underlying virtualization methods. Moreover, we focus on scheduler-level attacks [22], [39].

The following are the contributions of this paper:

1) We propose a novel metric called HeteroScore to quantitatively measure the heterogeneity of cloud infrastructure. Detailed algorithms to calculate HeteroScore are defined and presented.
2) Inspired by HeteroScore, we propose mitigation techniques that trade-off heterogeneity and security, a trade-off not considered earlier.
3) We evaluate HeteroScore, and the proposed mitigation techniques on a real cluster to establish the relationship between HeteroScore and security and show how these mitigation techniques enhance security. Empirically, under the settings in this paper, by keeping a HeteroScore to below 0.9, the co-location rate can be reduced to a safe level.

The remainder of the paper is organized as follows. We present related background knowledge and assumptions of the study in Section II and Section III, respectively. We propose HeteroScore in Section IV and provide related mitigation techniques in Section V. Section VI evaluates the HeteroScore metric as well as the proposed mitigation technologies. We provide additional discussion about our theory and findings in Section VII. Finally, we review related literature in Section VIII and conclude in Section IX.

II. Background

A. *Heterogeneous Cloud Infrastructure*

Recently, heterogeneous clouds have gained adoption in cloud computing environments. This is because heterogeneity enables high performance as users can run their workloads on suitable hardware/software configurations. Also, homogenous cloud environments can become heterogeneous with the deployment of new hardware within existing environments [17], [13], [56]. Compared to homogeneous clouds, where all machines in the clusters are the same, heterogeneous clouds are more diverse and flexible in terms of available hardware and software environment configurations. Therefore, running the same application with different configurations on a heterogeneous cloud can result in varying performances and costs [51].

The fundamental function of a cloud scheduler is to orchestrate the cloud system and make appropriate instance placement decisions, i.e., assign user instances to suitable machines [46]. There have been designs of schedulers and resource provisioning systems that consider heterogeneity [51], [9], [50], [19], [18] and enable service providers to offer suitable resource types based on user application characteristics. Users can also specify requirements and preferences of nodes to run their applications by specifying "Affinity", a widely existing feature in the cloud and cluster schedulers [23], [40], [4], [7], [6]. In our work, we will consider affinity scheduling algorithms and show how different levels of heterogeneity can affect the scheduling results. However, our work is generally applicable to all schedulers and resource provisioning systems that consider heterogeneity.

B. *Co-location Attacks*

Within the cloud, applications from different users can be running simultaneously on the same node. To enhance security and avoid interference between different user applications, specific isolation techniques, e.g., virtual machine (VM) isolation [7] and container isolation [4] are utilized. Applications from the same user are typically packaged in a VM or a container and deployed on a node assigned by cloud scheduler.

However, even with VM or container isolation, there are threats that exist for cloud applications. Prior research works [45], [57] have shown that cross-VM side-channel attacks can be used to extract information from a target VM on the same machine. Emerging micro-architectural attacks impose severe security risks for cloud users. Micro-architectural side-channel attacks, such as FLUSH+RELOAD [53] use the structure of shared caches to allow the initiation of attacks across VMs. In recent years, there have been various forms of other micro-architectural attacks targeting hardware design flaws. Rowhammer attacks [33], [47] utilize circuit features in DRAM chips, like electromagnetic coupling effects, to issue attacks and stealthily cause bit-flips in DRAMs. Transient execution attacks [37], [34] manage to execute instructions that should not be executed by exploiting out-of-order execution patterns or branch predictors in micro-architecture. Fault attacks [41], [48] exploit frequency/voltage adjustment features in modern computer systems and maliciously insert faults during the execution of a program. As new hardware micro-architectural vulnerabilities continue to be discovered, there is a need for solutions to protect against known as well and yet-to-be-discovered.

An important prerequisite to initiating a micro-architectural attack is achieving co-location, i.e., managing to run on the same nodes as the victims. It has been shown that cloud schedulers can be exploited by attackers to achieve co-location [39], [22]. Both policy-based schedulers [22] and machine learning-based schedulers [39] can be exploited to achieve relatively high co-location rates in the heterogeneous cloud. These attack methods exploit the fact that schedulers in heterogeneous clouds tend to place application instances on the most suitable machines. Furthermore, because heterogeneity is considered during the scheduling process, there is a higher chance that schedulers can be tricked and place attack instances on the same node as the victim.

III. THREAT MODEL

In this work, we quantitatively evaluate the security level of a heterogeneous cloud against co-location attacks. The threat model is similar to [22]. In this scenario, the attackers' goal is to achieve co-location with victim applications to issue micro-architectural attacks.

We consider a neutral cloud setting, where service providers are non-malicious and trusted. We do not consider the situation in which service providers cooperate with malicious users to attack other users. After all, offering help to malicious users is against the profit goal of cloud or cluster service providers. Victim applications, malicious applications, and other unrelated applications on the cluster are considered equivalent by the cluster service provider and the scheduling algorithm. All scheduling decisions are determined by cluster resource states and user requests. We assume that cloud schedulers do not apply mitigation strategies against co-location attacks [58] in the scheduling step as the idea of forcing co-location is relatively novel [22], [39] and possible mitigation strategies are not yet integrated into schedulers to the best of our knowledge.

Regarding users of the system, we assume attackers do not have any privileges over other users. Permissions available to attackers are limited to: (1) requesting computing resources in the cluster by submitting configuration scripts, and (2) running their programs on assigned nodes and resources. Attackers do not have access to resources that other users cannot access, nor do they have access to special features of the service infrastructure software system that other users cannot access. In this study, all users can only access and operate on nodes assigned to their applications and corresponding assigned resources. Since our threat model does not assume any privileges of the attackers on the system, the analysis has wider applicability than a model that assumes privileges. We assume without argument, justification, or experiment that the attacker knows the execution specifics required to target the victim. This is a reasonable assumption since these specifications are usually provided to optimize the placement, for example, forcing the scheduler to schedule an instance to a node with higher I/O capacity or a specific platform. We can safely assume that attackers know the execution features of victims and victims always want to optimize performance by selecting more suitable execution environments; hence the assumption that attackers can infer the victim's specifications is reasonable.

Conforming to the general settings of cloud and cluster service infrastructures [4], [6], [7], [54], [1], scheduling decisions are made based on the user-submitted metadata. This metadata contains resource requirements, e.g., number of CPU cores required, amount of memory needed, etc. This metadata also contains requirements and preferences that fine-tunes scheduling processes e.g. requirements and preferences about whether or not to schedule on a node with certain features. Previous work [22] shows that by exploiting this scheduler feature and replicating the second type of information, it becomes easier for an attacker to co-locate with victims on a heterogeneous cluster. In this study, we assume attackers use a similar method to achieve co-location.

This study targets the co-location attack vulnerability of heterogeneous clouds and is not a study of micro-architectural hardware vulnerabilities. How specific types of micro-architectural attacks are deployed and related low-level mitigation techniques are beyond the scope of this paper. We first evaluate the relationship between heterogeneity and co-location attack vulnerability and then add and evaluate mitigation techniques later in this paper.

IV. HETEROSCORE

In this section, we introduce a metric called HeteroScore to quantitatively measure heterogeneity of a cloud computing cluster. We first provide the intuition behind this metric, and then we define related variables and details of the algorithm to calculate HeteroScore. A mitigation technique based on HeteroScore is provided in Section V.

A. Definition and Explanation of HeteroScore

To quantitatively measure heterogeneity, we will (1) define a formal representation of the target cloud computing cluster, and (2) define a mathematical metric and a calculation procedure of the metric based on the representation.

1) Cluster Representation: In this part, we introduce how a single node of a cloud computing cluster can be represented. Nodes in a cluster have metadata information in the format of "label-value" pairs that record specific features of nodes. This is consistent with cloud computing clusters that utilize management frameworks such as OpenStack [7] and Kubernetes [4]. In this approach, "label" refers to a feature of the node and "value" refers to a specific value that corresponds to that feature. An example "label-value" description can be "`CPUType - Intel Xeon`", which indicates the model of the CPU. The label-value pairs can be used to specify various resources like memory, GPU, network bandwidth, disk space, etc. There can be multiple labels attached to a node and usually depict physical hardware properties. The "label-value" description information is used by the cluster schedulers to assign suitable machines to the user.

In our model, each node consists of multiple "label-value" description items. We consider each label as one dimension. By assigning each value in the "label-value" pairs a numerical integer value, we craft a vector for node $\mathrm{N}^{(i)}$ in a d-dimensional space:

$$\mathrm{N}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_k^{(i)}, ..., x_d^{(i)})^{\mathrm{T}}, \tag{1}$$

where d is the total number of labels that have appeared in the cluster and $x_k^{(i)}$ corresponds to the assigned label value in the k-th dimension.

