Managing Data Center Tickets: Prediction and Active Sizing

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*—Performance ticket handling is an expensive operation in highly virtualized cloud data centers where physical boxes host multiple virtual machines (VMs). A large body of tickets arise from the resource usage warnings, e.g., CPU and RAM usages that exceed predefined thresholds. The transient nature of CPU and RAM usage as well as their strong correlation across time among co-located VMs drastically increase the complexity in ticket management. Based on a large resource usage data collected from production data centers, amount to 6K physical machines and more than 80K VMs, we first discover patterns of spatial dependency among co-located virtual resources. Leveraging our key findings, we develop an Active Ticket Managing (ATM) system that consists of (i) a novel time series prediction methodology and (ii) a proactive VM resizing policy for CPU and RAM resources for co-located VMs on a physical box that aims to drastically reduce usage tickets. ATM exploits the spatial dependency across multiple resources of co-located VMs for usage prediction and proactive VM resizing. Evaluation results on traces of 6K physical boxes and a prototype of a MediaWiki system show that ATM is able to achieve excellent prediction accuracy of a large number of VM time series and significant usage ticket reduction, i.e., up to 60%, at low computational overhead.**

I. Introduction

Performance ticketing systems provide the means to data centers to interactively improve user experience, maintain performance at tails, and guarantee smooth system operation. Typically, system monitoring and users issue tickets when encountering an array of performance violations, e.g., unresponsive service, high resource usage due to transient load dynamics, or persistent insufficient provisioning. Ticket resolution is unfortunately very expensive [1], [2] as a significant amount of manual labor is required for root-cause analysis and to remedy the detected problem [3]. Prior work has shown that there is strong correlation of ticket issuing with resource usage exceeding certain predefined thresholds [4]. In today's data centers, with physical resources being aggressively multiplexed across multiple virtual machines (VMs), the likelihood of issuing performance tickets due to physical or virtual machines crossing predefined usage thresholds dramatically increases.

Past work has established that resource usage at data centers exhibits strong temporal patterns [5], [6]. Beyond temporal dependencies that are established by usage time series [7], it is common for co-located VMs to simultaneously compete for the limited physical resources, essentially exhibiting strong *spatial* dependency. We illustrate a motivating example in Figure 1 depicting the CPU usage time series[1] of 4 VMs co-located within the same physical box, where performance tickets are issued automatically when a VM utilization exceeds a threshold of 60%. One can easily see the spatial dependency of VMs 1, 3, and 4, i.e., time usages move up and down synchronously, and their respective tickets are triggered together, at around the 19:00 hour mark. These time series come from a data center production system and are quite representative of typical patterns in such systems. The temporal and spatial dependencies among VMs not only increase the number of tickets but also the difficulty in identifying their root cause and the corresponding resolution.

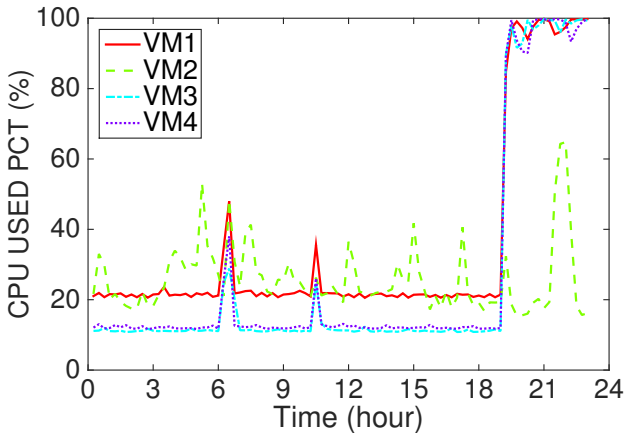


Fig. 1: An illustration of spatial dependency across usage time series for 4 VMs co-located on a box.

The focus of this paper is to develop a methodology to increase the data center dependability by using a *proactive* approach: reduce the number of tickets by predicting when they will occur in the future and by employing dynamic virtual machine resizing to adjust resource usage to avoid the triggering of future tickets. To this end, we first do a detailed, post-hoc workload characterization study of usage time series in production data centers of a major vendor which correspond to 80K VMs hosted on 6K physical servers. We develop an Active Ticket Managing (ATM) system that predicts future VM resource usage and proactively resizes the virtual resources of the resident VMs. The research challenges are numerous and outlined as follows.

Effective usage prediction is prerequisite to the development of any management policy. Indeed, in our past work

[1]We interchangeably use the terms time series and series.

we have shown that neural networks can be effectively employed for prediction [7], but their effective usage remains prohibitively expensive in practical situations as it suffers by its high training cost. In practice, in a large-scaled data center, with more than tens of thousands of physical boxes and hundreds of thousands of VMs, it is infeasible to rely on neural networks to predict future resource usage. We solve this first problem by developing a prediction methodology that discovers spatial dependencies across usage series and exploits them to develop an agile methodology for prediction. To this end, we introduce the concept of *signature VM series*, a subset of usage series that are representative of all other usage series. We are able to predict usage series not in the signatures set and the usage violation tickets of co-located VMs, via a linear combination of signature VM series, which provides a time series prediction model with as low as only 26% of the original time series. Second, based on predicted resource usage, we define a multi-choice knapsack problem and develop a greedy algorithm to dynamically adjust virtual resource allocation across co-located VMs. ATM is evaluated on production traces of 80K VMs and a small test-bed deployment on a cluster that runs MediaWiki [8], the open source platform for Wikipedia. Our extensive evaluation results show that ATM has remarkably high accuracy in prediction, i.e., reaching prediction errors as low as 20% and significant ticket reductions, i.e., up to $60\% - 70\%$ less tickets while using only 26% of the original time series. The contributions of this paper are as follows:

1) We do post-hoc characterization of usage ticket issuing in a large data center setting. We focus on discovering the distribution of usage tickets and spatial patterns of resources usages across co-located VMs. We find that usage tickets are mostly contributed by a small set of VMs, and that VMs show significant cross correlation among their CPU and RAM usage series.

2) Motivated by the strong spatial patterns across resources and co-located VMs, we argue that a small number of signature usage time series can be used as predictors to represent well the entire set of resource usage time series. This prediction methodology is the basis of ATM.

3) We develop a VM resizing policy to reduce usage tickets by setting the upper limits of CPU and RAM allocations when several VMs are co-located, a problem which is shown to be NP-hard. We rigorously formulate the ticket minimization problem subject to the physical capacity constraints. We propose a greedy algorithm to solve it and compare its performance to the max-min fairness algorithm.

The outline of this work is as follows. Section II provides a characterization study on the usage tickets as well as the spatial patterns among usage series of co-located VMs. We propose spatial-temporal prediction methods for demand series in Section III. In Section IV, we formulate the ticket minimization problem and demonstrate a greedy resizing algorithm to reduce usage tickets. An extensive evaluation of ATM on both production traces and a Wikipedia cluster is discussed in Section V. Section VI presents related work, followed by the summary and conclusions in Section VII.

II. Statistics and Observations

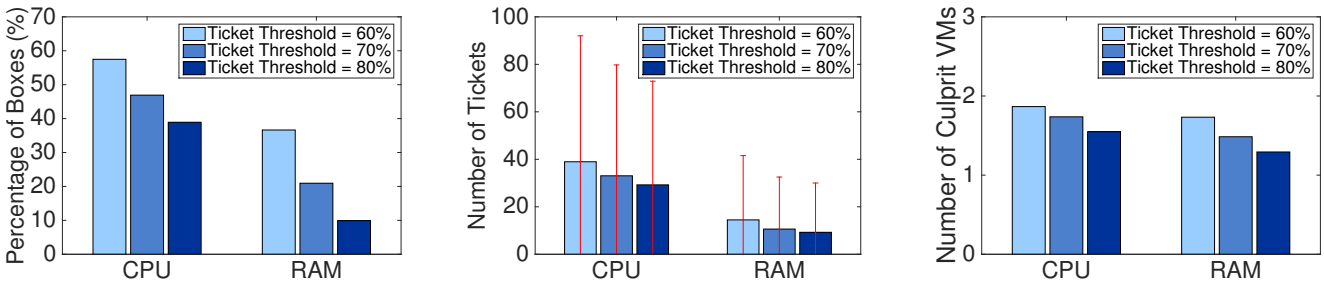
The motivation for the design of ATM is the urge to reduce usage tickets that are typically issued when VM resource utilizations exceed certain thresholds. The trace that we consider here comes from IBM production data centers serving various industries, including banking, pharmaceutical, IT, consulting, and retail, and using various UNIX-like operating systems, e.g., AIX, HP-UX, Linux, and Solaris. The majority of VMs in the trace are VMware VMs. The trace contains CPU and RAM capacity but also utilization data taken 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]. In addition, both VMs and boxes are very heterogeneous in terms of resource configuration.

