2016 IEEE 35th Symposium on Reliable Distributed Systems

Tale of Tails: Anomaly Avoidance in Data Centers

Ji Xue
College of William and Mary
Virginia, USA
xuejimic@cs.wm.edu

Robert Birke
IBM Research Zurich Lab
Ruschlikon, Switzerland
bir@zurich.ibm.com

Lydia Y. Chen
IBM Research Zurich Lab
Ruschlikon, Switzerland
yic@zurich.ibm.com

Evgenia Smirni
College of William and Mary
Virginia, USA
esmirni@cs.wm.edu

***Abstract*—It is a common practice that today's cloud data centers guard the performance by monitoring the resource usage, e.g., CPU and RAM, and issuing anomaly tickets whenever detecting usages exceeding predefined target values. Ensuring free of such usage anomaly can be extremely challenging, while catering to a large amount of virtual machines (VMs) showing bursty workloads on a limited amount of physical resource. Using resource usage data from production data centers that consist of more than 6K physical machines hosting more than 80K VMs, we identify statistic properties of anomaly instances (AIs) on physical servers, highlighting their burst duration and potential root causes. To strike a tradeoff between a strong performance guarantee and resource provisions, we propose a tail-driven anomaly avoidance policy for boxes, *TailGuard*, which allows a small fraction of AIs, e.g., 5% of usages can be above the target value, and still avoid severe performance degradation, typically caused by a burst of continuous AI. Specifically, *TailGuard* first introduces a novel usage tail prediction that explores the similarity patterns across a great number of boxes within a very recent history, and then redistributes the server load in an online fashion by proactive VM cloning and reactive load balancing. Evaluation results show that *TailGuard* can not only achieve an accuracy comparable with prediction methodology that relies on long history of usage data but also dramatically reduce the number of CPU AIs by 60%, with a tenfold reduction of their duration, from more than 25 time windows to only 2.**

I. Introduction

Ticket issuing is widely used in today's data centers for performance anomaly detection [1], [2]. Performance tickets are issued either automatically by the system (e.g., when high resource usage is detected and signals potential deteriorated user experience) or by the users themselves (e.g., after the system is perceived slow or unresponsive). Ticket resolution is an expensive process as it usually requires manual labor for root-cause analysis [3]. Transient resource usage that grows beyond a predefined target value is considered an anomaly instance (AI) as it threatens user performance at tails and signals unsteady system operation. Such usage fluctuations are the result of aggressive multiplexing of multiple VMs across physical servers, termed boxes, competing for limited physical resources and the dynamic nature of each VM workload [4].

We analyze data center usage time series of a major vendor. The trace data correspond to 6K physical boxes serving more than 80K VMs over a time period of a week. Our analysis shows that anomaly instances fall into two categories: single AI where the duration of the anomaly is short and continuous AI where the duration of the anomaly is long. Figure 1 gives an overview of usage anomaly instances. Resource usage is typically reported within discrete time windows (e.g., in our trace each time window equals to 15 minutes). While the usage series has fluctuations across time, it goes beyond the target value three times. Twice the usage exceeds the target value for a short time of a single time window (single AI). Once the usage exceeds the target value for a long time, corresponding to multiple consecutive time windows (continuous AI). The trace characterization points to one more important factor that distinguishes single and continuous AIs: we find that not only the *duration* but also the *magnitude* of a continuous AI is larger than the single AI. While single AIs may be considered relatively harmless, continuous AIs have the potential to significantly undermine the user perceived performance.

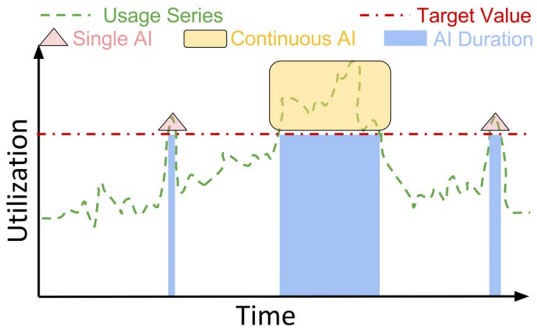


Fig. 1: How to determine a usage anomaly instance. An example on CPU usage.

In this paper, we develop a tail-driven anomaly avoidance policy, termed *TailGuard*, which aims to ensure at most a certain fraction of AIs and eliminate or at least drastically reduce continuous AIs in physical servers. For example, *TailGuard* tries to ensure that the box CPU usage is below the predefined target (e.g., 60%) for 95 percent of the time, meaning that the 95%ile of usage, the so-called tail usage, is below the target. To motivate the design of *TailGuard*, we first do a detailed, post-hoc workload characterization study of usage time series (for both CPU and RAM) in production data centers. *TailGuard* particularly consists of two steps: a light-weight tail usage prediction method that explores the power of vast number of last values of usages and a VM cloning strategy that redistributes the box CPU and RAM loads based on the prediction. Overall, this work makes three main contributions: AI characterization, usage tail prediction, and anomaly mitigation.

This characterization analysis allows us to view the statistical characteristics of usage time series and focuses on the properties of their tails. The key findings are as follows: 1) the culprit of continuous CPU AIs is VM consolidation.

1060-9857/16 $31.00 © 2016 IEEE
DOI 10.1109/SRDS.2016.19

2) CPU tail usage is highly correlated to the mean CPU usage of physical servers and follows a Normal distribution. 3) RAM anomalies do not relate strongly to VM consolidation in contrast to CPU anomalies.

Based on these observations, the proposed *TailGuard* avoids AI, particularly CPU, by predicting the tail usage of box CPU and redistributing VMs across boxes in an online fashion. *TailGuard* first predicts the box CPU tail usage, e.g., 95%ile, by capturing the steady state of tail distribution with respect to different levels of mean usage. In contrast to conventional time series prediction, *TailGuard* is not only very light-weight using very recent data, e.g., past day, instead of long history of usage series, but also aware of the resource availability at the tenants. Secondly, based on tail predictions and the level of resource availability for each tenant, *TailGuard* redistributes box loads by proactively creating and placing VM clones so as to ensure the box tail usage does not exceed the target value. The VM cloning strategy combines the advantage of VM migration and load balancing at the cost of duplicating the VM memory footprint.

The proposed *TailGuard* is evaluated in detail using trace driven simulation. Results are summarized as follows: 1) the proposed prediction method of tail usages is computationally much cheaper compared to accurate but expensive time-series predictions (e.g., neural networks), while balancing tail usages across boxes; 2) VM cloning achieves a ten percent higher reduction in CPU tail target violations than classic VM migration. It is also noteworthy that, thanks to the reactive load balancing, our method achieves CPU tail usage reduction with minimal RAM usage violation increase (up to 3 percent), which is lower than the one achieved by proactive VM migration. Even by allowing target violations in up to 5 percent of the time windows, *TailGuard* not only reduces the number of CPU AIs by 60 percent, but also mitigates the duration of continuous CPU AIs dramatically, from a maximum duration of over 25 time windows to 2 time windows.