By acquiring and gathering the representation for each node in the cluster, the cluster can be represented as a set of N points $\mathrm{N}^{(i)}$ $(1 \leq i \leq n)$ on a d-dimensional space:

$$\mathcal{C} = \{\mathrm{N}^{(1)}, \mathrm{N}^{(2)}, ..., \mathrm{N}^{(i)}, ..., \mathrm{N}^{(n)}\}, \tag{2}$$

where n is the number of nodes in the cluster.

After mapping the cluster into a geometric space, we are able to process cluster information according to the geometric attributes of the representation. Our further processing will be based on this representation. It is worth noting that the actual meanings of certain labels/values are ignored in our calculation process, i.e., we focus on a higher representation level and ignore lower-level details in real-world configurations after finishing the mapping.

2) Proposed Metric: Our goal is to use a quantitative metric to depict the heterogeneity of a cluster. The "label-value" pairs typically depict physical attributes of the node. It is intuitive from the representation of the nodes that in the d-dimensional space, longer distances between points correspond to larger hardware differences between nodes. In a relatively homogeneous cluster, nodes will be densely clustered in the d-dimensional space as each node has similar hardware configurations, whereas in a relatively heterogeneous cluster, nodes tend to sparsely scatter in the space.

To depict how sparse $\mathcal{C}$ is in the d-dimensional space, we define the HeteroScore metric denoted $\mathcal{H}_c$:

$$\mathcal{H}_c = 1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{I}\{\rho(\mathrm{N}^{(i)}, \mathrm{N}^{(j)}) \leq t_h\}}{n^2}. \tag{3}$$

Here $\rho(\mathrm{N}^{(i)}, \mathrm{N}^{(j)})$ refers to the Euclidean distance between node $\mathrm{N}^{(i)}$ and node $\mathrm{N}^{(j)}$ in the d-dimensional space:

$$\rho(\mathrm{N}^{(i)}, \mathrm{N}^{(j)}) = \sqrt{\sum_{k=1}^{d}(x_k^{(i)} - x_k^{(j)})^2}, \tag{4}$$

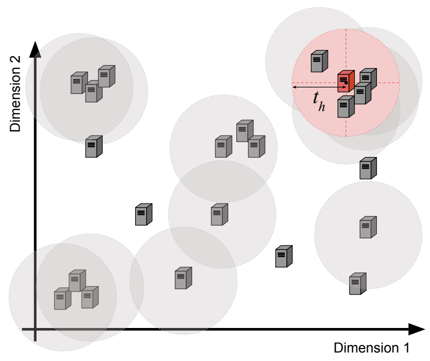


Fig. 1: Cluster mapped to a d-dimensional space ($d = 2$) and demonstration of $\mathcal{H}_c$ calculations. Nodes are assigned coordinates in this space, which combine to represent cluster information. To compute $\mathcal{H}_c$, for every node $\mathrm{N}^{(i)}$ we count how many other nodes lie within distance t_h and perform the mentioned operations.

and t_h is a predefined threshold value. We define $\mathbf{I}\{*\}$ in Eq. (3) as an indicator:

$$\mathbf{I}\{*\} = \begin{cases} 1, & \text{Given condition } * \text{ is satisfied,} \\ 0, & \text{Otherwise.} \end{cases} \tag{5}$$

The calculation process in Eq. (3) works as a two-step process. First, it iterates through every node $\mathrm{N}^{(i)}$ $(1 \leq i \leq n)$ and counts how many nodes (including itself) lie within a predetermined distance t_h (calculated by $\sum_{j=1}^{n} \mathbf{I}\{\rho(\mathrm{N}^{(i)}, \mathrm{N}^{(j)}) \leq t_h\}$) of the current node. Second, all counts are aggregated and normalized to a value between 0 and 1. The normalized value is subtracted from 1 to ensure that the higher heterogeneity of a cluster produces a higher HeteroScore value. A similarity threshold is applied to quantize the vector differences to be either 1 or 0. The calculation process is depicted in Figure 1 for a 2-dimensional space.

There are a few notable features of HeteroScore ($\mathcal{H}_c$).

Range. Due to the definition of $\mathbf{I}\{\}$ and the process of summation, it is obvious that

$$0 < \sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{I}\{\rho(\mathrm{N}^{(i)}, \mathrm{N}^{(j)}) \leq t_h\} \leq \sum_{i=1}^{n}\sum_{j=1}^{n} 1 = n^2. \tag{6}$$

Hence:

$$0 \leq \mathcal{H}_c < 1. \tag{7}$$

Generalizability. With the normalization step in Eq. (3), we are able to derive the proportion of nodes similar to each other of the exact number of nodes. Also, in the calculation process, we do not consider specific hardware label-value pairs or features. Therefore, the values of $\mathcal{H}_c$ can be compared across clusters of different sizes and settings.

Determining factor. Since during the computation process, we only consider cluster settings and ignore user-side information and run-time states, the **only** factor that influences $\mathcal{H}_c$ is how cluster managers configure and expose information about their clusters to users. $\mathcal{H}_c$ is hence a suitable metric to quantitatively measure the heterogeneity of a cluster as exposed to an application by the cluster manager.

B. Algorithm

Although the calculation in Eq. (3) is relatively straightforward, there are more practical considerations in its algorithmic implementation in the real world. This subsection provides details as to how HeteroScore can be calculated for a real-world cloud computing cluster.

Please note that in this subsection, the use of notation $\mathrm{N}^{(i)}$ refers to the assigned coordinate vector of the corresponding node in the d-dimensional space, as shown in Eq. (1).

1) Assigning Coordinates to Nodes: In a real-world cluster setting, certain information in the metadata may be missing, i.e., the values of some labels may be missing. Our previous discussion assumes that all label-value pairs are always available. However, that is not always the case and this scenario needs to be carefully handled to generate suitable representations that are consistent with theory. A portion of our algorithm will be dedicated to policies that properly define and assign missing values to these nodes.

In our implementation, we:

1) gather and number labels that appeared in the metadata of the cluster nodes, define the values of all labels that have appeared, and generate axes for the d-dimensional space that represents the cluster;
2) generate corresponding coordinates according to the description of each node.

The first step is done by iterating through the metadata of all nodes and recording description labels and corresponding values that have appeared. Once this is completed, the dimension d is determined. The mapping between label values and numbers is then defined. In this paper, we only assign integer numbers to each value. We assume the corresponding value-number mapping of the cluster is stored in $\mathcal{V}$ and can be accessed by $\mathcal{V}(k, v)$, where k is the assigned dimension number of a label, and v is the label value, i.e., $\mathcal{V}(k, v)$ returns a number corresponding to the value for the k-th label. In the process of constructing this mapping we let $\mathcal{V}(k, v) > 0$ if k is a valid label.

In the second step, we process coordinates according to node metadata information. In the first step we parse every node's metadata and stored related "label-value" pairs in $\mathcal{K}^{(i)}$ $(1 \le i \le n)$ as:

$$\mathcal{K}^{(i)} = \{v_1^{(i)}, v_2^{(i)}, ..., v_k^{(i)}, ..., v_d^{(i)}\}. \tag{8}$$

Here, v_k is the value matching the k-th label. v_k can be assigned a special value ϕ if the value of the label is missing in the description of a node. We define $\mathcal{V}(k, \phi) = 0$. Coordinates $\mathrm{N}^{(i)}$ will be generated based on $\mathcal{K}^{(i)}$.

There are two cases:

- Fully defined nodes: all d labels have a label;
- Partially defined nodes: values of 1 or more label fields are missing.

For fully defined nodes, a simple assignment policy is enough to generate coordinates based on $\mathcal{K}^{(i)}$. We call this policy as Trivial Assignment. For partially defined nodes, we consider relations with other nodes to assign coordinates. This policy is called as Relation-based Assignment.

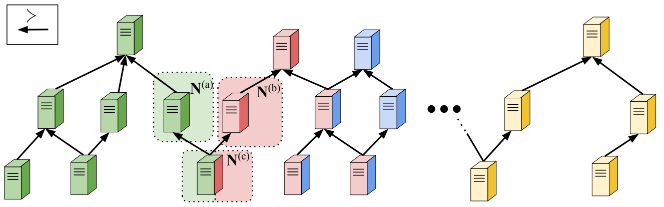


Fig. 2: The diagram of constructed data structure. The representation of a cluster can be parsed to multiple tree-like data structures in the diagram. In this figure, $\mathrm{N}^{(a)} \succ \mathrm{N}^{(c)}$, $\mathrm{N}^{(b)} \succ \mathrm{N}^{(c)}$, and $\mathrm{N}^{(c)}$ is a leaf node.