In the following, we first show the distribution of usage tickets under different ticket thresholds, followed by a more detailed analysis on the spatial patterns of usage series of co-located VMs. We aim to uncover how usage tickets are distributed across resources and most importantly how usage patterns trigger usage tickets. We anticipate that the design principles of the proposed ATM system leverages this characterization analysis.

A. Usage Tickets

Usage tickets are generated when utilization values exceed target thresholds. Naturally, lower thresholds trigger a higher number of usage tickets and increase the cost of resolution, whereas higher thresholds result into fewer tickets but at a higher risk of performance degradation. To quantify the effect of different thresholds, we consider three threshold levels, namely 60%, 70%, and 80%. Such values are commonly adopted in production systems [9]. Figure 2 illustrates quantitative information on the issued tickets for the CPU and RAM usage series of April 3, 2015. We focus on the following: how many boxes have tickets and how these tickets are distributed across co-located VMs and their resources.

Figure 2(a) plots the percentage of boxes that have at least one VM usage ticket under the different thresholds. Even with the highest ticket threshold of 80%, almost 40% of boxes obtain at least one ticket due to CPU violation and 10% due to RAM violation, these percentages increase to 57% and 38%, respectively, when the threshold is 60%. Overall, the percentage of boxes having CPU tickets is higher than RAM tickets, independently of the threshold. This can be explained by the fact that RAM tends to be over-provisioned for performance reasons. Figure 2(b) illustrates the mean and standard deviation of the number of tickets per box for CPU and RAM. The average number of CPU(RAM) usage tickets per box are 39(15), 33(11), 29(9), for the three thresholds of 60%, 70%, and 80%, respectively, showing a relatively minor decreasing trend. The next natural question is whether tickets are evenly distributed across all co-located VMs. To this end, we compute the number of VMs that accounts for the majority of tickets, where the majority is defined to 80% of usage tickets per box (this is an ad-hoc value). Figure 2(c) shows that on average one to two VMs per box cause the majority of tickets irrespective of the three threshold values. A further



(a) Percentage of boxes having tickets (b) Distribution of tickets (c) Number of culprit VMs

Fig. 2: Characterization of usage tickets for CPU and RAM of VMs per box.

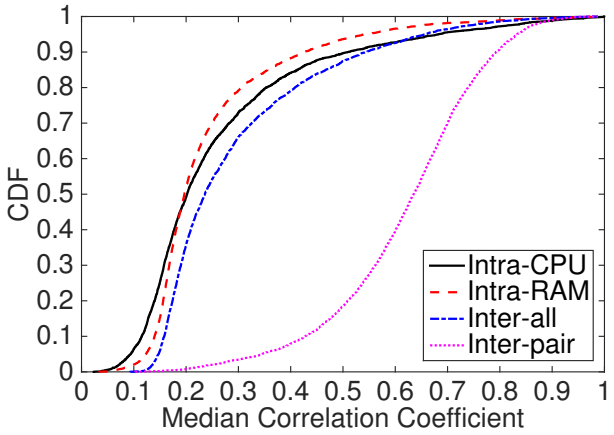


Fig. 3: Cumulative distribution of correlation of intra-CPU, intra-RAM, inter-CPU/RAM.

interesting observation is that since the culprit VMs are few, if we increase the capacity allocation of the culprit VMs by removing resources from other co-located VMs, then tickets may reduce. On the contrary, if tickets are evenly distributed, resizing does not help.

B. Do Spatial Dependencies Exist?

To better understand the spatial patterns of usage tickets, we estimate the magnitude of spatial dependency by computing the Pearson's correlation coefficients [10] over each pair of CPU and RAM usage series of co-located VMs. For each box and co-located VMs, we compute four types of correlation coefficients ρ: (i) between any pair of CPU usage series (intra-CPU), (ii) between any pair of RAM usage series (intra-RAM), (iii) between any pair of CPU and RAM usage series (inter-all), and (iv) between CPU and RAM usage series from the same VM (inter-pair). The first two correlation metrics measure the relationship among specific resources, i.e. CPU and RAM, time series ("intra" resource measures), the latter two focus on the relationship between CPU-RAM pairs ("inter" measures). For each box, we compute the median value of the above measures and present the cumulative distribution functions (CDFs) across all the boxes in Figure 3.

One can immediately see from the shapes of the CDFs that intra-RAM ρ is higher than intra-CPU, followed by inter-resources measured from any pair of VM or the same VM. This implies that inter-resource dependency is higher than the intra-resource one. Indeed, the mean values for intra-CPU, intra-RAM, inter-CPU/RAM from any pair, inter-CPU/RAM from the same VM are 0.26, 0.24, 0.30, and 0.62 respectively. The CDFs give a clear message: the CPU-RAM pairs across co-located VMs are correlated, this is a fact that we take advantage of when we attempt to use clustering to reduce the cost of prediction.

III. Spatial-Temporal Prediction Models

We first elaborate on the challenges for concurrent prediction for a large number of time series representing multiple resource usages from co-located VMs at production data centers. The immediate obstacles of prediction given a large number of demand series are accuracy, training overhead, and model scalability. Typically, temporal models [10], such as ARIMA are not able to capture well bursty behaviors. More sophisticated temporal models such as neural networks, capture irregular patterns better but at much higher computational overheads. Given such restrictions, it is important to come up with efficient and accurate prediction models that also scale well.

We propose a new prediction methodology that combines both temporal and spatial models to predict on each box the resource demand time series[2] $\boldsymbol{D_i}$ $(\forall i \in [1, M \times N])$ where M is the number of co-located VMs and N is the number of different resources taken into consideration. We introduce the concept of *signature series:* a minimum number of time series that are predicted via *temporal models*. The rest of the demand series, termed as *dependent series*, are predicted through a linear combination of signature series via *spatial models*. Essentially, we divide the demand series, $\boldsymbol{D_i}$, into two sets: the signature set, denoted by Ω_s, and the dependent set, Ω_d.

The novelty of ATM is to derive novel spatial models for dependent series while applying existing temporal models to predict signature series. Many practical techniques exist in the literature for reducing the overhead of temporal models by extracting and storing features of the time series [7], [11]. We stress that any temporal prediction model can be directly plugged into the ATM framework.

[2]Demand series is the product of usage series and the allocated virtual capacity. Both demand and usage series share the same correlation characteristics. For the purpose of virtual resource resizing, we predict demand series directly.

To derive the spatial models, we want to express all demand series $\boldsymbol{D_k}, k \in \Omega_d$ by a linear combination f_k of the signature series $\boldsymbol{D_j}, j \in \Omega_s$:

$$\boldsymbol{D_k} = f_k(\boldsymbol{D_j}). \tag{1}$$

As every demand series can be either a signature or a dependent series, a brute force solution to find the minimum signature set is to explore all $2^{N \times M}$ combinations of regression models. For boxes hosting an average number of VMs, i.e., M around 10 and expected to grow as servers become more powerful, it is clear that this method is not viable. To address this issue, we devise an efficient searching algorithm that can quickly find signature series without using exhaustive search, by leveraging time series clustering techniques and stepwise regression.