The outline of this work is as follows: Section II provides a characterization study on the CPU and RAM AIs. We propose *TailGuard*, describing tail usage prediction method and a VM cloning strategy in Section III. An extensive evaluation of our proposed method for CPU AI reduction on production traces is discussed in Section IV. Section V presents related work, followed by conclusions in Section VI.

II. Characterization

The trace considered here comes from production data centers serving various industries, including banking, pharmaceutical, IT, consulting, and retail, and contains CPU and RAM utilization at a time granularity of 15 minutes for 6K physical boxes hosting more than 80K VMs during a 7-day period from April 3, 2015 to April 9, 2015. Naturally, the level of consolidation is very high, i.e., on average 10 VMs are consolidated within a single physical box [5].

A. Overview

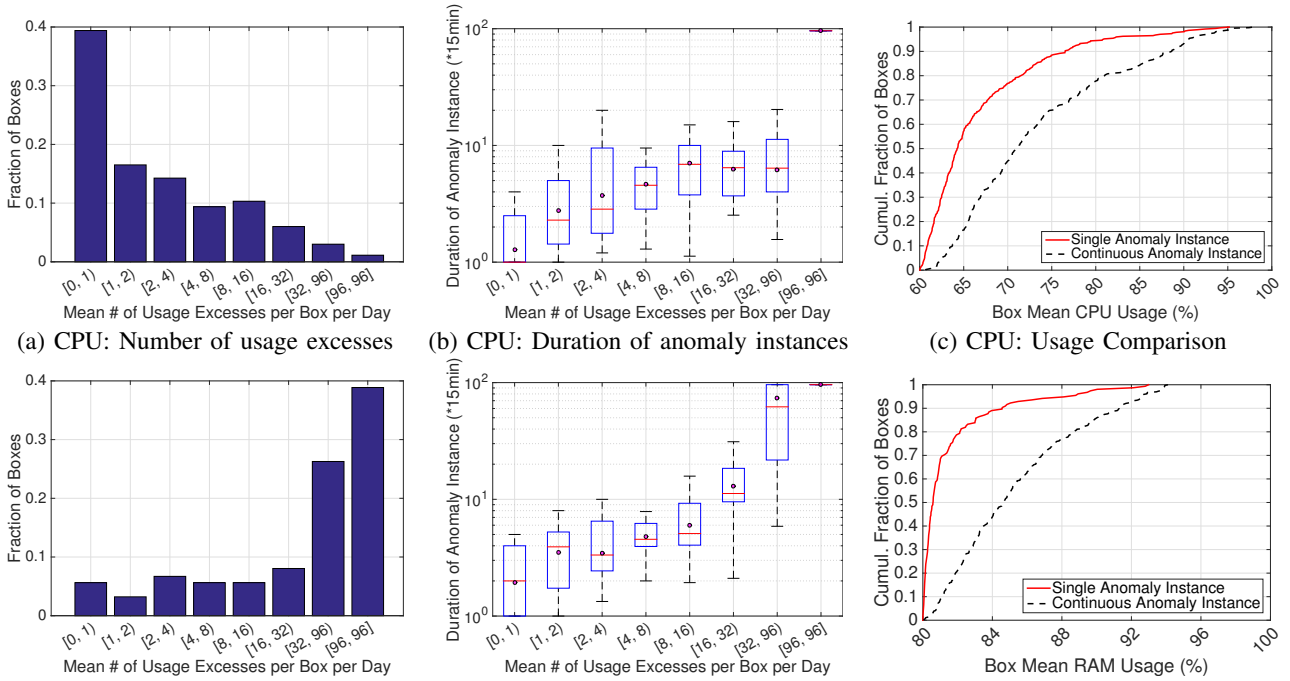
We first look into daily statistics of anomaly instances, assuming that an anomaly instance is triggered when a resource usage exceeds a *target* [4]: 60% for CPU and 80% for RAM. Figure 2(a) illustrates the empirical PDF of the daily average number of CPU excesses per physical box. Note that each excess corresponds to the mean utilization across 15-minute time window being above the target value. 40% of the boxes have a daily average number of excesses below one. After that the fraction of boxes rapidly decays with only 3% of boxes which experience over 32 excesses per day (i.e., for more than 8 hours per day). Figure 2(b) focuses on whether these excesses are single or continuous AIs. The boxplots in this figure show the 25^{th}, 50^{th}, and 75^{th} percentiles of the anomaly duration, the whiskers correspond to extremes of the distribution, and the dots represent the average. Note that the y-axis is in logscale and in units of 15-minute time windows. The figure clearly illustrates that most anomaly instances are continuous, i.e., longer than one time window. Figure 2(c) illustrates the relationship between the box mean CPU usage across the excesses and the type of anomaly that it experiences. Here, the CDFs of single and continuous anomaly instances are presented: it is clear that continuous anomaly instances have higher usages than single ones.

Figure 2(d)-(f) presents similar results as Figure 2(a)-(c) but for RAM. Figure 2(d) illustrates the empirical PDF of the average number of RAM excesses per box per day and shows that RAM excesses seldom come alone: from at least 32 (26% of boxes) up to 96 (40% of boxes). Consequently, the boxplots of Figure 2(e) illustrate that these anomalies are mostly continuous. Finally, Figure 2(f) shows that the CDF of single and continuous anomaly instances for different box mean RAM usages across the anomaly instances. RAM continuous anomaly instances have higher usages than single ones and the difference is even larger comparing to CPU anomalies. In summary, Figure 2 illustrates that for both CPU and RAM anomaly instances, it is the continuous ones that tend to exceed significantly the target value. Hence, continuous anomalies have the potential to harm performance for a long period of time. This motivates us to look in detail into the tail usages for CPU and RAM.

B. Root Cause Analysis for Box Anomaly Instance

A natural question is whether there is any relationship between the VM consolidation level and the number of CPU/RAM anomaly instances. Figure 3(a) presents boxplots that show the 25^{th}, 50^{th}, and 75^{th} percentiles (boxes), the extremes of the distributions (whiskers) and the means (dots) of the number of CPU usage excesses for different ranges of VM consolidation levels on the x-axis. Here we define consolidation level as the number of collocated VMs. Figure 3(b) presents the same information but for RAM anomaly instances. It is clear that higher VM consolidation levels result in more CPU anomaly instances while RAM anomalies do not show a strong relationship with the VM consolidation level.

Figures 3(c) and (d) illustrate the probability that at least one excess is observed on a VM residing on a given box that is also observed an excess for CPU and RAM usage, respectively. Note that these probabilities are reported as a function of the VM consolidation level. The figures illustrate that there is a strong relationship among the box anomaly instances and a VM anomaly instance for CPU, as shown by probabilities higher than 0.9 for all the consolidation levels. For RAM, the probabilities are less than 0.02 for all consolidation levels. The figures illustrate that there is a clear relationship among the



(a) CPU: Number of usage excesses (b) CPU: Duration of anomaly instances (c) CPU: Usage Comparison

(d) RAM: Number of usage excesses (e) RAM: Duration of anomaly instances (f) RAM: Usage Comparison

Fig. 2: CPU and RAM: overview of anomaly instances.

hosted VM anomalies and the box anomalies for CPU; this relationship is clearly not present for RAM.