Trivial assignment. We apply a mechanical assignment policy shown in Algorithm 1 to all nodes. After executing

Algorithm 1 Trivial assignment of $\mathrm{N}^{(i)}$.

```
Require: $\mathcal{C}$, $\mathcal{K}^{(i)}$ (for $1 \le i \le n$), $\mathcal{V}$.
  function TrivialAssignment($\mathcal{K}^{(i)}$, $\mathcal{V}$)
      for $k$ in $1..d$ do
          $x_k^{(i)} = \mathcal{V}(k, v_k)$
      end for
      return $(x_1^{(i)}, x_2^{(i)}, ..., x_k^{(i)}, ..., x_d^{(i)})^{\mathrm{T}}$
  end function
  for $\mathrm{N}^{(i)}$ in $\mathcal{C}$ do
      $\mathrm{N}^{(i)}$ = TrivialAssignment($\mathcal{K}^{(i)}$, $\mathcal{V}$)
  end for
```

TrivialAssignment for every node, (1) coordinates of fully defined nodes will be non-zero values and the assignment process is finalized; (2) partially defined nodes will contain 0s in their coordinates.

Relation-based assignment. To generate coordinates that do not contain 0s, we will apply a predefined assignment policy. Firstly, we define relations between nodes notated: $=$, $\neq$, $\succ$ and $\prec$:

For indexes i, j $(1 \le i, j \le n)$, let $\delta = \{k \mid x_k^{(i)} \neq x_k^{(j)}\}$.

Definition 1: If δ is empty, we say $\mathrm{N}^{(i)} = \mathrm{N}^{(j)}$.

Definition 2: If $\forall k$ we have $x_k^{(i)} = 0$ and $x_k^{(j)} \neq 0$, we say $\mathrm{N}^{(i)} \succ \mathrm{N}^{(j)}$ or $\mathrm{N}^{(j)} \prec \mathrm{N}^{(i)}$.

Definition 3: If none of the following relations hold: $\mathrm{N}^{(i)} = \mathrm{N}^{(j)}$, $\mathrm{N}^{(i)} \succ \mathrm{N}^{(j)}$ or $\mathrm{N}^{(i)} \prec \mathrm{N}^{(j)}$, we say $\mathrm{N}^{(i)} \neq \mathrm{N}^{(j)}$

In our definition of this type of relations, if $\mathrm{N}^{(i)} \succ \mathrm{N}^{(j)}$, node $\mathrm{N}^{(j)}$ is described with more details and hence contains more label-value pairs in the description metadata. With this definition, we are able to construct multiple tree-like data structures from the coordinates obtained from step 1, as shown in Figure 2. We define some features of such data structure:

1) If $\mathrm{N}^{(c)}$ is the child of $\mathrm{N}^{(p)}$, then $\mathrm{N}^{(c)} \prec \mathrm{N}^{(p)}$;
2) Overlapping of trees is allowed, and a node can appear in multiple different trees.

We can prove the following Lemma:

Lemma 1: Fully defined nodes can only be leaf nodes.

Proof: Assume there is a fully defined node $\mathrm{N}^{(a)}$ which is not a leaf node, i.e., $\exists \mathrm{N}^{(b)}$ s.t. $\mathrm{N}^{(a)} \succ \mathrm{N}^{(b)}$. In this case, according to Definition 2 $\exists k$ s.t. $x_k^{(a)} = 0$, which contradicts our assumption that $\mathrm{N}^{(a)}$ is fully defined. ■

Once constructed such data structures, for every partially defined node $\mathrm{N}^{(i)}$ we can recursively search for leaf nodes in the same tree and store them in a set $\mathcal{L}^{(i)}$. We define the following policy:

Policy 1: We do the following to assign coordinates to $\mathrm{N}^{(i)}$:

1) For all partially defined nodes $\mathrm{N}^{(j)}$ in $\mathcal{L}^{(i)}$, for every index k s.t. $x_k^{(j)} = 0$, let $x_k^{(j)} = \dfrac{\sum_{l=1}^{n} x_k^{(l)}}{\sum_{l=1}^{n} \mathbf{I}\{x_k^{(l)} \neq 0\}}$.
2) For every index k s.t. $x_k^{(i)} = 0$, let $x_k^{(i)} = \dfrac{\sum_{\mathrm{N}^{(j)} \in \mathcal{L}^{(i)}} x_k^{(j)}}{|\mathcal{L}^{(i)}|}$.

Policy 1 does two things. (1) It first processes coordinates of partially defined leaf nodes in $\mathcal{L}^{(i)}$. For every dimension, it iterates through the coordinates of all nodes in $\mathcal{C}$ and calculates the average of non-zero entries for this dimension. Then it uses these values to update zero entries in the coordinates of leaf nodes. (2) Secondly, it uses the processed leaf node coordinates to update $\mathrm{N}^{(i)}$. It calculates average coordinates in $\mathcal{L}^{(i)}$ and uses the results to update zero entries in $\mathrm{N}^{(i)}$. The reason for using this policy is that during the scheduling process if a user does not provide a specification for a certain feature domain, all nodes that satisfy other requirements, either with/without a description in that feature domain, will be considered candidates. Policy 1 places all partially defined nodes in balanced positions to consider this scheduling effect.

The algorithm to construct such data structures and assign coordinates is shown in Algorithm 2. Here, we first use a matrix $\mathbf{R}$ that is similar to an adjacency matrix to record all relations between nodes and hence finish the construction of the proposed data structure. Then we apply Policy 1 and finish assigning coordinates to every node in $\mathcal{C}$.

Algorithm 2 Constructing relation data structures and assigning coordinates.

```
Require: $\mathcal{C}$ after executing TRIVIALASSIGNMENT on every node.
  $\mathbf{R} = \{r_{ij}\}_{n\times n} = \{\neq\}_{n\times n}$
  for $i$ in $1..n$ do
      for $j$ in $i..n$ do
          $r_{ij}, r_{ji}$ = Relation between $\mathrm{N}^{(i)}$ and $\mathrm{N}^{(j)}$ $(=,\neq,\succ,\prec)$
      end for
  end for
  for all partially defined $\mathrm{N}^{(i)}$ do
      Perform depth-first search for leaves in $\mathbf{R}$ and store them in $\mathcal{L}^{(i)}$
      Apply Policy 1 to $\mathrm{N}^{(i)}$
  end for
  return $\mathcal{C}$
```

After this step, the representations of all nodes will be completed. For any $\mathrm{N}^{(i)}$ $(1 \leq i \leq n)$ there is no 0 component.

2) Calculating HeteroScore: Algorithm 3 calculates $\mathcal{H}_c$ based on the acquired coordinates and the computation shown in Eq. (3).

Algorithm 3 Calculation of $\mathcal{H}_c$.

```
Require: Cluster representation $\mathcal{C}$ with coordinates fully assigned, number
    of nodes in cluster $n$.
  $s = 0$
  for $i$ in $1..n$ do
      for $j$ in $1..n$ do
          $s = s + \mathbf{I}\{\rho(\mathrm{N}^{(i)}, \mathrm{N}^{(j)}) \leq t_h\}$
      end for
  end for
  return $\mathcal{H}_c = 1 - \dfrac{s}{n^2}$
```

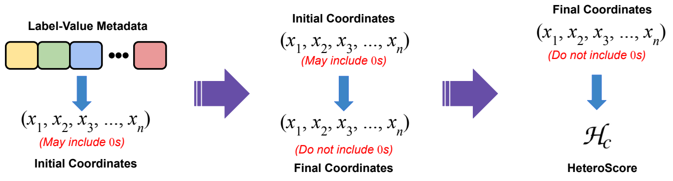


Fig. 3: 3 stages of $\mathcal{H}_c$ calculation.

After executing Algorithm 3, HeteroScore $\mathcal{H}_c$ of a cluster viewed from users is determined.

In summary, Algorithm 1 initializes all coordinates for every node. Algorithm 2 processes the case that label-value pairs are missing in some nodes and assigns proper coordinates according to a set of predefined rules. Finally, with Algorithm 3 the computation in Eq. (3) is performed and $\mathcal{H}_c$ is determined. The summary diagram is shown in Figure 3.

3) Elaboration: In this paper, the metric HeteroScore $\mathcal{H}_c$ is an indicator of the heterogeneity of a cluster viewed from a user. Cluster managers can make design choices based on HeteroScore and control the level of heterogeneity information exposed to users and take measures to defend against co-location attacks.

Selection of t_h in the calculation process of HeteroScore $\mathcal{H}_c$. The value of t_h can be considered the threshold of similarity. A smaller t_h value indicates more strict requirements for nodes to be considered similar during scheduling, i.e., the geometric representation of two nodes in the representation space should be closer for them to be considered similar. Choices of t_h affect the calculation results of HeteroScore $\mathcal{H}_c$; hence designers should make the trade-off in the selection of t_h.

Dealing with Real-World Label-Value Pairs. In our calculation process, we assign an integer value to each value in the label-value pairs and generate coordinates accordingly. However, in practice, since these values are usually categorical and are not continuous, it may not be suitable to process node information using this method. We suggest cluster managers use one-hot encoding to split one such dimension into several dimensions and continue using the proposed algorithm to calculate HeteroScore.