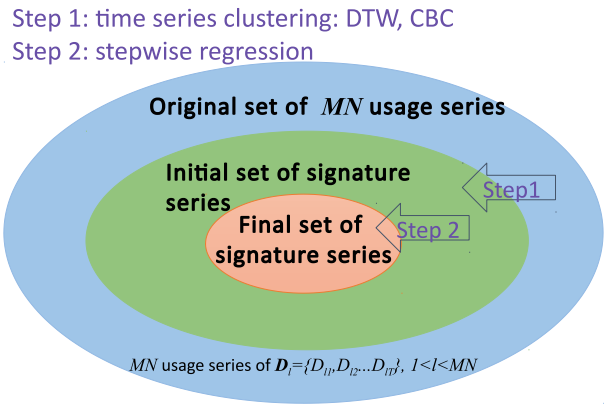


Fig. 4: Overview of searching for signature set.

A. Searching for Signature Demand Series

Key to the discovery of signature series is clustering. We propose a two-step algorithm to identify the signature set Ω_s. *Step 1* defines the initial set of signature series. This is achieved using time series clustering, specifically dynamic time warping (DTW) [12] or correlation based clustering (CBC) that we propose here. *Step 2* defines the final set of signature series by detecting and removing multicollinearity among initial set of signature series using variance inflation factors (VIF) and stepwise regression. The intent of the second step is to fix the pitfall that although signature series appear independent, it is possible that a combination of certain subsets of the initial signature series can well represent the others. For example, a group of series can be separated into three clusters because of their dissimilarity in the distances or the correlation patterns. If however one of the clusters can actually be well expressed as a linear combination of the other two, then this falls under a classical example of multicollinearity. Figure 4 illustrates the steps of signature set search.

Step 1: Time Series Clustering: Dynamic time warping is an effective solution for finding clusters of time series where the distance across the series is short. A potential problem is that DTW falls short in capturing within the cluster series that are of larger distance. Correlation based clustering solves this problem by capturing highly correlated time series that are far enough apart and cannot be captured by DTW. Applying DTW on the exemplary four series shown in Figure 1 illustrates how clustering with DTW only offers a partial solution. DTW detects three clusters: cluster 1 VM1, cluster 2 VM2, and cluster 3 VM3 and VM4. CBC instead puts VM1, VM3, and VM4 within the same cluster. Indeed, the series $\boldsymbol{D_1}$ and $\boldsymbol{D_4}$ of VM1 and VM4 can be well represented as linear models of the series $\boldsymbol{D_3}$ of VM3, e.g., $\boldsymbol{D_1} = a_0 + a\boldsymbol{D_3}$, and $\boldsymbol{D_4} = b_0 + b\boldsymbol{D_3}$, where a_0, a, b_0, and b are scalars. In the remaining of this section we provide details on DTW and CBC.

Dynamic Time Warping Clustering: The high level idea of DTW is to group series that show low *distance dissimilarity*. To obtain the distance dissimilarity between two series $P = \{p_1, p_2, ..., p_i, ..., p_n\}$ and $Q = \{q_1, q_2, ..., q_j, ..., q_m\}$, we first build a matrix that consists of the pair-wise squared distances, i.e., $d(p_i, q_j) = (p_i - q_j)^2$, between each pair of elements p_i and q_j in the two series. The distance dissimilarity $\lambda(n, m)$ of the two series is given by the wrapping path through the matrix that minimizes the total cumulative distance [12] and can be recursively computed as follows:

$$\lambda(i,j) = d(p_i, q_j) + min\{\lambda(i-1, j-1), \lambda(i-1, j), \lambda(i, j-1)\}. \tag{2}$$

Next, we apply hierarchical clustering [13] for any given number of clusters, ranging from 2 to $(M \times N)/2$ since we aim to reduce the original set to at least its half. We determine the optimal number of clusters, based on the average silhouette value [14] of all time series within each cluster. For each series i, its silhouette value $s(i)$ is defined as

$$s(i) = \frac{b(i) - a(i)}{max\{b(i), a(i)\}} \tag{3}$$

where $a(i)$ is the average distance dissimilarity between series i to all the other series within the same cluster using DTW, and $b(i)$ is the lowest average distance dissimilarity between series i to the all the series in a different cluster. The higher the silhouette value, the better the series lies within its cluster. For each number of clusters, we average the silhouette values of all the series as the representative silhouette value. The optimal number of clusters is the one with the maximal silhouette value. As last step and beyond conventional DTW, we identify each signature series as the series with the lowest average dissimilarity in each cluster.

Correlation-based Clustering: CBC focuses on grouping series showing high correlation. For each box, we first compute the pairwise correlation coefficients, denoted as ρ, for all pairs of the $M \times N$ series. For a demand series $\boldsymbol{D_i}$, there are $(M \times N - 1)$ pairs $\rho_{i,l}, \forall l \neq i$. To form the clusters, we rank each series $\boldsymbol{D_i}, i \in [1, M \times N]$ first by the total number of $\rho_{i,l}$ above a threshold ρ_{Th}, and second by the mean value of the $\rho_{i,l}$ above the threshold. In the following we set $\rho_{Th} = 0.7$, a common threshold value used to determine strong correlation between two series, which suggests a potential for linear fitting [15]. After the series have been ranked, we select the topmost one and remove it together with all the series being correlated with it with a correlation coefficient higher than the threshold. These series are now considered within a new cluster with the top ranked series being the signature series. This procedure continues by selecting the next topmost series still in the ranked list and ends when the ranked list becomes empty.

Step 2: Stepwise Regression: To further reduce the number of signature series, we calculate the variance inflation factor – a metric that can detect multicollinearity in regression. For each series in the signature set, we regress it on the rest of signature series and obtain its VIF value [15]. The rule of practice is that a VIF greater than 4 indicates a dependency with the other series in the initial set. After detecting the risk of multicollinearity, i.e., at least one series has a VIF greater than 4, we perform standard stepwise regression to remove the series that can be represented as linear combinations of the other signature series.

B. Prediction Models

To predict all $M \times N$ demand series, we first predict the signature series $\boldsymbol{D_i}\,(i \in \Omega_s)$, using neural network models and their historical data [7]. To predict all dependent series, we regress each dependent series on the set of signature series, obtaining coefficients using ordinary least square estimates. We stress that the signature series predictions are not tied to the any specific model rather any suitable prediction model can be easily plugged into our ATM framework.

In summary, we first leverage historical data to develop spatial models to define dependent series and their respective signatures. Later, we use temporal models to predict the signature series and inexpensive linear transformation models to predict the dependent series.

C. Results on Spatial Models

Prior to moving to the proposed VM resizing policy, we present evaluation results of the proposed spatial models across the demand series of the trace data (6K boxes and 80K VMs) presented in Section II. Our evaluation focuses on: (i) the difference between DTW and CBC clustering, (ii) the effectiveness of clustering and stepwise regression, and (iii) inter- v.s. intra-resource models, i.e., if it is necessary to treat different resource series, e.g., CPU and RAM, separately. Since the purpose of spatial models is to use a minimum subset of original series to accurately represent the data center, the metrics of interest are: (i) the percent of signature series out of the total demand series and (ii) the prediction error. In this section we only focus on the effectiveness of the spatial models, i.e., how close the dependent series are from the actual time series counterparts. The overall prediction accuracy of combining spatial models with temporal models is presented in Section V.

1) Difference between DTW and CBC: Figure 5 compares the distribution of the number of clusters resulting from DTW and CBC and highlights the type of each signature series, i.e., CPU or RAM. For DTW, roughly 70% of boxes have only 2 to 3 clusters, and the rest have 4 to 31 clusters. In contrast, CBC is less aggressive resulting in a higher number of clusters and, consequently, a higher number of signature series. This indicates a higher overhead to develop their temporal models. Moreover, in terms of signature series types, under DTW, one can see that both CPU and RAM roughly account for 50% of the signature series. This is consistent across all DTW bars in Figure 5. Instead, with CBC, most signature series are CPU series.