To summarize, the above two figures illustrate that CPU anomaly instances in boxes strongly depend on the VM CPU usage within these boxes, while this is not the case for RAM anomaly instances which are triggered mainly because of boxes themselves and not necessarily their residing VMs. This motivates us to focus on CPU AIs rather than RAM ones.

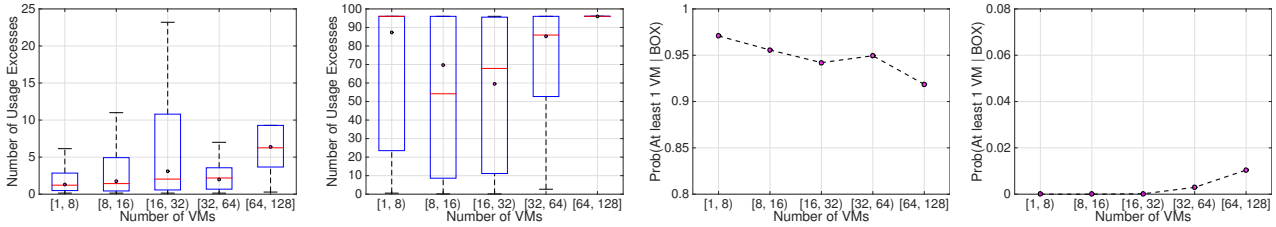
C. Is CPU Usage Balanced?

In a private cloud data center, which is a single *tenant* environment, the hardware, storage and network are dedicated to a single client or company. A *tenant* can be seen as a cluster of boxes, which allows customized placement of VMs and assignment of resources. Figure 4(a) plots the CDF of box mean CPU usage across tenants. For selected means, we also plot the standard deviation. The figure shows that the average usage of boxes for the majority of tenants is low: 90% of tenants have an average CPU usage less than 35%. Yet, the standard deviation is large, suggesting that there is significant load imbalance across boxes. The same imbalance is observed for the CPU tail usages as shown in Figure 4(b) that plots the CDF of the 95%ile of box CPU usage across tenants. If we are able to predict the tail usages, then based on this information we can devise a balancing algorithm that reduces tail usage and consequently anomaly instances.

A simple common approach for predicting moments of time series makes use of the last value prediction, which predicts the future using the most recent observations. In Figure 5, we demonstrate that the last value prediction works well for predicting the mean usage, but not for the tail based on their coefficient of variation (C.V.), which is equal to standard deviation divided by mean. The C.V. allows us to combine the information on the mean and standard deviation into a single value. Figure 5(a) shows the CDF of the C.V. of the daily box mean and 95%ile usage computed over one week across all boxes. It is clear that the box tail usages show higher C.V. values, on average 0.5, than the box mean usages, on average 0.12, indicating that the mean usages are more constant over time. This is why applying simple last value prediction results in low prediction errors for the mean usages and high prediction errors for the tail usages. This is clearly shown in Figure 5(b) reporting the CDF of absolute percentage error (APE)[1] of the predictions. This observation motivates us to look at the relationship between the mean (rather constant) and tail (highly variable) usages, as to predict the tail usage via the mean usage predicted accurately using the last value.

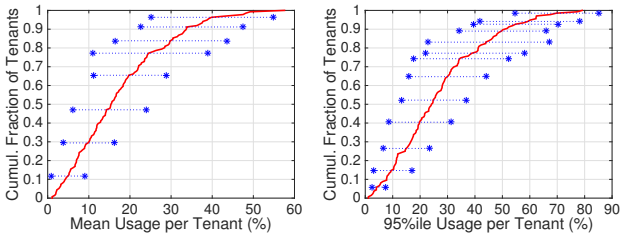
To achieve this goal we divide the boxes into bins based on their mean CPU usage and compute within each bin the distribution of the CPU tail usages across all boxes falling into the same bin. Figure 6 illustrates the resulting PDFs for the 95%ile of CPU usages with a bin width of $\pm 3\%$. The figure shows that for each bin the tail usage distribution resembles a Normal distribution. For example, for the bin corresponding to a mean CPU usage equal to $(60 \pm 3)\%$, we see that average and standard deviation of a fitted Normal distribution of the 95%ile CPU usages are 79.9% and 8.2%, respectively. This allows us to propose an anomaly instance reduction method

[1] $APE = \frac{|Actual - Prediction|}{Actual}$



(a) CPU: Consolidation level (b) RAM: Consolidation level (c) CPU: Prob(VM|Box) (d) RAM: Prob(VM|Box)

Fig. 3: CPU and RAM: root cause analysis for the box anomaly instance.



(a) Mean (b) Tail

Fig. 4: CDF of box CPU usage per tenant: mean and tail.

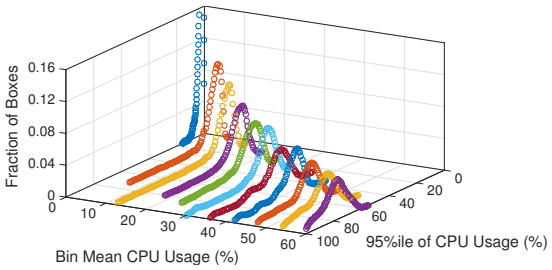
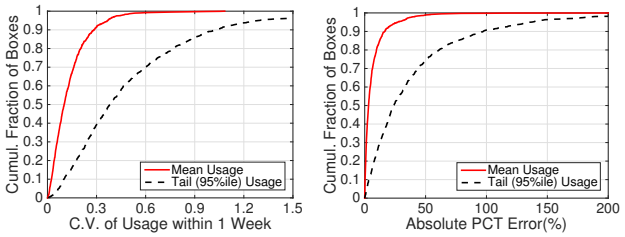


Fig. 6: PDF of box tail usages for different mean usage bins.



(a) C.V. (b) Prediction Error

Fig. 5: Predict box CPU mean and tail usages using the most recent day's observation.

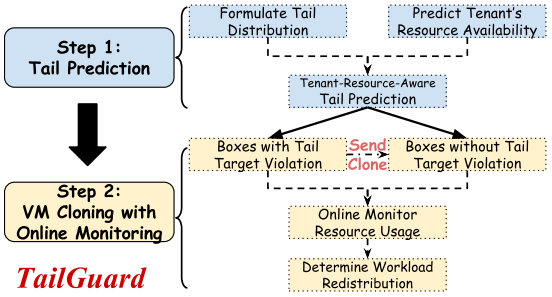


Fig. 7: Overview of *TailGuard* to avoid box CPU tail violation.

based on the mean usage and the (already known) distribution of the tail usages that correspond to this mean usage level.