V. Mitigating Co-Location Attacks

Based on the computation results of HeteroScore $\mathcal{H}_c$, we are able to derive mitigation strategies toward co-location steps at the scheduler level.

A mitigation strategy is discussed in [22]. In the authors' strategy, during the scheduling process, some nodes are listed as candidates without fully examining whether their features can match users' specifications. Therefore, the search space is enlarged, resulting in increased difficulty in achieving co-location. To relate this approach to HeteroScore, we consider the calculating process of $\mathcal{H}_c$. This approach is equivalent to randomly selecting some nodes and manually setting the representation of these nodes to the same position in the mapped space that satisfies users' requirements during the scheduling process. With this step, part of the cluster's heterogeneous nodes are replaced by a set of homogeneous nodes; hence HeteroScore $\mathcal{H}_c$ drops. Therefore, the discussed strategy in [22] can be seen as a coarse-grained approach (node-level) to reduce HeteroScore.

Inspired by the calculation process of HeteroScore, we can have a more fine-grained mitigation strategy to reduce the cost of mismatching users' requirements. We propose a strategy called **Hiding-Label-defense (HLD)**. We assume that cluster managers can dynamically change the parameters of the scheduler. While maintaining a cluster, cluster managers can randomly select a set of labels to hide from users for a period of time, i.e., configure the scheduler to ignore information related to these labels during scheduling. This label-hiding process will be integrated to the scheduler by default; however, performance-sensitive users still have the chance to specify hard requirements to run on servers with certain features. Users will not have knowledge regarding which labels are hidden and non-performance-sensitive users will operate as if the defense is not deployed. By doing this, from the users' view, the cluster is described in fewer details; hence the search space for the attacker to achieve co-location will be larger. To relate this defense strategy with HeteroScore, we observe that after applying this strategy, several of the d dimensions are reduced, and the calculation of $\mathcal{H}_c$ is performed in a lower-dimensional space. Qualitatively viewing Eq. (3), reducing dimensions leads to an increased number of pairs (i, j) satisfying $\rho(\mathrm{N}^{(i)}, \mathrm{N}^{(j)}) < t_h$ hence $\mathcal{H}_c$ will decrease. In this way, HLD reduces HeteroScore and hide heterogeneity to users.

To get a more balanced reduction instead of mechanically hiding the same label, we propose another strategy called **Randomly-Hiding-Label-defense (R-HLD)**. R-HLD randomly ignores label-values pairs in the scheduling constraints submitted by users with a specified probability p_{hide}. Compared to HLD, which selects labels and hides the same labels in each node, R-HLD hides different labels in nodes and can achieve a more balanced sacrifice in performance. Same as HLD, users still have the chance to select whether or not and to what extent to involve in the defense by choosing the parameter p_{hide}. To relate R-HLD with the calculation of $\mathcal{H}_c$, we observe that R-HLD is equivalent to randomly selecting and setting coordinate components of $\mathrm{N}^{(i)}$ $(1 \leq i \leq n)$ to 0, which adds more homogeneous points to the point set hence making the d-dimensional space less sparse.

Compared to the coarse-grained approach [22], HLD and R-HLD are more controllable regarding the sacrifice of mismatching users' requirements. Also, the cluster managers can have better control regarding the outcome of the scheduling algorithm. Though the cost seems relatively high, we allow users to flexibly select whether or not to involve in the defense process and choose the level of protection. Also, in practice, R-HLD and HLD do not necessarily need to be applied to all labels – cloud managers can select relatively unimportant features to perform these defense operations.

VI. Evaluation

A. *Calculation Results in Simulated Cluster*

In this part of the evaluation, results are obtained via behavioural simulation. Our goal in this part is conceptually validating our theory with a focus on how HeteroScore is related to cluster heterogeneity as well as providing visualization results of the previously proposed method.

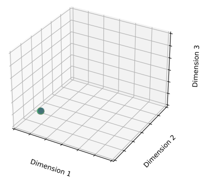
1) Experiment Setup: Our simulator randomly constructs information about a cluster based on user-provided number of total nodes in the cluster, total number of labels in the cluster, etc. In our experiment, we limit the number of nodes to 100. In every experiment, we randomly re-generate a cluster according to rules specified as follows. We repeat a node generation process 100 times to generate label-value description metadata of 100 nodes. Within each node, we randomly generate a list of integers of length n_l to represent the label values of the corresponding n_l labels. We define n_c as the number of choice values in each label-value pair. Every element in the integer list has a value between 0 and n_c (including 0 and n_c), where 0 means the corresponding label value is missing.

2) HeteroScore Results and Visualization: We provide the calculation results of a simulated cluster in this part. To simplify the visualization, we limit the number of labels to 3 (i.e., $n_l = 3$). According to our algorithm, we can derive that $d = 3$ and the whole cluster will be mapped to a 3-dimensional space.

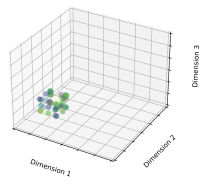
We provide visualization results of our cluster representation in Figure 4. In this experiment, we vary n_c, which is the number of candidate values for each label-value pair and provide the visualization results of the cluster and calculate the corresponding HeteroScore of the generated cluster. The probability that a label-value pair is missing in a node's description metadata is 20%. Figure 4 shows that:

1) As the number of candidate values for each label-value pair n_c increases, the possible number of label-value pairs increases as well, resulting in a more heterogeneous cluster. This change in heterogeneity (diversity) in a cluster is reflected in the sparsity of the corresponding visualization results. As n_c increases, the point set constructed by the representations of nodes becomes more and more sparse in the 3-dimensional space.
2) This change in the sparsity of the point set of node representations is captured and reflected in the calculation results of HeteroScore $\mathcal{H}_c$. As n_c increases, the constructed cluster becomes more and more heterogeneous, and the quantitative metric $\mathcal{H}_c$ hence increases from 0 (homogeneous cluster) to approximately 1 (very heterogeneous cluster).

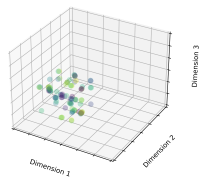
Results shown in Figure 4 prove that our proposed method of calculating HeteroScore is able to capture the sparsity in the geometric representation of the cluster, hence reflecting the heterogeneity of a cluster. Also, as a byproduct, our representation of a cluster can be easily visualized and presented, which can work as a visualization step for cluster designers to better understand the setting of a cluster.



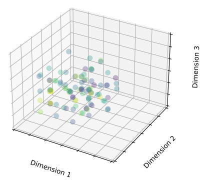
(a) $n_c = 1$, $\mathcal{H}_c = 0$.



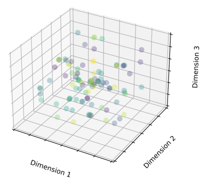
(b) $n_c = 2$, $\mathcal{H}_c = 0.6976$.



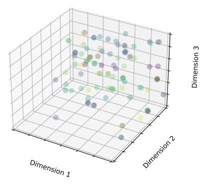
(c) $n_c = 3$, $\mathcal{H}_c = 0.8826$.



(d) $n_c = 4$, $\mathcal{H}_c = 0.9456$.



(e) $n_c = 5$, $\mathcal{H}_c = 0.9678$.



(f) $n_c = 6$, $\mathcal{H}_c = 0.9704$.

Fig. 4: Visualization results of a cluster representations and calculation results of HeteroScore $\mathcal{H}_c$ under different n_c settings. As n_c increases, the point set representation of the cluster becomes more and more sparse, and HeteroScore $\mathcal{H}_c$ increases. To clearly show the representations of a cluster, for each node, we randomly add small perturbations to its coordinates to avoid multiple points with the same coordinates crowding at the same position when plotting these points.

In a real-world scenario, some settings may change: the number of labels in the cluster n_l will change and possibly have a value larger than 3, the number of possible values can vary, etc. However, since our calculation process relies on none of these parameters, our metric can still be applied to measure the heterogeneity of a cluster.

3) Parameter Choices: To evaluate how the design parameters may affect HeteroScore calculation, we change the value of threshold t_h in Eq. (3) and compare the calculation results of one-hot-encoding coordinate assignment and continuous coordinate assignment, as discussed at the end of Section IV. The results are shown in Figure 5.

From Figure 5, we can see that: (1) Different selection of t_h result in different $\mathcal{H}_c$ values. While under $t_h = 1$ both categorical and continuous coordinate assignments show similar results, higher t_h values cause $\mathcal{H}_c$ to be different. Also, under a relatively high value ($t_h = 3$), one-hot-encoding assignment does not perform well in capturing the heterogeneity of the cluster anymore. (2) Though under $t_h = 1$ the two methods have similar results, the counter-intuitive continuous coordinate assignment policy is more robust when t_h changes.