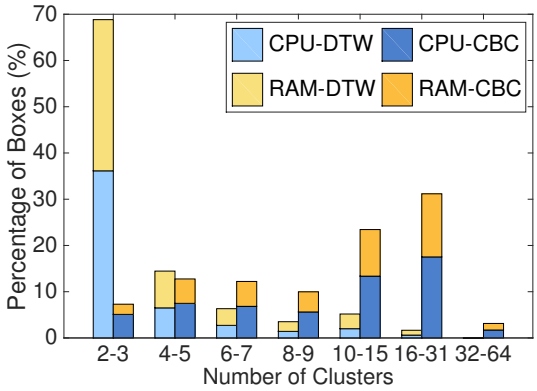
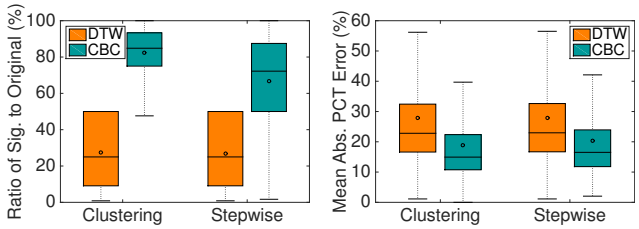


Fig. 5: Comparison of clustering results using DTW and CBC.



(a) Signature set reduction (b) Prediction error

Fig. 6: Comparison of the two steps: effectiveness of clustering and stepwise regression.

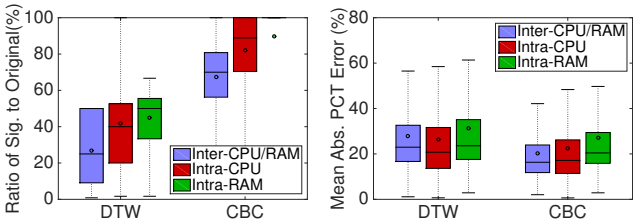
2) Effectiveness of the Two-Step Approach: To better illustrate the benefits of time series clustering (DTW or CBC) and stepwise regression, we compare the signature set reduction and prediction accuracy of each step in Figure 6. Each box represents the 25^{th}, 50^{th} (mid line), and 75^{th} percentiles, whereas the dot marks the mean and the whiskers the most extreme data points.

Figure 6(a) shows the percent of signature series out of the total number of series for each of the 6K physical boxes. Since DTW is quite aggressive in reducing the number of time series, there is almost no further reduction after applying stepwise regression. Both steps reduce the entire set to 26%. After CBC, the set is reduced to 82%, however stepwise regression brings further down the number to 66%.

Considering prediction accuracy, as shown in Figure 6(b), both DTW and CBC experience minor losses. The average absolute percentage error (APE)[3] from DTW is about 28%, while the average APE for CBC is only around 20%. Since stepwise regression almost does not affect the signature set of DTW, one expects no obvious decrease in prediction accuracy. This is indeed shown in the graph. Surprisingly, the same holds true with CBC where stepwise regression reduces CBC's accuracy only by 1%. These results confirm the effectiveness of stepwise regression in reducing the signature set without degrading in prediction accuracy.

3) Inter- v.s. Intra-Resource Models: We compare the effectiveness of the proposed inter-resource models, i.e., com-

[3]APE is defined as $APE = \frac{|Actual - Fitting|}{Actual}$



(a) Signature set reduction (b) Prediction error

Fig. 7: Comparison of inter- and intra-resource models.

bining CPU and RAM as predictors, against the intra-resource models, in which CPU and RAM are treated separately. In Figure 7, we summarize the prediction errors and reduction in the original demand series. Inter-resource models can not only reach a lower prediction error but also a higher reduction in the number of demand series, than intra-CPU and intra-RAM models. We present the results in box plots. In terms of average APEs of prediction for CBC(DTW), the inter model is around 20%(28%), whereas the intra-CPU and intra-RAM are 21%(26%) and 23%(31%). Again, these results are in good agreement with our observation in Section II that inter resource correlation is higher than intra-CPU and intra-RAM correlations. In terms of the average number series in the signature set for CBC(DTW), the inter model uses roughly 66%(26%) of the original series, while intra-CPU and intra-RAM can use up to 81%(41%), 90%(45%) compared with the original set. Overall, the inter model can greatly benefit from the correlation across co-located resources.

IV. Virtual Resource Resizing

Being able to accurately predict future usage enables the very first step to actively manage usage-related tickets. Having future usage knowledge, it is possible to develop a virtual resource resizing policy that can effectively reduce the number of usage tickets. The monitoring systems in modern data centers track the resource usages at discrete windows, e.g., 15 minutes, termed as the ticketing window, and compare them with ticket thresholds to determine whether a ticket needs to be issued or not. To avoid incurring overreaction to transient loads, we set the resizing window to be greater than the ticketing window. For the data centers considered here, ticket resolution occurs within a day of the ticket being issued, so setting the resizing window to one day is a reasonable assumption. This implies that the prediction horizon of the demand series needs to be also one day. Note that past work has shown that the accuracy of prediction decreases as the prediction horizon increases [7], so setting the prediction window to such a high value makes ATM more conservative than it can actually be. During each resizing window, ATM devises and actuates the virtual resource allocation of co-located VMs on boxes. The objective is to find optimal sizes for co-located VMs to achieve the lowest number of tickets, subject to various resource constraints at boxes. The resources considered are: virtual CPU measured in GHz and virtual RAM measured in GB.

There exist a large body of virtual resource allocation studies aiming to satisfy various performance targets, e.g., user response time, system utilization, and fairness. Max-min fairness [16], [17] is one of the most applied allocation policies that tries to guarantee the performance of small VMs, given the assumption of known demands. Our resizing problem can be viewed similarly but with the objective to minimize the occurrences of target utilization threshold violations. We develop a resizing algorithm based on a rigorous optimization formulation, which is later transformed into a multi-choice knapsack problem (MCKP) with tunable discretization parameters. The introduction of such discretization parameters enables us to reduce the complexity and increase the safety margin in resource allocation. In contrast to spatial-temporal prediction models, the resizing algorithm treats CPU and RAM separately due to separate constraints on each resource. Hence for simplicity, in the following we redefine the index i in $\boldsymbol{D_i}$ to be the index of a VM rather than the index of a specific resource on a VM.

A. Ticket Optimization Formulation

We formally introduce the problem, including notations and constraints, for resizing all co-located VMs on a single box. The foremost important constraint is that the summation of allocated virtual resources should be less than or equal to the total available virtual resource, i.e., $\sum_i C_i \leq C$, where C_i denotes the virtual capacity allocated to VM i, and C is the total available virtual capacity at the box. The decision variable is C_i and needs to be determined at the beginning of the resizing horizon.

The prediction module provides all demand series values for the entire resizing window, equal to T ticketing windows, for VM i, $\boldsymbol{D_i} = \{D_{i,1}, \dots D_{i,T}\}$. We introduce an indicator variable, $I_{i,t}$, when $I_{i,t} = 1$ a usage ticket occurs to VM i at ticketing window t, because the demand exceeds a certain threshold of the capacity, say, αC_i (e.g., $\alpha = 0.6$); otherwise $I_{i,t} = 0$. We aim to minimize the total number of tickets occurring on all co-located VMs during the resizing window. Thus, we can write the objective function as $\sum_i \sum_t I_{i,t}$. In summary, we can define the ticketing optimization problem as:

$$(\mathbb{R}) \min \quad \sum_i \sum_t I_{i,t} \tag{4}$$

$$s.t. \quad \sum_i C_i \leq C \tag{5}$$

$$D_{i,t} - \alpha C_i \leq D_{i,t} I_{i,j} \tag{6}$$

$$I_{i,t} \in \{0, 1\} \tag{7}$$

Constraint (6) ensures that $I_{i,t} = 1$, when the demand exceeds the ticket threshold, αC_i; otherwise the objective function drives $I_{i,t}$ to zero. The problem $\mathbb{R}$ is a classical mixed integer linear programming (MILP), whose complexity greatly depends on the number of integer variables, i.e., the indicator variables $I_{i,t}$ in our case. The number of indicator variables for each box is thus the product of the number of ticketing windows, T, and the number of VMs, M.