III. *TailGuard* Policy

In this section, we present *TailGuard*, a tail-driven anomaly avoidance policy, that aims to enhance datacenter tenants' dependability by continuously ensuring their box CPU tail usages are below a predefined target value. To avoid suffering from continuous AIs and over-reacting to spontaneous single CPU AIs, *TailGuard* focuses on bounding the CPU tail usages for each tenant. *TailGuard* proactively manages the CPU and RAM usages of the boxes by intelligently distributing their load, i.e., learning from the past, predicting the future, and actuating at the present. *TailGuard* is composed of two key steps: (i) CPU tail usage prediction and (ii) box load redistribution via VM cloning and online monitoring for workload management. The workflow of the proposed approach for box CPU tail usage reduction is shown in Figure 7. The focus of the tail prediction module is three-fold: prediction accuracy, computational efficiency, and awareness of the overall resource availability of tenants. To achieve good accuracy with low computation overhead and also low requirement of data of the immediate past of a single tenant, *TailGuard* leverages trace data across *all* tenants' boxes, by constructing the empirical distribution of tail usages. We advocate to use simple statistics related to such a distribution, e.g., the mean and standard deviation, to predict the future tail usage of each box. In addition, *TailGuard* determines the prediction scheme, i.e., the specific statistics related to the distribution of tail usages, depending on the resource availability of each tenant.

The tail prediction is used as input for redistributing CPU and RAM loads across boxes in each tenant such that their box CPU AIs, particularly the continuous ones, are mitigated and avoided, with no or only few additional RAM usage violations happening. We propose a tail redistribution policy via VM cloning, essentially by combining the ideas of migration and load dispatching. Based on the CPU tail usage prediction for all boxes belonging to a tenant, *TailGuard* first proactively creates VM clones for boxes that need to reduce their loads,

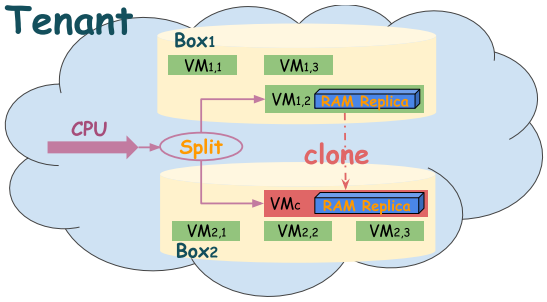


Fig. 8: Overview of VM cloning and workload distribution.

and places clones on boxes with spare capacity. *TailGuard* then dispatches the CPU loads across original and cloned VMs, while assuming memory loads are replicated on both original and cloned ones. The overview of the proposed VM cloning strategy is presented in Figure 8.

After VM cloning, *TailGuard* enables online monitoring of CPU and RAM usage for each box. When unexpected CPU or RAM usages are detected in the boxes with VM clones, *TailGuard* reactively redistributes the workloads between the cloned and original VMs, such that the boxes with VM clones stay with as few CPU AIs or RAM violations as possible.

A. Predicting Box CPU Tail Usage

First we illustrate how *TailGuard* predicts individual box CPU tail usage, based on the distribution of tails observed from a large population of boxes. Our solution is motivated by the findings in Section II: CPU tail usages are variable across time, while mean CPU usages remain constant, see Figure 5. In addition, CPU tail usages appear to follow Normal distributions after binning boxes based on their mean usages as reported in Figure 6. We incorporate the resource availability of each tenant, as an indicator that determines whether the tail prediction should be aggressive or conservative, i.e., the best safety margin to use.

Our objective is to predict the tail usage, L, for a given box by the mean and standard deviation of the tail distribution it belongs to. As such, we can write

$$L = \overline{T} + \alpha S_T \tag{1}$$

where $\overline{T}$ and S_T denote the mean and standard deviation of tail distribution where the box belongs to, and α is a safety margin for the tail prediction based on the resource availability of the tenant. For tenants with abundant resources, higher α values allow to over-estimate the VM resources. This allows the use of more spare resources from boxes without tail target violations and a stronger tail target violation avoidance. For tenants with scarce resources, lower α values allow for more conservative estimates of the VM resources. This results in less requests of spare capacity from boxes without tail target violations protecting such boxes from potential tail target violations. We first explain how to obtain the box tail distributions by binning, and then extract the mean and standard deviation of the tail distribution per bin, to finally derive the tenant-resource-aware (TRA) tail prediction based on the properties of tail distribution.

1) Finding the Tail Distribution: Here, we describe how to bin boxes by their mean usages such that we can obtain $\overline{T}$ and S_T (of the tail distributions) as a function of their mean usages. Figure 6 shows how binning tail usages from all boxes by their mean usages can result into empirical distributions that resemble the Normal distribution. We experiment with different bin widths of mean usages, e.g., $\pm 1\%$, $\pm 2\%$, $\pm 3\%$ (as used in Figure 6) or higher. Different bin widths result in different number of samples in each bin. On the one hand our objective is to find a bin width that is big enough to contain a sufficient number of samples in each bin to be statistically significant, such that the box tails in the majority of bins follow a Normal distribution. On the other hand we want the bin width to be small to have a stronger relationship between the mean and tail usages. Specifically, we compute a pair (M, T) for each box and for *all* tenants, where M represents the mean usage and T is the tail usage, and we bin them by their M values. From the data set considered in this work, we empirically conclude that using a bin width of $\pm 1\%$ is already sufficient such that all bins contain at least 30 tail samples [6], and more than 90% of the bins follow a Normal distribution as verified by the Kolmogorov-Smirnov test [7].

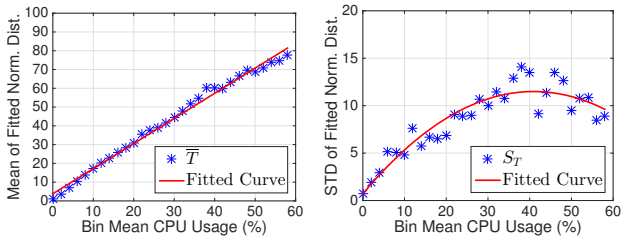
We then use the tail binning results to answer the following question: can we find a compact representation to describe the relationship among the bins, regarding to statistics of tail usages. For each bin, we extract two sets of pairs: (i) $(\overline{M}, \overline{T})$ – the average of M, namely bin mean, and average of T, and (ii) $(\overline{M}, S_T)$ – the average of M, and standard deviation of T. We plot these two sets of pairs for all bins in Figure 9(a) and (b), respectively. Visual inspection of Figure 9 indicates that there exists a linear dependency between the bin mean and the average tail usage across bins and a quadratic dependency between the bin mean and the standard deviation of tails across bins. Given $\overline{T}$ and S_T for each bin mean $\overline{M}$, we fit the two curves using Eq. 2-3. Here a_0 and a_1 are the coefficients of the linear fitting for $\overline{T}$, whereas b_0, b_1, and b_2 are the coefficients of the quadratic fitting for S_T.

$$\overline{T} = f_1(\overline{M}) = a_0 + a_1(\overline{M}) \tag{2}$$

$$S_T = f_2(\overline{M}) = b_0 + b_1(\overline{M}) + b_2(\overline{M})^2 \tag{3}$$

Substituting Eq. 2-3 into Eq. 1, one can thus obtain the closed-form expression for the estimates of the tail usages as a function of its mean usages. Essentially, to predict the CPU tail usage for a box, we need to "learn" the fitting coefficients of functions Eq. 2-3 from the empirical distributons of historical data and then use simple last value prediction of $\overline{M}$, as argued in Section II-C. The final step is to decide the value of α in Eq. 1, which is addressed in the next subsection in the light of the overall resource availability of tenants.