Based on the results, we conclude that continuous value assignment is better: under lower t_h settings, it achieves similar results as categorical value assignment and is more robust under higher t_h values. In this paper, we will select $t_h = 1$ to make $\mathcal{H}_c$ more sensitive to heterogeneity changes and show that even this trivial selection is sufficient to capture heterogeneity in a cluster, though, as shown in Figure 5 higher t_h values like $t_h = 2$ can be better choices in reality.

4) The Effects of Mitigation Strategy on HeteroScore: This part showcases how our mitigation strategies affect HeteroScore. First, we construct random clusters using our previously mentioned simulator, then apply our proposed mitigation strategies to see how HeteroScore $\mathcal{H}_c$ is affected. We follow the settings of the previous experiment, where there are 3 labels for nodes, and each label has 6 possible values. Corresponding HeteroScore results, as well as visualization results, are shown in Figure 6.

To compare the results of R-HLD and HLD, we control the total number of labels hidden to be equal in these two scenarios. In Figure 6, we set the hiding probability p_{hide} of R-HLD to 0.33 and 0.67, respectively to compare with hiding 1 and 2 labels in HLD. In Figure 6b and Figure 6c, Figure 6d and Figure 6e, the numbers of labels hidden in R-HLD and HLD are approximately the same (e.g., the number of labels hidden by (1) applying HLD to hide 1 of the 3 dimensions, as shown in Figure 6c equals to (2) applying R-HLD to randomly hide labels with a probability of 0.33, as shown in Figure 6b, etc.). From the $\mathcal{H}_c$ results in Figure 6, we can see that:

1) With approximately the same number of labels hidden, both R-HLD and HLD achieves the same effect of HeteroScore reduction, i.e., $\mathcal{H}_c$ is reduced to approximately the same level.
2) In HLD, deterministically hiding labels results in dimension reduction of the space of a cluster.
3) Applying R-HLD results in a denser point set in the d-dimensional space.

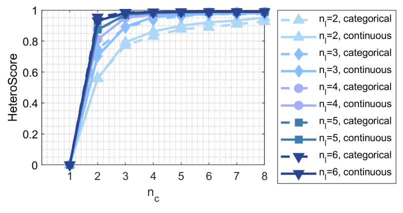
The choice of R-HLD and HLD will be further discussed in Section VII.

B. Relating HeteroScore with Security

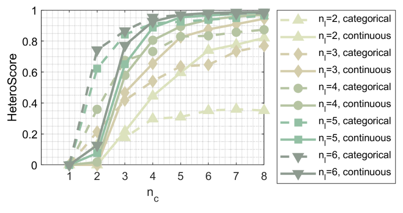
In this part, we present results through experiments in a 40-node Kubernetes cluster. We first show the relation between HeteroScore and co-location rate, and then we present how our HLD and R-HLD mitigation strategies can decrease the co-location rate. Finally, we provide a case study of applying HeteroScore to clusters in a production environment.

1) Experiment Setup: Due to the limited access to large scale hardware, our experiments are deployed on a 40-node Kubernetes cluster on CloudLab [21]. CloudLab is a platform that provides dedicated cluster nodes for researchers to deploy cloud systems. We utilize the default `k8s` profile to deploy a Kubernetes cluster [4]. Our results obtained on Kubernetes is representative since (1) Kubernetes is widely used and the same type of schedulers are widely used in other types of cloud infrastructure [6], [7], (2) our focus is only on the scheduler part; hence the selection of specific cloud framework is not important.

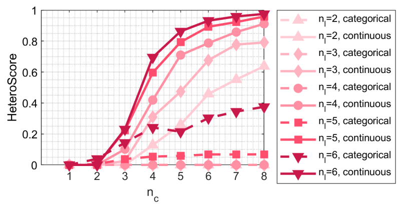
Nodes in our cluster are all configured to have 10 label-value pairs in their corresponding metadata description fields. Because we cannot control the hardware or low-level system environments in CloudLab cluster, we assign these label-value descriptions only for scheduling purposes. These label-value pairs resemble the setting in a real cloud, containing: GPU



(a) $t_h = 1$.

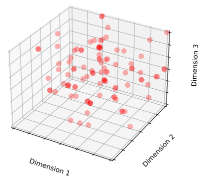


(b) $t_h = 2$.

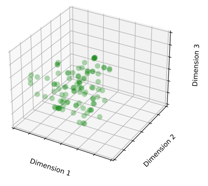


(c) $t_h = 3$.

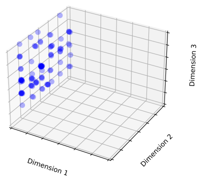
Fig. 5: $\mathcal{H}_c$ scores under different t_h and encoding settings.



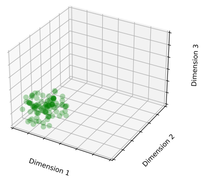
(a) Original, 0.9768.



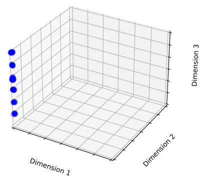
(b) R-HLD, 0.9108.



(c) HLD, 0.915.



(d) R-HLD, 0.6784.



(e) HLD, 0.7118.

Fig. 6: Effects of our proposed mitigation strategies on HeteroScore $\mathcal{H}_c$. In (b) and (c), approximately 33% of labels are hidden either by random selection (R-HLD), or by deterministically hiding certain labels (HLD). In (d) and (e), 67% of labels are hidden.

type, CPU type, memory type (high, medium or low capacity), disk type (local SSD or not), network bandwidth, region, partition, operating system version, FPGA type and security level.

Since our evaluation in this part does not involve performance analysis, symbolically assigning such description information to nodes is acceptable. To better simulate a production environment cluster, numbers of possible values are not fixed for each label. For each label field, there can be 2 to 7 different available values to choose from. These descriptions are randomly generated and assigned to nodes in our cluster using `kubectl label nodes <node name> <label>=<value>` command.

To mimic the execution environment in real world, applications to be deployed on the cluster are selected from the most widely used container applications from Docker Hub [2]. To generate a job on the cluster, we randomly select an application type out of 6 types of container applications and generate its required `.yaml` metadata file that contains basic information as well as scheduling constraints. We generate 1400 applications in each experiment, with: (1) 200 applications as victim instances of 200 separate, independent co-location attacks; (2) 200 applications as attack instances of the 200 corresponding co-location attacks; and (3) 1000 applications are considered as background applications to provide background contention/noise in a real multi-user environment. Here, each attack instance only targets one victim instance.

Regarding scheduling constraints, we assume all users use Node Affinity features [4], [23] to fine-tune scheduling results and force the cloud scheduler to assign a node with specified features. Also, attackers take the approach described in [22] to replicate the scheduling constraints of victims to maximize the probability of achieving co-location.

We set up machines and assign full node descriptions to each node without empty label-value pairs, i.e., for every node, each of the 10 label fields is assigned a value. Each experiment consists of three phases: (1) generating `.yaml` files for the 1400 involved applications; (2) useing `kubectl apply -f <job name>.yaml` command to deploy all applications to the cluster; (3) collecting and analyzing pod (Kubernetes container instance [4]) information to obtain the co-location rate of attackers.

2) Relation between HeteroScore and Co-Location Rate: In this part, we show the relation between HeteroScore and co-location rate. To obtain clusters of different HeteroScore from users' view, we change the generation script of nodes and applications (HLD is involved in the generation process to produce cluster views with different HeteroScores), calculate the resulting HeteroScores, deploy applications, and collect co-location data. The results are shown in Figure 7.

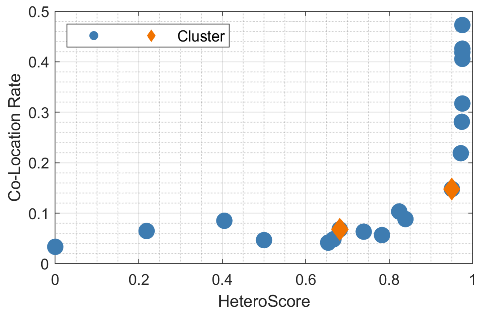


Fig. 7: The relation between HeteroScore and co-location rate. The two orange points show clusters that have the same number of labels hidden but different HeteroScores, and hence co-location rates are different.

In Figure 7, we can see that co-location rate and HeteroScore are correlated. As HeteroScore increases, the co-location success rate of attackers increases as well. The fastest increase in co-location rate occurs at higher HeteroScore values.

The trend in Figure 7 can be explained as follows. When HeterScore $\mathcal{H}_c$ is lower, which indicates a lower heterogeneity in the cluster, there are more schedulable nodes for each submitted application. The search space for schedulers shrinks as HeterScore $\mathcal{H}_c$ increases, which results in more deterministic scheduling results. Attackers can hence achieve co-location with a higher success rate. As HeteroScore $\mathcal{H}_c$ is related to the average number of similar nodes in the cluster, in clusters, with higher heterogeneity (i.e., higher HeteroScore), the change in the value of HeteroScore indicates a larger change in the size of search space hence the co-location rate increases faster.