1) Resizing Algorithm: Instead of resorting to a standard MILP solvers, such as CPLEX [18], we transform the original problem into a multi-choice knapsack problem by Lemma 4.1: the optimal size for each VM must be equal to one of the demand values in $\boldsymbol{D_i}$ or 0. The advantages of transforming the original problem into a MCKP are twofold: (i) there exist a large number of efficient algorithms for MCKP and (ii) it

allows for a reduction of the number of integer variables. We elaborate on the second point after formally introducing the transformation of the original optimization problem to MCKP.

Lemma 4.1: For VM i, the optimal size $C_i* \in \boldsymbol{D_i} \cup \{0\}$, $\boldsymbol{D_i} = \{D_{i,1}, D_{i,2} \dots D_{i,T}\}$.

Proof: If there exists an optimal solution (C_i*) for each VM (i) for the resizing problem, C_i* has to be in one of the three ranges: $[0, min\{\boldsymbol{D_i}\})$, $[min\{\boldsymbol{D_i}\}, max\{\boldsymbol{D_i}\})$, and $[max\{\boldsymbol{D_i}\}, +\infty)$. If C_i* is less than $min\{\boldsymbol{D_i}\}$, we argue that C_i* could be set to 0 and the objective function stays unchanged while the constraints are not violated. Similarly, it is proven that if C_i* is not less than $max\{\boldsymbol{D_i}\}$, C_i* can be set to $max\{\boldsymbol{D_i}\}$. If C_i* is in $[min\{\boldsymbol{D_i}\}, max\{\boldsymbol{D_i}\})$, sort $\boldsymbol{D_i}$ in a descending order as $\boldsymbol{D_i}^{descend} = \{O_1, O_2, ..., O_p, O_{p+1}, ...\}$. Following the same reasoning, it is possible to determine that $\exists\ q$, $C_i* \in [O_q, O_{q+1})$. In addition, setting C_i* equal to O_q, the minimum objective function can be obtained without any constraint violation. Hence the optimal size C_i* is either in $\boldsymbol{D_i}$ or 0. ■

Based on Lemma 4.1, we can transform the original formulation into a multi-choice knapsack problem, whose complexity can be further simplified by reducing the number of indicator variables. We first introduce a reduced demand set with 0 added, denoted as $\boldsymbol{D_i'}$, containing the unique values of the original demands in decreasing order, $D'_{i,v+1} \leq D'_{i,v}$. According to Lemma 4.1, one of them is the optimal capacity. We note that $D'_{i,v}$ is not the same as $D_{i,t}$. The following small example illustrates the difference. Given a specific demand series $\boldsymbol{D_i} = \{30, 30, 40, 40, 23, 25, 60, 60, 60, 60\}$, its reduced series is $\boldsymbol{D_i'} = \{60, 40, 30, 25, 23, 0\}$ containing only the unique values plus 0 in descending order.

We introduce a new binary variable $Y_{i,v}$, denoting that the unique value $D'_{i,v}$ is chosen to be the capacity for VM i. The next step to reduce the problem into MCKP is to define the number of tickets, denoted $P_{i,v}$, seen by VM i when the value of $D'_{i,v}$ is chosen as capacity, i.e., $Y_{i,v} = 1$. Following the previous example of reduced demand set, we show an example of ticket calculation. Let us assume the current capacity is 70 and the ticketing threshold for issuing usage tickets is 60%. We thus know that demands greater than $70 \times 60\% = 42$ at any ticketing window will result into tickets. We can then obtain $\boldsymbol{P_i} = \{0, 4, 6, 8, 9, 10\}$. Due to the decreasing order of $\boldsymbol{D_i'}$, $\boldsymbol{P_i}$ has an increasing order, i.e., $P_{i,v+1} \geq P_{i,v}$. The total number of tickets for a box can thus be written as $\sum_i \sum_v Y_{i,v} P_{i,v}$ and the resource constraint of the total capacity as $\sum_i \sum_v Y_{i,v} D'_{i,v} \leq C$.

In summary, we reach a multi-choice knapsack problem, where items (in the original knapsack problem) are divided into subgroups and exactly one item needs to be selected from each group. Putting our problem into the context of multi-choice problem, we have M groups of VM demands and we need to choose exactly one demand from each group as their capacity. The decision variables are $Y_{i,v}$ denoting that a particular demand is chosen as the size for VM i, where $i \in [1, M]$ and that the number of tickets, $P_{i,v}$, can be seen as "weights". The transformed ticket reduction problem is:

$$(\mathbb{R}') \min \qquad \sum_i \sum_v Y_{i,v} P_{i,v} \tag{8}$$

$$s.t. \qquad \sum_i \sum_v Y_{i,v} D'_{i,v} \leq C \tag{9}$$

$$\sum_v Y_{i,v} = 1 \tag{10}$$

$$Y_{i,v} \in \{0, 1\} \tag{11}$$

The formulation of problem $\mathbb{R}'$ enables the introduction of a tunable parameter, ε, which decides the discretization of demand values. We illustrate this point using the running example of original series $\boldsymbol{D_i}$ and its reduced series $\boldsymbol{D_i'}$. The original formulation $\mathbb{R}$ has 11 integer variables (including the 0), whereas the transformed problem $\mathbb{R}'$ has only 6 integer variables. One can even further decrease the number of binary variables in $\boldsymbol{P_i}$ by discretizing the demand values, such as rounding off the first digit. For example using $\boldsymbol{D_i'} = \{60, 40, 30, 0\}$, where 23 and 25 are rounded up to 30. Another point worth mentioning is that we need to update the number of corresponding tickets too, i.e., $\boldsymbol{P_i} = \{0, 4, 6, 10\}$. Rounding up demands makes the resizing algorithm more aggressive in allocating resources. Consequently, we formally introduce a discretization factor, ε, which further reduces the complexity and provides a safety margin for resource allocation. We note that ε is only applied on the predicted series. In summary, the initial step computes $\boldsymbol{D_i'}$ from $\boldsymbol{D_i}$ using ε, and calculates their corresponding tickets, $\boldsymbol{P_i}$ for all co-located VMs i.

To solve the MCKP problem, we resort to the so-called minimal algorithm [19]. We illustrate the general idea in the context of our resizing problem. The algorithm chooses capacity candidates for each VM and shuffles around the capacity across VMs, comparing to the available capacity and marginal ticket reductions. For all VMs, it chooses capacity candidates that can incur a minimum number of tickets, i.e., starts from the maximum values in $\boldsymbol{D_i'}$. When there is no sufficient capacity to achieve such allocations for all VMs, the priority is given to the VM having the lowest marginal ticket reduction values (MTRV). MTRV represents the additional ticket increment when reducing one unit of capacity provisioning. Its formal definition is:

$$MTRV = \frac{P_{i,o} - P_{i,o-1}}{D'_{i,o-1} - D'_{i,o}}, \tag{12}$$

where o denotes the index of candidates in $\boldsymbol{D_i'}$. The VM with the lowest MTRV is always chosen to reduce the capacity provision from its current candidate value to the next one in $\boldsymbol{D_i'}$. Note that as $\boldsymbol{D_i'}$ is in decreasing order, the next candidate immediately implies a capacity reduction. Once the candidate list is updated, the same process continues until the sum of all candidates is less or equal to the available capacity.