2) TRA Tail Prediction: When considering Eq 1, the most accurate prediction is achieved by setting $\alpha = 0$, i.e., $L = \overline{T}$, whereas higher values of α will tend to over-estimate L providing increasing safety margins. As one of the ultimate goals is to mitigate anomaly instances of tail usages exceeding the target value, we determine the α value for all boxes belonging to the same tenant, based on the tenant's resource availability. Low-CPU-utilized tenants have abundant resources for tail usage reductions, therefore *conservatively*



(a) Fitting of $\overline{T}$ (b) Fitting of S_T

Fig. 9: Fit mean and standard deviation of tail (95%ile) distributions for different bins, using bin mean usages.

predicting tail usages (i.e., higher α safety margins), and *aggressively* re-allocating resources reduce the tail usage below the target value, but do no harm to the original boxes with no tail target violations. High-CPU-utilized tenants have relatively scarce resources, therefore *aggressively* predicting tail usages (i.e., lower α safety margins), and *conservatively* re-allocating resources result in less resource re-allocation for tail usage reduction, but guarantee that the original boxes without tail target violation remain in the 'safe zone'.

The above illustration suggests that α should be customized based on the CPU availability of tenants. We propose to compute α_i for tenant i by considering the target value TG equal to the average tail usage obtained under an optimal case, where all boxes of tenant i undergo perfect load balancing, i.e., every box is equally utilized at the optimal mean value, M_i^*. Basically, one can compute the maximum value of α_i by solving the following equation,

$$TG = f_1(M_i^*) + \alpha_i f_2(M_i^*). \tag{4}$$

where f_1 and f_2 represent the fitted functions of Eq. 2-3.

Additionally, we impose a lower bound on α_i, i.e., $\alpha_i \geq 0$, to ensure that the tail prediction is at least equal to $\overline{T}$. All in all, the TRA tail prediction $\hat{L}_j$ for box j belonging to tenant i is

$$\hat{L}_j = f_1(\hat{M}_j) + \alpha_i f_2(\hat{M}_j), \tag{5}$$

where $\hat{M}_j$ denotes the predicted mean usage for box j. Comparing the predicted tail, $\hat{L}_j$, with the target value indicates if box j is expected to have a tail target violation or not.

B. VM Cloning

Here, we explain how *TailGuard* redistributes the box CPU and RAM loads by VM cloning strategy, such that the probability of boxes in each tenant with CPU tail target violation is minimized. It consists of two steps: (i) proactively create VM clones on boxes at the beginning of the optimization horizon, e.g., one day ahead, and (ii) online monitor the resource usage and reactively redistribute loads among original and cloned VMs. The intention of additional VM clones is to redirect the load from the original box host to the box host of the cloned VM at the cost of duplicated memory footprint. We redistribute incoming requests to the original and cloned VMs via simple Domain Name Server (DNS) based load balancing [8]. This scheme is simple and compatible with a large set of applications.

1) Creating VM Clones: The TRA tail prediction provides two key pieces of information for cloning VMs for each tenant i: the predicted tail usage for each box j, $\hat{L}_j$, and the optimal box mean usage under perfect load balancing, M_i^*. Based on the comparison of the target value and the predicted box tail usage, *TailGuard* decides which boxes need to migrate some workloads through additional VM clones, these boxes constitute the *reduction set*, and which boxes have spare capacity to receive VM clones on top of their existing VMs, so-called *increasing set*.

Creating VM clones: The VM configuration that can benefit most from cloning is the one with large CPU usage and low RAM usage, as cloning has the advantage of distributing CPU load but at the cost of replicating memory usage. Consequently, for each box in the *reduction set*, *TailGuard* ranks their VMs (indexed by k) by the VM's *cloning benefit* ratio, ρ_k, defined as the mean VM CPU usage C_k divided by the mean VM RAM usage R_k computed over all CPU AI points in the training window W, e.g., $W = 1$ day, $\rho_k = \frac{C_k}{R_k}$. The top ranked VMs have heavy CPU loads but low memory footprints.

For each box in the *reduction set* of tenant i, starting from the highest ranked VM, *TailGuard* clones VMs to reduce the box mean CPU usage from its the latest value to M_i^*. We aggressively assume that upon cloning of VM k, the box CPU usage can be reduced by C_k, except for the last cloned VM from this box. When cloning the last VM from box j, removing all the CPU workloads on this VM may reduce the box CPU mean usage to *less than* M_i^*. As a result, for the last VM to be cloned, we only redistribute part of the CPU workload to the cloned one, such that the mean CPU usage of box j is exactly reduced to M_i^*. At each iteration we update the reduction set, as soon as the resulting box mean usage is equal to M_i^*.

Placing VM clones: *TailGuard* ranks the boxes in the *increasing set* of each tenant i by the amount of spare resources, i.e., CPU and RAM, in a descending order with priority first to CPU and then to RAM. The spare CPU is defined as the difference between the current CPU usage and M_i^*, while spare RAM refers to the difference between current RAM usage and RAM target value. When placing the VM clones, *TailGuard* starts from the top-ranked box regarding CPU and RAM spare resources, as long as it has sufficient amount of spare resources to cater to the demands of cloned VMs. Whenever the clone is placed we update the spare resources and the box rank in the increasing set. We terminate the cloning strategy when either the reduction set or the increasing set exhausts.

2) Maintaining Safety Margins: We advocate the use of online monitoring on allocating CPU loads across the original and cloned VMs and terminating clones upon observing RAM violation, to control that cloning does not inadvertently introduce more violations. The focus is to protect the performance of boxes that receive VM clones. *TailGuard* monitors online the CPU and RAM AIs for each box receiving VM clones. Upon detecting a box CPU tail target violation, e.g., accumulating more than 5 CPU excesses given a tail target metric of 95%ile and 96 observation points per day, we reduce the workload of the hosted cloned VMs proportionally to the intensity of the past observed excesses. On the contrary, if a box RAM violation is detected, we directly terminate the cloned VMs, based on the RAM usage rank in a descending order, till RAM usage is less than the RAM target value.

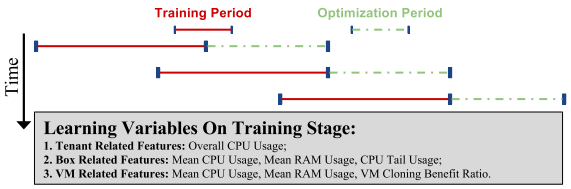
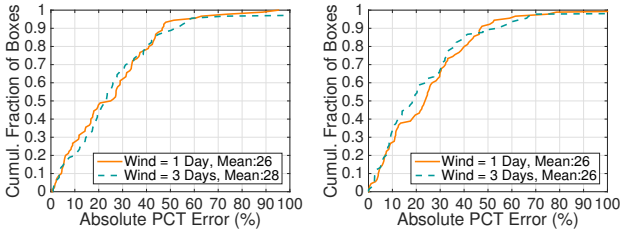


Fig. 10: Dynamic method for tail prediction and VM cloning.



(a) 95%ile (b) 98%ile

Fig. 11: CDF of box CPU tail usage prediction error for different training window sizes.