It is worth noting that in [22] the authors conclude that when affinity features are used more often (users use more labels), co-location can be achieved with a higher success rate. Here, we argue that this observation is not accurate enough but can still be explained by HeteroScore. Usually, when fewer labels are used by users, from the users' view, the cluster is less heterogeneous with a lower HeteroScore. However, this is still related to features of node labels, and there can be counterexamples. In Figure 7, there are two points in the scatter plot that are marked as orange. For these two experiments, the numbers of labels contained in users' scripts are the same (both 7, with 3 labels that are hidden). The difference lies in the selection of labels to hide and the resulting HeteroScore. We can see from Figure 7 that though the numbers of labels users use are the same, co-location rates are different. However, it still matches our conclusion above that higher HeteroScore leads to a higher co-location rate.

In conclusion, our experiment results shown in Figure 7 prove that HeteroScore is correlated with the co-location success rate an attacker can achieve and hence, Empirically, under $t_h = 1$, keeping a HeteroScore under 0.9 can lead to co-location rates that are low enough to be threats for cloud users. HeteroScore can be a quantitative metric to evaluate the security level of a cluster against co-location attacks.

In addition, to further show how HeteroScore is related to co-location rate and the threat level a cloud provider may face, we set up experiments in the simulator mentioned in [22]. In this set of experiments, we vary the size of the target cluster and HeteroScore $\mathcal{H}_c$, then run scheduler simulation for over 10000 instances. The results are shown in Table I.

The results show that for a given HeteroScore, larger clusters tend to be less vulnerable to co-location attacks due to the larger scheduling space of the scheduler. However, attackers can still achieve an acceptable co-location success rate if they can increase the number of attack instances and the HeteroScore of a cluster is high. Unfortunately, commercial cloud infrastructure details, such as numbers and types of servers, are not available to the general public. Therefore, we were not able to calculate the exact HeteroScores for commercial clouds. Nonetheless, due to the size/scale and applications of commercial clouds, they tend to contain heterogeneous servers and a large number of identical servers from each server type, which are factors for a low HeteroScore. Lastly, for smaller and more heterogeneous clusters, co-location attacks and subsequent micro-architectural attacks can be a real threat.

TABLE I: Co-location rates for varying cluster sizes and degree of heterogeneity.

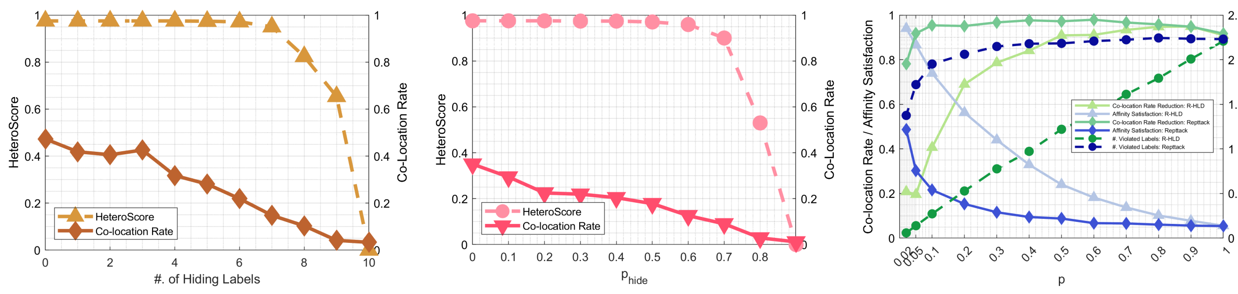
#. of Nodes	$\mathcal{H}_c$	Co-location Rate	
		1-Instance Attack	10-Instance Attack
100	0.9878	51.16%	92.65%
	0.9497	34.04%	65.88%
	0.7126	11.10%	37.42%
	0.4070	4.07%	26.33%
	0	1.12%	8.09%
1,000	0.9975	41.53%	79.20%
	0.9522	15.89%	37.30%
	0.7381	13.78%	22.74%
	0.4084	7.74%	12.35%
	0	1.90%	3.23%
10,000	0.9988	19.88%	65.23%
	0.9437	14.06%	44.09%
	0.7335	7.33%	28.81%
	0.4138	6.42%	9.40%
	0	0.80%	0.87%

3) Effects of Proposed Mitigation Technologies: This part shows how our proposed mitigation technology can help reduce the co-location threat in a real cluster. We apply HLD and R-HLD respectively, calculate HeteroScore for each parameter setting and collect co-location results on our cluster. In our experiment, to implement mitigation, we only add corresponding rules to our generation scripts of `.yaml` files. In the generation step, these rules can be considered an intermediate layer in the scheduler that processes user-submitted scheduling requirements. For HLD, we deterministically delete specified label description fields in `.yaml` files. For R-HLD, we delete label description fields with a preset probability. The results are provided in Figure 8.

To evaluate HLD, we change the number of labels hidden from users. Results in Figure 8a show that HLD can decrease the HeteroScore in our cluster, and applying HLD decreases the co-location rate attackers can achieve hence protecting the cluster from co-location attacks. Furthermore, the co-location rate drops as more labels are hidden, eventually reaching a value near 0 when all labels are hidden, and the cluster becomes homogeneous.

Similarly, in Figure 8b we change the parameter p_{hide} in R-HLD and present HeteroScores as well as co-location rates. We can see that R-HLD has similar effects on $\mathcal{H}_c$ and achieves similar co-location reduction trends. As p_{hide} increases, co-location rate drops and reaches a value close to 0 when on average 90% of labels are hidden ($p_{\text{hide}} = 0.9$) and the cluster becomes nearly homogeneous.

We also compare R-HLD with the mitigation strategy proposed in [22], as shown in Figure 8c, since both rely on bringing randomization controlled by probability parameters (e.g. p_{hide} in R-HLD). It is worth noting that in [22], the corresponding parameter p is related to the probability of skipping node filtering, which is not directly related to labels, unlike in R-HLD. Same as [22], we perform behavioral simulation, use the percentage of user applications that can meet their scheduling requirements (named Affinity Satisfaction) and the number of violated labels as metrics for cost. We use co-location rate reduction (%) to represent mitigation performance. We can see that the method proposed in [22]



(a) Results of applying HLD. (b) Results of applying R-HLD. (c) R-HLD vs mitigation proposed in [22].

Fig. 8: Effects of proposed mitigation strategies HLD and R-HLD. In (a), how co-location rates change when more labels are involved in HLD. In (b), how co-location rates change with p_{hide}. In (c), we compare our method with literature [22].

is more sensitive to parameter changes when p is small: with small parameter changes, the mitigation method in [22] brings higher reduction and higher costs. But both methods achieve similar performance when costs are similar (when p is close to 1). Compared to [22], curves of R-HLD are more smooth, indicating better controllability.

For evaluating the loss of performance, power, etc., at cloud scale, obtaining representative/accurate results is very challenging. This is not only due to the limited access to heterogeneous clusters in CloudLab, but also because there are various ways for cluster managers to configure nodes, and different configurations may result in totally different evaluation results. We only show an example regarding performance loss in the next part.

C. Case Study of University Clusters

To the best of our knowledge, infrastructure information regarding numbers of machine types in a cluster is not available to the public. Thus, we utilize two university clusters from High Performance Computing Center of University of California, Davis (HPC1 Cluster and Genome Center Bioinformatics Cluster, denoted as Cluster A and Cluster B respectively) to perform real-world data analysis and try to provide an example of performance loss due to mitigation.

Both Cluster A and Cluster B are managed by SLURM [54], with 73 and 194 servers available to users, respectively at our experiment time. In Cluster A, users are able to select nodes by choosing partitions when submitting tasks to the cluster. There are 6 partitions in the cluster (priority partitions: Low, Med, High; GPU partitions: GPU, QiGPU, ZdingGPU), and these partitions can overlap. All partitions are defined by the cluster manager. In our processing, we split the partition to 6 one-hot-encoding labels and perform HeteroScore evaluation.

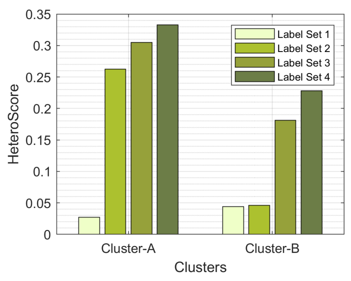
In Cluster B, however, machines are divided into 6 non-overlapping manager-defined partitions. Since these partitions are non-overlapping, we can directly plugin our proposed metric to perform HeteroScore evaluation without the need to split labels. Besides this dimension, it also allows users to specify GPU resources to use and choose to run on nodes with different network bandwidths.