For a practical implementation, in addition to the constraint of total available capacity, it is also imperative to consider the lower and upper bounds of capacity. In order to avoid spillovers of unfinished demands from previous ticketing windows, we impose a lower bound on the VM capacity size, such that its peak usage before resizing is satisfied. Moreover, as any VM is not able to use more resources than the available resource amount of the underlying physical box, we introduce the allocation upper bound based on the box resource capacity. We can easily incorporate such lower and upper bounds into our resizing algorithm by limiting the values in $\boldsymbol{D_i'}$ for each VM i.

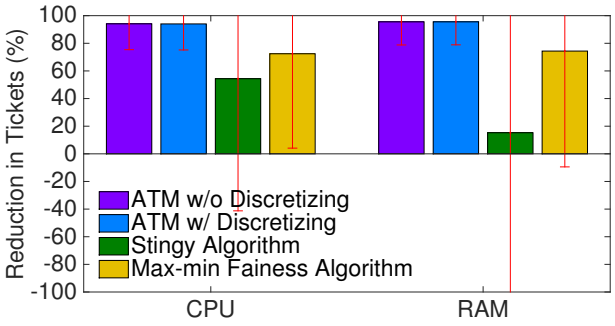


Fig. 8: Ticket reduction for CPU and RAM: comparing ATM, max-min fairness, and stingy algorithms.

B. Results on Usage Ticket Reduction

Prior to moving on to the evaluation of the full-fledged ATM, i.e., the combination of spatial-temporal prediction and resizing policy, we first show how effective the proposed resizing algorithm is against existing resource allocation heuristics. For a fair comparison, the demand inputs are based on the original dataset described in Section II, instead of prediction. We implement the max-min fairness algorithm [16] and a "stingy" algorithm which only allocates the capacity according to the lower bound, i.e., the maximum demand regardless of the ticket threshold, often used in practice. In contrast, the max-min algorithm starts to allocate to all VMs the demand of the smallest VM, considering its ticket threshold, and continues onto VMs in the increasing order of their demands until all capacity is exhausted.

Here, we evaluate the data of April 3, 2015 across all 6K boxes and set the threshold to trigger usage tickets to 60%: i.e., every 15-minute ticketing window the monitoring system checks if the average usage of CPU or RAM of each VM exceeds the 60% of the allocated capacity. Figure 8 summarizes the mean ticket reduction (in percent) and its standard deviation, when applying the proposed ATM resizing, max-min fairness, and stingy algorithms. As expected, the stingy algorithm is completely unaware of the ticket threshold. On average it achieves a ticket reduction of 54% and 15% for CPU and RAM, respectively. Max-min fairness reduces the tickets by around 70% for both CPU and RAM. This is still roughly 30% worse than the ATM resizing results. Due to the nature of favoring small VMs, large VMs can be severely punished under max-min fairness resulting in no ticket reduction and explains the high standard deviation under max-min fairness.

As a pleasant surprise, our resizing algorithm does exceptionally well. It achieves 95% and 96% usage ticket reductions for CPU and RAM, respectively, a remarkable improvement for both performance and cost. This is also attributed to the fact that the systems of the original traces are equipped with abundant resources, i.e., typically data centers are lowly utilized [5]. By simply shuffling resources across co-located VMs, we are able to achieve significant performance gain. Moreover, we also eliminate the overhead of inspecting and resolving a large number of usage tickets, a process that is known to be expensive.

C. Actuation of Virtual Capacity

Cloud data center tenants are typically charged by the amount of virtual resources, for example, the number of virtual cores. Consequently, any practical sizing policy should adhere to such a constraint, due to accounting and financial concerns. Therefore, to enforce the virtual capacity limits decided in our algorithm, we use the control groups (cgroups) feature of the Linux kernel [20]. Cgroups allow to limit, account for, and isolate resources usages of groups of processes. By placing the processes and threads relating to each VM in a separate cgroup, we can dynamically change the resource usage limits for each VM. To simplify the cgroups configuration, we expose the resource limits through a web-based API by running a small daemon at each hypervisor. The advantage of cgroups over directly modifying the allocated virtual VM resources is that the latter typically requires a restart of the guest OS while the former can be changed on-the-fly without disrupting the VM operation. Moreover, cgroups offer a finer-grained CPU control with an almost continuous CPU limit control rather than the stepwise decrease/increase of virtual cores.

V. Evaluation

We extensively evaluate ATM not only on a large number of data center production traces but also experimentally on a cluster running MediaWiki. We focus on presenting the effectiveness of ATM in ticket reduction to improve system dependability and to reduce the high cost associated with ticket resolution. In the remaining of this section, we assume that usage tickets related to CPU and RAM are automatically issued when VM utilization is greater than 60%.

A. Production Systems

We focus on a subset of boxes from the data center trace (400 boxes) which have no gaps in their traces. The remaining box traces suffer, throughout the 7 days of the trace, from occasional gaps with no data. We show how different configurations of ATM can proactively reduce the number of tickets. We engage training of the signature series for 5 days and then apply ATM and VM resizing for the following day. We stress that this analysis is post-hoc, i.e., we can not change the size of the actual VMs in the trace, we focus only on the prediction accuracy and ticket reduction via ATM. On the contrary, in the experimental evaluation on the MediaWiki cluster presented in the Section V-B, we do also illustrate VM resizing in a working system.

For the spatial models, we consider DTW and CBC clustering techniques and set the discretization factor $\varepsilon = 5$. The temporal models used for the signature series are neural networks [7]. ATM performs the prediction of 16000 usage series, each of which has 96 ticketing windows, with each window being 15 minutes long. After obtaining the predicted series, ATM triggers the resizing algorithm for every box to determine the near optimal CPU and RAM capacity for all co-located VMs. We note that results presented in this section differ from Section III and IV, where only the proposed spatial models and resizing algorithms are evaluated individually, excluding the temporal prediction models. Here, we have the full effect of both prediction models.

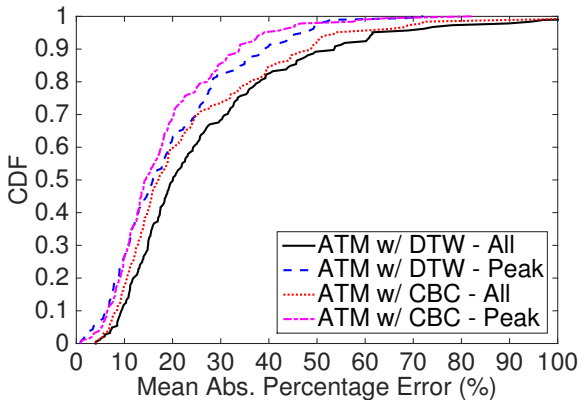


Fig. 9: CDF of prediction accuracy of ATM on 400 production servers: ATM_{CBC}, ACM_{DTW}.

1) Prediction Errors: Figure 9 presents the CDF of the prediction accuracy of ATM in terms of APE with different spatial models, i.e., DTW and CBC clustering. For CPU and RAM usage, we use the inter-resource model, i.e., signature series are a mix of CPU and RAM. The average prediction errors of resources usage per box are 31% and 23%, for DTW and CBC, respectively. These are only slightly higher than the errors without the temporal models presented in Section III. The figure also illustrates the CDF of the mean absolute errors for peak demands, i.e., usage higher than 60%. The average peak errors across all boxes are 20% and 17% for DTW and CBC, respectively. This shows that neural networks can capture well the temporal dynamics of the signature series. We further note that this high accuracy of temporal models is achieved at a high computational time and long historical data, i.e., 5 days, whereas the prediction of dependent series via spatial models has a negligible cost. We also note that the reduction in demand series for this subset of 400 boxes is similar to results shown in Section III across 6K boxes.