We note that the online workload distribution scheme proposed here can be implemented via DNS-based load balancing. When a user wants to send a request to a VM, it has to first resolve the hostname to the IP address via a DNS lookup. DNS-based load balancing allows to associate multiple IP addresses to the same hostname so that different DNS lookups for the same hostname return different IP addresses. By updating the entries one can control the workload sent to the original and cloned VM. Since most applications use hostnames rather than IP addresses, DNS-based load balancing is compatible with a large set of applications.

C. Putting All Together

Lastly, we illustrate how *TailGuard* puts the proposed TRA tail prediction, and VM cloning strategy together. The optimization period is one day and the training period of the model is past W days. Past work [2] has shown that most VM migrations in corporate data centers occur once a day and around midnight. We propose implementation for VM cloning to follow this same time frame. Figure 10 illustrates the schematics of the proposed scheme. Particularly, *TailGuard* uses historical data of W days to learn the tail distribution, and derive the closed-form solution of tail estimates in Eq. 5. The higher the value of W is, the longer the historical data used. *TailGuard* then makes VM cloning decisions for the next day. Prior to moving into the next optimization period, *TailGuard* terminates all clones generated in the current period.

We experiment with prediction accuracy of the proposed scheme with two different lengths of W. Specifically, we present the CDF of absolute percentage errors to predict the 95%ile and 98%ile, using one and three days, see Figure 11. As the focus here is to see the impact of the training window length, we set all α_i to 0. We can see that there is no obvious difference in the CPU tail usage predictions between three-day and one-day training. Thanks to large amount of tail data collected from 6K boxes, one-day training already shows robust prediction results. We leverage many box observations (spatial observations), rather than long-historical observations (temporal observations) for each box.

IV. Evaluation

In this section, we evaluate *TailGuard* for mitigating performance anomalies for 80 datacenter tenants, using more than 1K boxes and 10K VMs. Our focus is on the following metrics of interest: (1) reduced CPU tail target violations for boxes, (2) reduced CPU anomaly instances, and (3) reduced anomaly durations. Since accurate tail prediction is central to the effectiveness of the proposed strategy, we also compare the tail prediction accuracy presented in Section III with the time series prediction based on neural networks that have been shown very effective on the same trace [9]. In the following, we first sketch the simulator design and present the effectiveness of the proposed methodology in reducing (continuous) AIs. We then highlight how two parts of *TailGuard*, i.e., the tail prediction scheme and VM cloning, outperform alternative approaches, i.e., time series prediction and VM migration.

A. Experimental Set Up

Simulator: We develop a trace-driven simulator that emulates the system dynamics. The input data of the simulator are CPU and RAM usages from VMs and boxes of 80 selected tenants. The simulator computes the box CPU/RAM usages as the sum of the hosted VM CPU/RAM usage plus the original box-only usage. When cloning a VM, its CPU usage is predicted and apportioned as described in Section III. For RAM usage of each cloned VM, we use directly the VM value from the trace as we assume that RAM is replicated. For all statistics related to RAM usage we use the last value from historical data as prediction since the traces show that RAM usage for both boxes and VMs is stable across time.

Targets: Here, we consider the target usage values for box CPU and RAM for triggering tickets for anomaly instances at 60% and 80%, respectively. We experiment different tail usages, such that different tolerances of CPU anomaly instances are evaluated. This is customized to different system requirements. The specific CPU tail usages evaluated here are the 90^{th}, 95^{th}, and 98^{th} percentiles. In the following, we focus on presenting the reduction of CPU tail target violation per tenant, as well as the reduction in the number of CPU AIs.

B. Big Picture

We first present an overview how *TailGuard* reduces tail target violations for the box CPU of the 80 tenants, see Figure 12. The usage tail considered here is 95%ile. Each point in Figure 12 represents a tenant, whose reduction of tail target violations is shown on the y-axis, while the average CPU usage is depicted on the x-axis. The number of boxes of each tenant is represented by the size of the bubble. A lower value on the x-axis shows higher spare resource availability. The figure clearly shows that for bigger tenants and for tenants that have higher spare capacity, the reduction is significantly higher due to the higher degrees of freedom in VM redistributing.

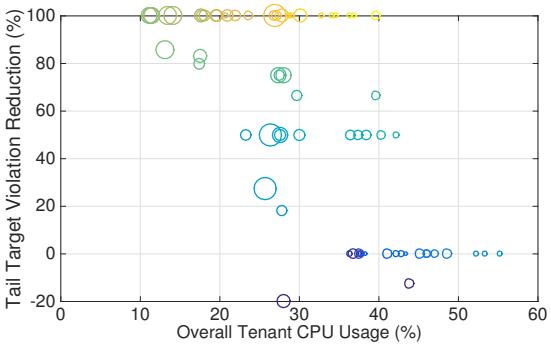
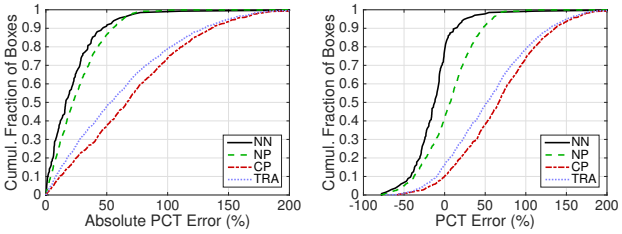


Fig. 12: Reduction of CPU tail target violations for all tested tenants: the tail is set as 95%ile.



(a) Absolute PCT Error (%) (b) PCT Error (%)

Fig. 13: CDF of CPU tail prediction errors using different methods: the tail is set as 95%ile.

C. Effectiveness of Tenant-Resource-Aware Tail Prediction

Here, we present how the proposed *TailGuard* can accurately capture the box CPU tail dynamics and effectively reduce the tail target violation when driving the VM cloning. We compare three variations for tail estimation with a neural network based time series prediction (NN) [9] that accurately forecasts the entire trajectories of usages in the trace. In order to build NN prediction models, we first need to use historical data to train the model. We use the trace data that correspond to the past three days, which is sufficiently long to capture the time dependency within the series. We use the tail estimation scheme proposed in Section III, i.e., $L = \overline{T} + \alpha S_t$ with different values of α:

- $\alpha = 0$, across all tenants, neutral prediction (NP), i.e., the mean of the tail distribution is used as the predicted tail;
- $\alpha = 2$, across all tenants, conservative prediction (CP), i.e., the predicted tail is the mean plus two standard deviations which corresponds to the 97.5^{th} percentile of the Normal distribution ;
- α_i, for tenant i, the proposed tenant-resource-aware tail prediction (TRA).

Prediction Accuracy: Figure 13(a) and (b) summarize the CDF of absolute and raw percentage of prediction errors for the box CPU tail for the three tested days. The two key findings are: (i) the proposed tail estimation scheme (of $\alpha = 0$) is almost as accurate as the time-consuming NN approach, and (ii) the proposed TRA overestimates the CPU tail, but is still less conservative than CP.