1) HeteroScore Evaluation: We perform our HeteroScore evaluation on the two clusters. For each cluster, we expose different sets of labels and calculate the resulting HeteroScore $\mathcal{H}_c$. The results are shown in Figure 9.

For Cluster A, we choose to expose 4 sets of labels and calculate the corresponding $\mathcal{H}_c$s respectively. We can see from Figure 9a that most nodes are included in Partitions named Low and Med, since the resulting HeteroScore is low when only exposing these two labels to users. By exposing GPU-related labels (GPU, QiGPU and ZdingGPU) to users, HeteroScore increases significantly. Also, exposing High Partition for users to choose from also contributes to the heterogeneity of the cluster. After combining all labels, we obtain the highest HeteroScore, since all selected heterogeneity information is now exposed to users.

For Cluster B, we follow the same procedure and expose 4 sets of labels to users as well. We can see from Figure 9 that the Partitions and GPU dimensions do not contribute much to heterogeneity. But with the Bandwidth dimension introduced, HeteroScore increases significantly. Same as in Cluster A, after combining all labels, we obtain the highest HeteroScore. From the HeteroScore results we obtain, we can see that both clusters are relatively homogeneous. However, in a cluster where there are users with more diversified needs, service providers will introduce more diversified hardware and expose these features to users. The HeteroScore will hence be higher.

2) Performance Impact of Mitigation: To give a real-world example of a performance impact when employing our proposed mitigation strategies, we conduct an experiment on Cluster B. Due to the access to heterogeneous hardware, we only select to hide one label related to network performance (Bandwidth) and show the impact of our mitigation on the performance of different benchmarks. We utilize a network benchmark that downloads contents from the Internet (called Download Benchmark) to represent applications that are affected by this label and a computation benchmark from Rodinia benchamrk set (Hotspot OpenCL build) [16] to represent applications that are not affected by this label. These applications are run on single machines and are submitted in batch. Our goal is to provide a real-world case study for the performance impact of hiding one label, where applications that are either affected or not affected by this label are involved.



(a)

Cluster	Label Set	Labels
Cluster A	1	Partition: Low, Partition: Med
	2	GPU-related labels
	3	Partition: High
	4	Partition: Low, Partition: Med, Partition: High, GPU-related labels
Cluster B	1	Partitions
	2	Partitions, GPU
	3	Bandwidth
	4	Partitions, Bandwidth, GPU

(b)

Fig. 9: Results obtained from our university clusters. (a) $\mathcal{H}_c$ calculation results. (b) Label information.

Within the Bandwidth dimension, there are two possible values. There are 26 nodes labelled as "10G Bandwidth" (referred to as "matching machine" later) and 131 nodes labelled as "1000M Bandwidth" (called "non-matching machine" later). Download Benchmark downloads a 10MB file for 100 times to stress the network. Rodinia-Hostpot, on the other hand, is a network-agnostic application. We deploy these benchmarks to different sets of machines and collect run time and schedule time data. In this way, we simulate a scenario where cloud manager selects Bandwidth label to hide. We are able to summarize the performance impact of hiding a single label from the collected data. The results are shown in Figure 10, Figure 11 and Figure 12.

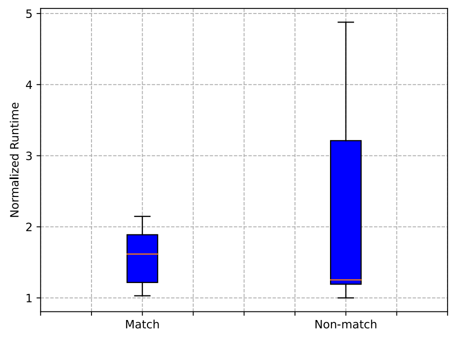


Fig. 10: Normalized runtime of benchmarks.

Figure 10 shows the comparison of our benchmarks running on matching machines and non-matching machines. We show the normalized run time of our benchmarks. For each run time data point, we divide itself by the average run time of its benchmark class. We can see that the normalized run time is more concentrated on matching machines, which means more applications are able to benefit from heterogeneity-aware scheduling decisions. However, the lowest normalized run times are similar and the median normalized run time of non-matching group is even lower, possibly due to the large portion of benchmarks that are not affected by this label being hidden.

Figure 11 shows the overhead of execution time brought by our label-hiding defense. In our experiment, we vary the CPU and memory configuration and deploy on matching and non-matching machines respectively. The overhead data is calculated by run time on non-matching machine divided by

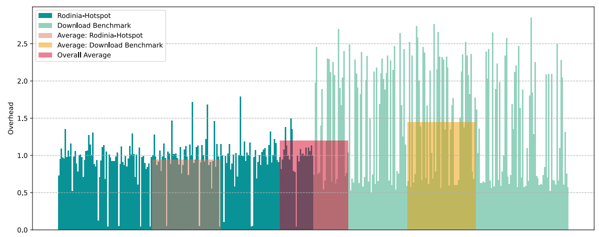


Fig. 11: Label-hiding overhead comparison.

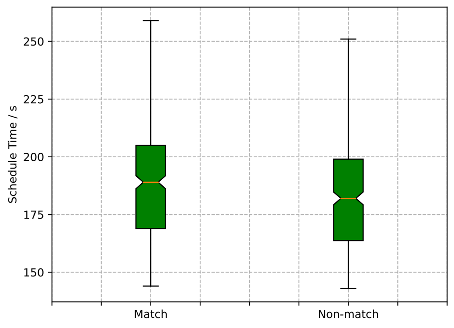


Fig. 12: Scheduling time analysis

run time on matching machine. We can see that for our download benchmark, overhead is higher, but in some cases due to the randomness of execution environments overhead could be lower than 1x. For Rodinia-Hotspot benchmark, the overhead is generally around 1x, i.e., hiding the selected label does not affect performance. The reported mean overheads are 0.95x for Rodinia-Hotspot, 1.45x for Download Benchmark and 1.20x for all applications. Download Benchmark seems to suffer the most from the defense mechanism; however, as we mentioned before, performance-sensitive users can still specify hard requirements to schedulers in our defense, hence this is only the maximum possible overhead for Download Benchmark. We conclude that as a selective defense mechanism, since not all user instances are affected by the selected label and users have the right to choose whether or not to be restricted by the defense strategy, the overhead can be mild.

Figure 12 shows the results of scheduling time. As expected, the time spent on scheduling is slightly higher for applications that run on matching machines, since the heterogeneity-aware decision process is relatively more complicated. This can be reflected on elapsed time (time spent in the scheduling queue plus execution time). It is interesting to note that for applications that are only executed for a short period of time, this difference in scheduling time can compensate for the performance loss brought by hiding labels. According to our results, since heterogeneous-aware placement can take a longer time, for short-lived applications the elapsed time may not be negatively affected much or can even be improved. Considering elapsed time performance, the cost of our mitigation can be even lower in a real-world scenario.

VII. Discussion

A. Choices of Mitigation Strategies

We have proposed two mitigation technologies in this paper, namely HLD and R-HLD. We have shown that both mitigation techniques achieve similar performance in decreasing co-location rates. However, it is not clear how to select from the two methods.

In HLD, during the scheduling process, certain labels are hidden from users; hence the costs of performance, power, etc., are determined and in control by cluster managers. Cluster managers can select which features to hide to reach the balance point between sacrifice and decrease in co-location rate. In R-HLD, scheduling constraints of users are randomly ignored; hence the costs on metrics like performance, power, etc., are not deterministic. However, since the probabilities of labels being hidden are equivalent and are solely determined by p_{hide}, R-HLD can achieve a more balanced sacrifice.

Designers can choose to use HLD or R-HLD based on considerations of the cost. For example, if certain scheduling constraints are not important and will not result in a serious violation of the service level agreement (SLA), HLD can be selected. If balancing the cost for not meeting scheduling constraints is more important, R-HLD can be selected. In practice, HLD and R-HLD can actually be combined. For example, a set of labels can be chosen to apply R-HLD, i.e., only labels in this chosen set will be randomly selected and be hidden from users. Also, p_{hide} does not need to be the same for all labels. By setting different p_{hide} cluster managers can define which labels are more important and hence has a lower probability of being hidden, while certain labels can have a higher probability of being hidden to reduce HeteroScore. By exploring this design space of mitigation strategy, cluster managers can find the most suitable solution to trade-off reduction in co-location attack threat and the cost of mitigation.

Currently, our implementation of defense mechanisms is only at the prototype stage. Our future work will integrate our mitigation strategy into a real cloud framework and make it available to the community.