2) Ticket Reduction: Figure 10 compares the results of average ticket reduction using two different versions of ATM against the max-min fairness, and stingy policies, see Section IV. Each bars illustrate the mean and standard deviation of ticket reduction across boxes divided into CPU and RAM tickets. The key observations are the following. Both versions of ATM are able to achieve a higher ticket reduction, around 60% and 70% for CPU and RAM, respectively, compared to the other two heuristics. We like to point out that the standard deviation is high for all four strategies indicating huge difference across boxes. Different from the resizing results shown in Section IV, max-min fairness shows worse reduction results than stingy. This can be explained by the observed high variability across the chosen 400 boxes which shows that max-min fairness could even result in a increase of the number of tickets for a subset of the boxes, see the range of standard deviation. Max-min fairness favors small VMs while dissatisfying big VMs, which results in more ticket violations than the other policies. Another fact worth mentioning is that both versions of ATM are able to achieve higher RAM ticket reductions, due higher RAM provisioning compared to CPU.

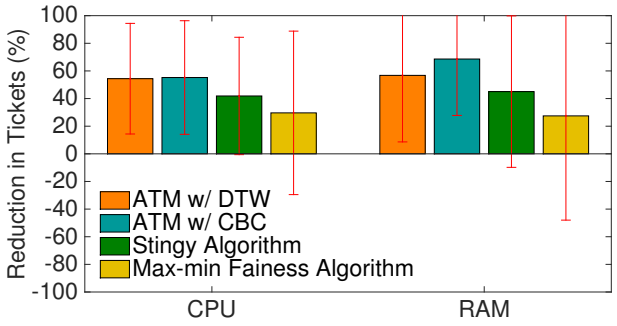


Fig. 10: Comparing ticket reduction: ATM, max-min fairness, and stingy resizing algorithms.

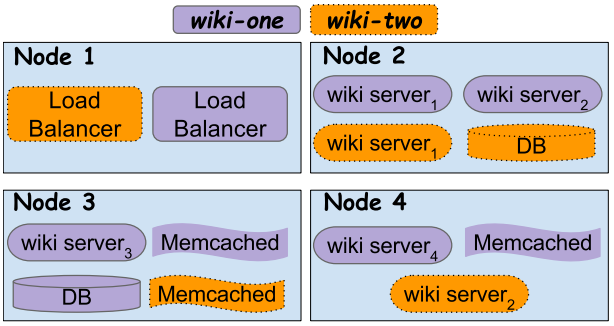


Fig. 11: MediaWiki testbed.

B. ATM on a MediaWiki Cluster

We experimentally evaluate our ticket reduction techniques also on a cluster running MediaWiki, a latency-sensitive 3-tier web application composed by Apache (v2.4.7) as the application server frontend, memcached (v1,4.14) as in-memory key-value store, and MySQL (v5.5.40) as the database backend. The testbed is composed of four identical physical servers. Each server runs Ubuntu server 14.04 LTS and is equipped with 16 GiB of DDR3 RAM with up to 41.6 GiB/s bandwidth, a 4-core Intel Core i7 3820 processor @ 3.6 GHz with SMT, one 2-TB Sata III 7200 rpm hard disk, and one Gigabit Ethernet adapter. Three servers host the VMs using QEMU-KVM (QEMU v2.0 with KVM on Linux kernel 3.13) as hypervisor. Each VM comprises two virtual CPUs and 4 GiB of RAM. The forth server is used as the experiment orchestrator and load generator. Each application tier is deployed into a separate VM. We consider a scenario of hosting two MediaWiki applications on these 4 physical servers, termed as wiki-one and wiki-two, see Figure 11. For wiki-one, there are 4 Apache servers, 2 Memcached, and 1 DB, whereas there are only 2 Apache, 1 Memcached, and 1 DB in wiki-two. For each wiki, we have one load balancer that distributes the requests across the different apache front-ends. The workload generator creates requests alternating between low and high intensity periods, each lasting one hour.

Figure 12 illustrates the CPU usage series across all VMs located on nodes 2, 3 and 4 against the ticketing threshold set to 60%. The figure shows the CPU usage levels without and with ATM resizing. One can observe that indeed resizing

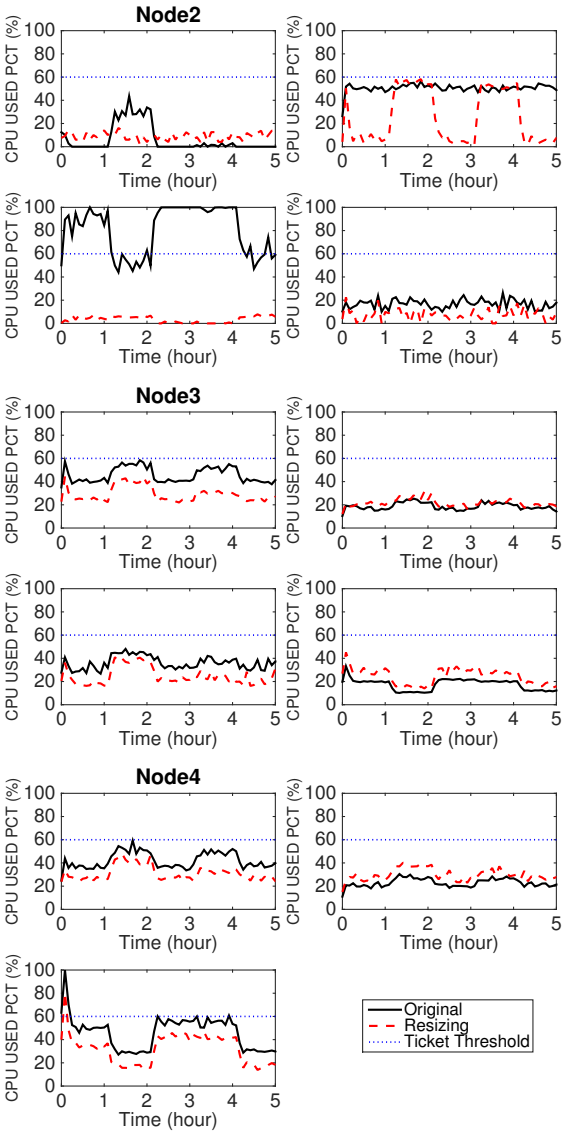
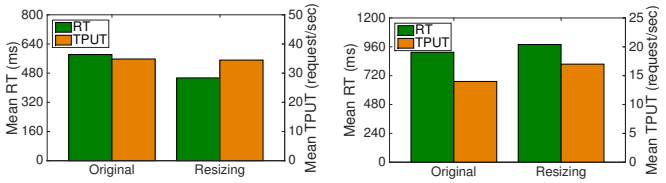


Fig. 12: Overtime plots of CPU utilization for VMs located on Node 2, 3 and 4, with and without resizing.

is very effective in achieving CPU usage levels across time and all VMs below the 60% threshold. The consequent ticket reduction is dramatic: tickets drop from 49 to only 1.

Besides ticket reduction, we also show performance values for the two wiki applications, see Figure 13. The figure plots the the average user latencies (response times) and average throughput (the average number of successful served requests per unit of time). For wiki-one, the mean response times with resizing decrease from 20% (from 582 ms to 454 ms) comparing to the original experiment, whereas throughputs are maintained at almost the same levels. For wiki-two, throughput increases by more than 20% (from 14 to 17 requests/sec) while response time increases by 7% (from 915 ms to 979 ms). This suggests that with ATM, the servers can fully serve the offered load, meet good performance values, while at the same time keeping the number of tickets to a minimum demonstrating the ultimate goal of ATM.



(a) wiki-one (b) wiki-two

Fig. 13: Performance comparison for wiki-one and wiki-two: original and resized with ATM.

VI. Related Work

Ticketing systems are widely used to improve on system dependability, e.g., slow responsiveness, failure [4], software bugs [21], [22] and system misconfigurations [23]. Prior art in ticketing systems centers on two directions: derive system management for software concurrency [21], database systems [3], and distributed data-intensive systems [24] but also to develop automatic detection systems for different types of tickets, bugs [22] and software misconfigurations by leveraging the rich correlation between configuration entries [23]. Machine learning has been used for automating ticket resolution recommendation [25], [26], [9]. To the best of our knowledge, there are no proactive methodologies for preventing ticket issuing, with the exception of models for database reconfiguration [27]. The proposed ATM policy fills this gap by not only deriving management insights for usage ticket patterns, but also by developing novel prediction and ticket avoidance strategies using VM resizing.