The lowest prediction errors are achieved by NN with average value of around 20%, but neural networks are computationally expensive and require long historical data for training. Given shorter training data and much lower computational complexity, NP can achieve very similar absolute prediction errors as NN. When looking to the distribution of raw errors, NP tends to overestimate the box CPU tails as more than 60% of errors are positive. In contrast, NN tends to underestimate the box CPU tail, since more than 70% of errors are negative. TRA and CP have significant higher absolute and raw errors, due to their conservativeness in estimation. In terms of raw errors, only 15% of prediction errors from TRA are negative. As the ultimate objective is to enable efficient resource management, conservative prediction tends to resource over-provisioning, which is more desirable than under-provisioning, which leads to high risk for performance anomalies.

Reduction of Tail Target Violation: To see the impact of the tail prediction schemes and to evaluate the proposed TRA, we use the three tail prediction schemes to drive VM cloning. Figure 14 summarizes the reduction of CPU tail target violations for each tenant, for different considered tail percentiles and prediction schemes. Each box in Figure 14 presents the distribution of tenants' reduction of CPU tail target violation, for different tail percentiles. Each rectangular box contains three horizontal lines that correspond to the 25^{th}, 50^{th} and 75^{th} percentiles, the circles show the mean. The whiskers show the extreme values in the distribution. We can see that the proposed TRA achieves the highest average reduction per tenant (shown by higher positions of circles), for all three tail percentiles. Specifically, the average reduction of CPU tail target violation under the TRA prediction scheme is around 50%, whereas CP and NP can only achieve average tail reduction per tenant at around $30\% - 40\%$. For all three tail prediction methods, the increase in number of RAM AIs is negligible, with all less than 3%. Another finding worth mentioning is that, when increasing the tail percentiles, i.e., from 90%ile to 98%ile, the reduction drops slightly for all prediction schemes. This is because a more stringent performance requirement is applied, allowing only very few AIs, and the potential of reducing violation by redistributing the loads across boxes becomes lower. This also leads to the explanation why CP outperforms NP in case of 90%ile, but NP results into better reduction in the case of 95%ile and 98%ile.

To highlight the impact of different prediction schemes on mitigating tail target violations, we zoom into the performance of two tenants. One of the tenants has a lower CPU utilization, meaning high resource availability, whereas the other tenant has a higher CPU utilization. We list their reduction of CPU tail target violation in Table I. CP is able to remove all tail target violations for the tenant with a higher resource availability but performs poorly for the second tenant. NP has the opposite performance for these two tenants, arguing for the need of a prediction scheme that can self-adapt to the resource availability. This is what TRA consistently does: tail prediction enables VM cloning to achieve the highest amount of reduction for both tenants, as it uses different α_i values for box tail predictions, based on the resource availability of different tenants.

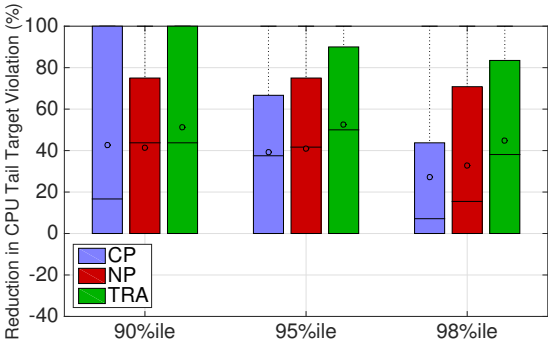


Fig. 14: Comprison of CPU tail target violation reduction: CP *v.s.* NP *v.s.* TRA.

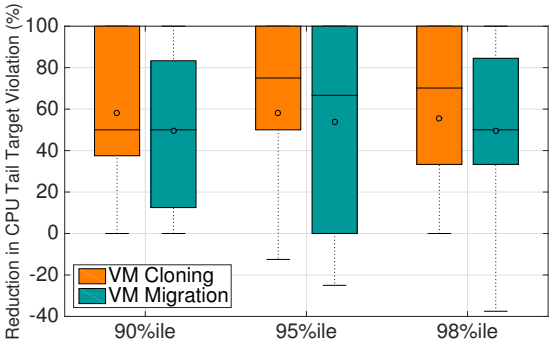


Fig. 15: Comparison of CPU tail target violation reduction: VM cloning *v.s.* VM migration.

TABLE I: CPU tail target violation reduction: a case study comparing CP, NP and TRA with the tail set as 95%ile.

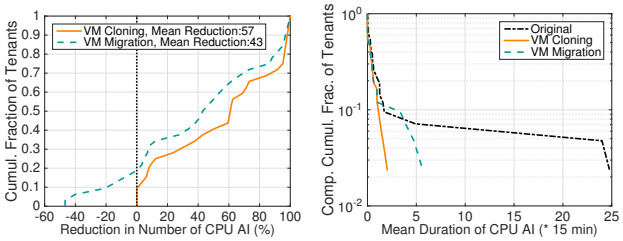
	Tenant with *higher* resource availability (mean CPU usage = 12%)	**Tenant with *lower* resource availability (mean CPU usage = 42%)**
CP	100	16.7
NP	75.0	50
TRA	100	50

D. Effectiveness of VM Cloning

Here, we highlight how *TailGuard* can reduce CPU tail target violation and AIs by VM clonning strategy, with negligible impact on the RAM violations. We compare the VM cloning with an alternative of VM migration only strategy, which migrates VMs from boxes to boxes, without replicating the memory. For a fair comparison, we provide the box tail prediction obtained from the TRA prediction scheme to both strategies. As both strategies aim to redistribute the box workloads, we use the same criteria to determine how many VMs to be moved out of certain boxes and how many additional VMs to be allocated to certain boxes, see Section III-B.

CPU Tail Target Violation: Figure 15 summarizes the distribution of the reduction of CPU tail target violations per tenant, when applying VM cloning and migration on different tail percentiles. One can see that VM cloning is able to achieve a slightly higher reduction of tail target violations for CPU by roughly 10%, shown by the difference of mean values. Moreover, with VM cloning, the range of reduction of tail target violations for CPU across all tenants is much smaller, supported by the shorter box. In addition, cloning guarantees tail reduction while migration does not necessarily do so, see the whiskers in the respective boxes. Indeed, when the tail percentile is set to 98%ile, namely allowing only 2 CPU AIs for the entire day, VM migration can result into undesirable scenarios, i.e., certain tenants may experience a tremendous increment of CPU tail target violation as shown by the negative values of reduction, while VM cloning can ensure a more consistent reduction in CPU tail target violations, and prevent the aggravation of CPU tail target violation by online usage monitoring and workload management.

Another advantage worth mentioning is that VM cloning ensures RAM performance, compared to VM migration. Our simulation results show that VM migration indeed results into significant increment of number of RAM AIs, with roughly



(a) Reduction in CPU AI (%) (b) Duration of CPU AI

Fig. 16: Comparison of CPU AI for each tenant between VM cloning and VM migration: the tail is set as 95%ile.

50% increment for each tenant on average. On the contrary, the proposed VM cloning strategy is able to bound the increment of RAM usage violation within 3%. This is thanks to the online usage monitoring and workload management, which is able to terminate clones upon detecting any RAM violation.