B. Additional Information Related to Schedulers

During our experiments, we observed that the scheduling algorithm in SLURM [54] in the university clusters presents timing locality, i.e., the scheduler tends to assign the same nodes for applications that are scheduled close in time, which has also been reported in other infrastructures [45]. Due to this reason, we did not perform a co-location attack evaluation on the two SLURM-managed clusters. Also, we observed that some nodes are constantly down/unschedulable, which further narrows down the search space of a scheduler. Combined with similar features, attackers can achieve an even higher co-location rate in a heterogeneous cluster which can be a real threat to non-malicious users.

VIII. Related Work

Co-location attack is proposed and proved to be a possible threat to cloud users by Ristenpart *et.al* [45]. By using brute-force submission to maximize the chance of co-location and network probing to detect co-location, the authors successfully issued side-channel attacks on victim instances. Ever since that, there have been various works targeting applying micro-architectural attacks to cloud infrastructures [29], [57], as well as defending against micro-architectural attacks on cloud [10], [25], [58], [8], [32], [35], [14], [31], [30], [20].

In previous works regarding co-location micro-architectural attacks on the cloud, only a few papers discussed scheduler-level vulnerabilities. In [11], the authors propose that involving randomness in the scheduling process help defend against co-location attacks. In [49], VM placement vulnerabilities are studied, and different attack strategies are tested, but the security threat is not quantized. In recent years, a few works have discussed scheduler-level co-location attacks. Makrani *et. al* [39] target neural-network-based scheduling strategies and prove that by generating fake traces to mimic micro-architectural traces of victim instances, attackers could increase the chance of getting co-located with victim instances. A similar work [22] targets more widely-used schedulers and showcases that if schedulers allow users to submit certain scheduling constraints, attackers can greatly increase the chance of achieving co-location in a heterogeneous cloud by replicating scheduling constraints submitted by victims.

Previous works on evaluating cloud heterogeneity mainly focus on performance or performance simulation [26], [24], [12], [55]. As far as we know, this study is the first work to quantitatively evaluate the heterogeneous configurations of cloud infrastructure and establish its relation with security.

Our work is also related to heterogeneity-related defense technology. Previous related works focus more on quantitatively evaluating the defense mechanism and use heterogeneity more as a defense instead of a vulnerability. Larsen *et.al* [36] discuss how diversifying software distribution may enhance security. Okhravi *et.al* [44] evaluate factors that can affect dynamic platform defense and come up with generalized quantitative models to calculate metrics like attacker success rate, etc. Carter *et.al* [15] utilize a game theory approach, quantitatively models the probability of success, and prove that optimized platform selection-based dynamic platform defense outperforms randomized strategies. In [27], Hu *et.al* propose a method to utilize heterogeneity as a defense in cyber systems. Similarly, Okhravi *et.al* [43] propose a defense mechanism based on heterogeneity for infrastructure applications. Compared to these works, our work is the first to quantitatively measure heterogeneity in a cloud setting and relate heterogeneity to the emerging micro-architectural vulnerabilities in the cloud.

IX. Conclusion

In this paper, we propose HeteroScore, a metric to evaluate the heterogeneity of a cluster. We define and introduce algorithms to calculate this metric and successfully set up a link between this metric and the security level against co-location attacks. Based on HeteroScore, we propose defense mechanisms and validate the efficiency of such techniques. HeteroScore can be considered a tool to guide the design and configuration of clusters, as well as the design of scheduling algorithms. Its relation with the security of a cluster implies that gain in performance by utilizing heterogeneity comes with security vulnerabilities as costs. Therefore, cluster managers should use HeteroScore as a tool to explore the design space of cluster and scheduling algorithms, determine what information can be securely exposed to users, and properly make trade-offs between security and performance.

Acknowledgment

The work in this paper is partially supported by National Science Foundation grants CNS-2155002 and CNS-2155029. We would like to thank the support of High-Performance Computing (HPC) Center of University of California, Davis. We are also grateful to all anonymous reviewers for their constructive feedback.

References

[1] "Amazon EC2," https://aws.amazon.com/pm/ec2/, 2022, [Online; accessed Apr. 2022].

[2] "Docker Hub," https://hub.docker.com/, 2022, [Online; accessed Apr. 2022].

[3] "google Cloud," https://cloud.google.com/, 2022, [Online; accessed Apr. 2022].

[4] "Kubernetes," https://kubernetes.io/, 2022, [Online; accessed Apr. 2022].

[5] "Microsoft Azure," https://azure.microsoft.com/en-us/, 2022, [Online; accessed Apr. 2022].

[6] "OpenNebula," https://opennebula.io/, 2022, [Online; accessed Apr. 2022].

[7] "OpenStack," https://www.openstack.org/, 2022, [Online; accessed Apr. 2022].

[8] H. Alhulayyil, K. Khalil, S. V. Krishnamurthy, D. Cansever, T. La Porta, and A. Swami, "On the detection of adaptive side-channel attackers in cloud environments," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[9] 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.

[10] M. Azab and M. Eltoweissy, "Migrate: Towards a lightweight moving-target defense against cloud side-channels," in *Proceedings of the IEEE Security and Privacy Workshops (SPW)*, 2016, pp. 96–103.

[11] 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.

[12] M. Bux and U. Leser, "Dynamiccloudsim: Simulating heterogeneity in computational clouds," *Future Generation Computer Systems*, vol. 46, pp. 85–99, 2015.

[13] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. Netto *et al.*, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–38, 2018.

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

[15] K. M. Carter, J. F. Riordan, and H. Okhravi, "A game theoretic approach to strategy determination for dynamic platform defenses," in *Proceedings of the first ACM workshop on moving target defense*, 2014, pp. 21–30.

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the IEEE international symposium on workload characterization (IISWC)*, 2009, pp. 44–54.

[17] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. P. Walters, "Heterogeneous cloud computing," in *Proceedings of the IEEE International Conference on Cluster Computing*, 2011, pp. 378–385.

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

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

[20] 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*, vol. 41, no. 4, pp. 868–881, 2021.

[21] 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)*, 2019, pp. 1–14.

[22] C. Fang, H. Wang, N. Nazari, B. Omidi, A. Sasan, K. N. Khasawneh, S. Rafatirad, and H. Homayoun, "Repttack: Exploiting cloud schedulers to guide co-location attacks," in *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2022.

[23] 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.

[24] A. Gupta, P. Faraboschi, F. Gioachin, L. V. Kale, R. Kaufmann, B.-S. Lee, V. March, D. Milojicic, and C. H. Suen, "Evaluating and improving the performance and scheduling of hpc applications in cloud," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 307–321, 2014.

[25] 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.

[26] W. He, H. Cui, B. Lu, J. Zhao, S. Li, G. Ruan, J. Xue, X. Feng, W. Yang, and Y. Yan, "Hadoop+ modeling and evaluating the heterogeneity for mapreduce applications in heterogeneous clusters," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 143–153.

[27] H. Hu, J. Wu, Z. Wang, and G. Cheng, "Mimic defense: a designed-in cybersecurity defense framework," *IET Information Security*, vol. 12, no. 3, pp. 226–237, 2018.

[28] 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.

[29] 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.

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

[31] M. Kayaalp, K. N. Khasawneh, H. A. Esfeden, J. Elwell, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "Ric: Relaxed inclusion caches for mitigating llc side-channel attacks," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[32] 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)*, 2019, pp. 1–6.

[33] 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.

[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)*, 2020, pp. 39–53.

[36] P. Larsen, S. Brunthaler, and M. Franz, "Security through diversity: Are we there yet?" *IEEE Security & Privacy*, vol. 12, no. 2, pp. 28–35, 2013.

[37] 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.

[38] X. Lou, T. Zhang, J. Jiang, and Y. Zhang, "A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021.

[39] 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 *Proceedings of the International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2021, pp. 1–13.

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

[41] 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)*, 2020, pp. 1466–1482.

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

[43] H. Okhravi, A. Comella, E. Robinson, S. Yannalfo, P. Michaleas, and J. Haines, "Creating a cyber moving target for critical infrastructure applications," in *International Conference on Critical Infrastructure Protection*. Springer, 2011, pp. 107–123.

[44] H. Okhravi, J. Riordan, and K. Carter, "Quantitative evaluation of dynamic platform techniques as a defensive mechanism," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 405–425.

[45] 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 (CCS)*, 2009, pp. 199–212.

[46] 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.

[47] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.

[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] 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.

[50] 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.

[51] F. Xu, F. Liu, and H. Jin, "Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2470–2483, 2015.

[52] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," https://github.com/0xADE1A1DE/Mastik, 2016, [Online; accessed Apr. 2022].

[53] 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.

[54] 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.

[55] M. Zakarya and L. Gillam, "Modelling resource heterogeneities in cloud simulations and quantifying their accuracy," *Simulation Modelling Practice and Theory*, vol. 94, pp. 43–65, 2019.

[56] Y. Zha and J. Li, "When application-specific ISA meets fpgas: a multi-layer virtualization framework for heterogeneous cloud FPGAs," in *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 123–134.

[57] 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.

[58] 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.