Time series prediction and analysis have been viewed as an excellent way to develop proactive system management policies [28], [29]. Temporal models such as ARIMA models [10] have been widely used to predict time series with strong seasonality. Sophisticated neural network models show a strong promise in capturing highly irregular time series at a cost of long training overheads [30]. Time series clustering aims to explore spatial dependency, either through their original series, e.g., DTW [12], or extracted features [11], e.g., moments. ATM combines spatial with temporal models to contain the cost of neural network training and scales well for very large numbers of time series.

Virtualization technology has become the industry standard offering great opportunities to multiplex physical resources over a large number of VMs. There are several ways to change the efficiency of resource multiplex ratios: by sizing the virtual resource capacities [31] and by dynamically consolidating or migrating VMs [32], [33]. While dynamically changing the degree of VM consolidation is shown effective to take advantage of the time variability of the workload [34], the overhead of migrating VMs can greatly reduce its performance benefits. On the contrary, sizing resource of co-located VMs incurs less system overhead [31]. A central question of multiplexing resources is how to strike a good tradeoff of fairness and performance for workloads, e.g., latency [35] and throughput [36]. Fairness driven policies, e.g., max-min fairness, proportional fairness, and bottleneck resource fairness [37], have been proposed for various systems components, including storage systems [36] and networks [38]. The sizing algorithm proposed

in ATM differs from related work by its objective to reduce the number of usage tickets. While max-min fairness also reduces the number of tickets, it cannot achieve this as effectively as ATM since ticket reduction is a side-effect rather than a main focus.

VII. Concluding Remarks

We presented ATM, a methodology to achieve efficient VM resizing so as to reduce VM usage tickets that are issued in production data centers. We have shown the effectiveness of ATM in predicting usage series in production data centers by exploiting spatial usage patterns of co-located VMs within the same box and by using detailed prediction of a small subset of the usage series, allowing the methodology to scale well. This prediction drives the development of a VM resizing policy that is shown effective on a production trace and a working prototype. In our future work we intend to use ATM's prediction abilities to drive online dynamic workload management.

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).

References

[1] Y. Liang, Y. Zhang *et al.*, "Bluegene/l failure analysis and prediction models," in *Proceedings of the 36th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2006, pp. 425–434.

[2] I. Giurgiu, J. Bogojeska *et al.*, "Analysis of labor efforts and their impact factors to solve server incidents in datacenters," in *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2014, pp. 424–433.

[3] I. Giurgiu, A.-D. Almasi, and D. Wiesmann, "Do you know how to configure your enterprise relational database to reduce incidents?" in *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 339–347.

[4] R. Birke, I. Giurgiu *et al.*, "Failure analysis of virtual and physical machines: patterns, causes and characteristics," in *Proceedings of the 44th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2014, pp. 1–12.

[5] R. Birke, A. Podzimek *et al.*, "State-of-the-practice in data center virtualization: toward a better understanding of VM usage," in *Proceedings of the 43rd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–12.

[6] R. Birke, M. Bjoerkqvist *et al.*, "(Big) data in a virtualized world: volume, velocity, and variety in cloud datacenters," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST)*. USENIX, 2014, pp. 177–189.

[7] J. Xue, F. Yan *et al.*, "PRACTISE: robust prediction of data center time series," in *Proceedings of the 11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 126–134.

[8] MediaWiki. [Online]. Available: https://www.mediawiki.org/wiki/MediaWiki

[9] M. M. Botezatu, J. Bogojeska *et al.*, "Multi-view incident ticket clustering for optimal ticket dispatching," in *Proceedings of the 21th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, 2015, pp. 1711–1720.

[10] C. Chatfield, *The analysis of time series: an introduction*. CRC press, 2013.

[11] B. D. Fulcher and N. S. Jones, "Highly comparative feature-based time-series classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3026–3037, 2014.

[12] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16, 1994, pp. 359–370.

[13] L. Rokach and O. Maimon, "Clustering methods," in *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 321–352.

[14] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[15] M. Kutner, C. Nachtsheim, and J. Neter, *Applied Linear Regression Models*. McGraw-Hill Education, 2004.

[16] L. Tassiulas and S. Sarkar, "Maxmin fair scheduling in wireless networks," in *Proceedings of the 21st IEEE International Conference on Computer Communications (INFOCOM)*, vol. 2. IEEE, 2002, pp. 763–772.

[17] A. Ghodsi, M. Zaharia *et al.*, "Dominant resource fairness: fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2011, pp. 323–336.

[18] CPLEX Optimizer. [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html

[19] D. Pisinger, "A minimal algorithm for the multiple-choice knapsack problem," *European Journal of Operational Research*, vol. 83, no. 2, pp. 394–410, 1995.

[20] CGROUPS. [Online]. Available: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[21] S. Lu, S. Park *et al.*, "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2008, pp. 329–339.

[22] A. Nistor, P.-C. Chang *et al.*, "Caramel: detecting and fixing performance problems that have non-intrusive fixes," in *Proceedings of the 37th International Conference on Software Engineering (ICSE)*. IEEE, 2015, pp. 902–912.

[23] J. Zhang, L. Renganarayana *et al.*, "Encore: exploiting system environment and correlation information for misconfiguration detection," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 687–700, 2014.

[24] D. Yuan, Y. Luo *et al.*, "Simple testing can prevent most critical failures: an analysis of production failures in distributed data-intensive systems," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 2014, pp. 249–265.

[25] W. Zhou, L. Tang *et al.*, "Resolution recommendation for event tickets in service management," in *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 287–295.

[26] Q. Shao, Y. Chen *et al.*, "Easyticket: a ticket routing recommendation engine for enterprise problem resolution," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1436–1439, 2008.

[27] I. Giurgiu, M. Botezatu, and D. Wiesmann, "Comprehensible models for reconfiguring enterprise relational databases to avoid incidents," in *Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, 2015, pp. 1371–1380.

[28] N. Tran and D. A. Reed, "Automatic ARIMA time series modeling for adaptive I/O prefetching," *IEEE Transactions on Parallel Distributed Systems*, vol. 15, no. 4, pp. 362–377, 2004.

[29] Z. Zhuang, H. Ramachandra *et al.*, "Capacity planning and headroom analysis for taming database replication latency: experiences with linkedin internet traffic," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, 2015, pp. 39–50.

[30] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS)*, 2014, pp. 855–863.

[31] S. Spinner, N. Herbst *et al.*, "Proactive memory scaling of virtualized applications," in *Proceedings of the 8th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 277–284.

[32] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proceedings of*

the 30th IEEE International Conference on Computer Communications (INFOCOM). IEEE, 2011, pp. 71–75.

[33] H. W. Choi, H. Kwak *et al.*, "Autonomous learning for efficient resource utilization of dynamic VM migration," in *Proceedings of the 22nd International Conference on Supercomputing (ICS)*. ACM, 2008, pp. 185–194.

[34] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and QoS-aware cluster management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2014, pp. 127–144.

[35] A. Gulati, A. Merchant, and P. J. Varman, "pClock: an arrival curve based approach for QoS guarantees in shared storage systems," in *Proceedings of the 33rd ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 2007, pp. 13–24.

[36] H. Wang and P. Varman, "Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST)*. USENIX, 2014, pp. 229–242.

[37] T. Bonald and J. Roberts, "Multi-resource fairness: objectives, algorithms and performance," in *Proceedings of the 41st ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 2015, pp. 31–42.

[38] A. Sridharan and B. Krishnamachari, "Maximizing network utilization with max–min fairness in wireless sensor networks," *Wireless Networks*, vol. 15, no. 5, pp. 585–600, 2009.