Occurrences and Duration of CPU AIs: Now, we present the difference between VM cloning and VM migration from the perspective of number of CPU AIs and their duration. Figure 16(a) and (b) present the CDF of average reduction in number of CPU AIs and the complementary CDF (CCDF) of average duration of CPU AIs per tenant, respectively. We could see that VM cloning can achieve a mean reduction of 57% in number of CPU AIs per tenant, while VM migration only reduces CPU AIs per tenant by around 43%. Moreover, similar to the case of reduction in CPU tail target violation, VM cloning guarantees almost no increment in the number of CPU AIs. However, there are 20% of tenants with an increased number of CPU AIs after VM migration. In terms of duration of CPU AIs summarized as CCDF in Figure 16(b), VM cloning can bound the duration of continuous CPU AIs below 2 time windows, whereas VM migration still results in 6 time windows of continuous CPU AIs in the worst case. We note that in the original testing trace, there are some tenants suffering from *long-term* continuous CPU AIs, as long as 25 time windows, i.e., CPU usage exceeds the target value of 60% for more than 6 hours.

V. Related Work

Performance anomaly detection has been the subject of many workload characterization studies in recent years to improve system reliability by better understanding the reasons behind system failures and/or bugs [2], [10], [11], [12]. These studies focus on statistical analysis of trace data ranging from production data centers to large-scale storage systems. Online performance anomaly detection methods have been proposed [13], [14], [15] in order to detect anomalies in the upcoming future using online behavior analysis via either statistical methods or machine learning techniques. An online performance anomaly prevention method for cloud computing is proposed in [16], which integrates online anomaly prediction, learning-based cause inference, with predictive prevention actuation. The spatial-temporal dependencies in usage time-series are explored in [4] for VM resizing, with an objective to reduce all VM performance tickets in data centers, i.e., all usages need to be below the target. In contrast, *TailGuard* focuses on avoiding box tail violations and makes a conscious tradeoff between resource requirement and anomaly avoidance, based only on the last values of usages collected from a vast number of boxes.

VM migration on a cluster of physical servers has long been proposed to mitigate performance anomalies via load balancing [17]. The main questions are determining *when* and *how* to migrate VMs, aiming to optimize resource usage (e.g., network bandwidth [18]) and performance measures, including non-traditional ones such as energy consumption [19]. Dynamic VM migration is an attractive solution to constantly fluctuating user workloads in data center. Several migration algorithms have been proposed including *best-effort* online VM migration [20], [21] as well as several performance models of online VM migration [22]. Yet, migration overhead cannot be disregarded [23], especially in systems where service availability and responsiveness are under Service Level Agreements (SLAs). Different from VM migration, in this paper, *TailGuard* makes use of VM cloning to handle excess load, opportunistic job placement, and parallel processing [24]. VM cloning enables VM management that is more *flexible* and of *finer-granularity* control than VM migration.

VI. Conclusion

In this paper, we develop *TailGuard*, a tail-driven anomaly avoidance policy, which can effectively reduce continuous AIs for box CPU and avoid over-reacting to the spontaneous single AI. The design of *TailGuard* is based on a large-scale characterization study of production data centers. *TailGuard* is composed of two novel steps: a light-weight tail usage predictor simply leveraging the power of the vast amount of usage data across boxes, and a VM cloning strategy combining the advantage of migrating VMs and balancing the loads between clones. Moreover, we incorporate the concept of resource availability per tenant into the tail prediction, such that *TailGuard* is able to adjust the conservativeness of prediction based on the flexibility of redistributing CPU loads. The extensive evaluation results on 80 tenants over 1K boxes over 3 days show that *TailGuard* is able to accurately predict the box tail usage, with an accuracy comparable to the neural network based time series prediction. More importantly, while *TailGuard* purposely allows a small fraction of AIs to avoid excessive resource provisioning, *TailGuard* can still achieve a significant reduction in CPU AIs and drastically reduce the anomaly duration by 10 times.

Acknowledgment

The research presented in this paper has been supported by NSF grant CCF-1218758, EU commission FP7 GENiC project (Grant Agreement No.608826), and the Swiss National Science Foundation (project 200021_141002, 200020_169089, and 407540_167266).

References

[1] I. Giurgiu, J. Bogojeska *et al.*, "Analysis of Labor Efforts and their Impact Factors to Solve Server Incidents in Datacenters," in *CCGrid*, 2014.

[2] R. Birke, I. Giurgiu *et al.*, "Failure analysis of virtual and physical machines: patterns, causes and characteristics," in *DSN*, 2014.

[3] I. Giurgiu, A. Almasi *et al.*, "Do you know how to configure your enterprise relational database to reduce incidents?" in *IM*, 2015.

[4] J. Xue, R. Birke *et al.*, "Managing data center tickets: Prediction and active sizing," in *DSN*, 2016.

[5] R. Birke, A. Podzimek *et al.*, "State-of-the-practice in data center virtualization: Toward a better understanding of VM usage," in *DSN*, 2013.

[6] R. Peck, C. Olsen *et al.*, *Introduction to statistics and data analysis*. Cengage Learning, 2015.

[7] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, 1951.

[8] P. Membrey, E. Plugge *et al.*, *Practical Load Balancing*. Apress, 2012.

[9] J. Xue, F. Yan *et al.*, "Practise: Robust prediction of data center time series," in *CNSM*, 2015.

[10] A. Rosa, L. Y. Chen *et al.*, "Understanding the dark side of big data clusters: an analysis beyond failures," in *DSN*, 2015.

[11] E. Chuah, A. Jhumka *et al.*, "Linking resource usage anomalies with system failures from cluster log data," in *SRDS*, 2013.

[12] B. Schroeder, R. Lagisetty *et al.*, "Flash reliability in production: The expected and the unexpected," in *FAST*, 2016.

[13] J. A. Cid-Fuentes, C. Szabo *et al.*, "Online behavior identification in distributed systems," in *SRDS*, 2015.

[14] L. Cherkasova, K. Ozonat *et al.*, "Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change," in *DSN*, 2008.

[15] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures," in *SRDS*, 2013.

[16] Y. Tan, H. Nguyen *et al.*, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *ICDCS*, 2012.

[17] H. W. Choi, H. Kwak *et al.*, "Autonomous learning for efficient resource utilization of dynamic vm migration," in *ICS*, 2008.

[18] A. Surie, H. A. Lagar-Cavilla *et al.*, "Low-bandwidth vm migration via opportunistic replay," in *HotMobile*, 2008.

[19] C. Ghribi, M. Hadji *et al.*, "Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms," in *CCGrid*, 2013.

[20] H. Liu, H. Jin *et al.*, "Live migration of virtual machine based on full system trace and replay," in *HPDC*, 2009.

[21] J. W. Jiang, T. Lan *et al.*, "Joint vm placement and routing for data center traffic engineering," in *INFOCOM*, 2012.

[22] H. Liu, H. Jin *et al.*, "Performance and energy modeling for live migration of virtual machines," *Cluster computing*, vol. 16, no. 2, 2013.

[23] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *MMSys*, 2010.

[24] H. A. Lagar-Cavilla, J. A. Whitney *et al.*, "Snowflock: rapid virtual machine cloning for cloud computing," in *Eurosys*, 2009.