AUDIBLE: A Convolution-Based Resource Allocator for Oversubscribing Burstable Virtual Machines

Seyedali Jokar Jandaghi
saj@cs.toronto.edu
University of Toronto
Canada

Kaveh Mahdaviani
mahdaviani@cs.toronto.edu
University of Toronto
Canada

Amirhossein Mirhosseini
miramir@umich.edu
University of Michigan
USA

Sameh Elnikety
samehe@microsoft.com
Microsoft Research
USA

Cristiana Amza
amza@ece.utoronto.ca
University of Toronto
Canada

Bianca Schroeder
bianca@cs.toronto.edu
University of Toronto
Canada

Abstract

In an effort to increase the utilization of data center resources cloud providers have introduced a new type of virtual machine (VM) offering, called a burstable VM (BVM). Our work is the first to study the characteristics of burstable VMs (based on traces from production systems at a major cloud provider) and resource allocation approaches for BVM workloads. We propose new approaches for BVM resource allocation and use extensive simulations driven by field data to compare them with two baseline approaches used in practice. We find that traditional approaches based on using a fixed oversubscription ratio or based on the Central Limit Theorem do not work well for BVMs: They lead to either low utilization or high server capacity violation rates. Based on the lessons learned from our workload study, we develop a new approach to BVM scheduling, called Audible, using a non-parametric statistical model, which makes the approach light-weight and workload independent, and obviates the need for training machine learning models and for tuning their parameters. We show that Audible achieves high system utilization while being able to enforce stringent requirements on server capacity violations.

ACM Reference Format:
Seyedali Jokar Jandaghi, Kaveh Mahdaviani, Amirhossein Mirhosseini, Sameh Elnikety, Cristiana Amza, and Bianca Schroeder. 2024. AUDIBLE: A Convolution-Based Resource Allocator for Oversubscribing Burstable Virtual Machines. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3620666.3651376

1 Introduction

The under-utilization of data center resources has been a long-standing problem. In public cloud platforms, a common reason for the low utilization of resources is that the resource demands of most applications vary over time, while the size of a regular virtual machine (VM) is fixed (e.g. 2 virtual CPU cores) forcing customers to size their VMs for application peak usage. To combat this problem cloud providers have over the last years introduced a new type of virtual machine offering, called a *burstable virtual machine (BVM)*, which is intended to accommodate workloads with time-varying CPU demands. In particular, a BVM accumulates *credit* while its resource usage stays below some specified *baseline* and can use these credits to occasionally burst above the baseline up to some specified *Peak CPU Usage*.

BVMs have the potential to benefit both customers and cloud providers. Customers with time-varying workloads benefit financially as they no longer have to pay for cores that they do not use most of the time (which is the case with fixed-sized regular VMs sized for peak usage). In fact, there are already a number of examples in the literature for how different applications can make use of BVMs [2, 3, 19, 23, 25, 27, 30, 31]. For cloud providers, BVMs provide an opportunity to increase resource utilization by packing VMs more tightly, as the BVM type makes it explicit to the resource allocator that a customer's workload is expected to stay below some baseline for most of the time and there is a mechanisms in place that limits how often it might burst above this baseline.

Realizing the potential of BVMs for improving resource utilization in cloud platforms critically hinges on the resource allocator. In particular, the resource allocator in a public cloud platform has to carefully balance the trade-off between *increasing utilization* and limiting *server capacity violations*, i.e. a situation where the available server capacity is not sufficient to meet the Service Level Objectives (SLOs) of all VMs running on it. In the case of servers running BVMs, a server capacity violation refers to a scenario where a BVM has accumulated credit that it wants to use to burst, but the server has no capacity to accommodate the burst. With resource oversubscription, the cloud operator specifies a limit on the acceptable rate of server capacity violations (e.g., below 1% to reduce negative impact on customer experience) for the servers in a cluster, and the goal of the resource allocator is to maximize utilization while keeping server capacity violations below the specified limit.

While there is a large body of work on resource allocation in cloud platforms, we are not aware of any work on resource allocation policies for BVMs. The goal of this paper is to close this gap. Our paper makes the following contributions:

- We provide the first study of BVM workloads, based on data collected from production systems in Microsoft Azure, spanning more than 4 million virtual machine. Our study provides lessons to guide resource allocator design and highlights some key differences to regular VM workloads.
- We use large-scale simulations, driven by production traces from Microsoft Azure to evaluate several state-of-the-art resource allocators designed for regular VMs, as well as the current allocator used at Microsoft Azure for BVMs. We find that existing methods either result in low utilization or are not able to reliably ensure limits on server capacity violations.
- We identify some fundamental reasons why previous approaches fall short. For example, we observe that approximations based on the Central Limit Theorem, which are often used to model server utilization [4, 6, 8], are not sufficiently accurate in the tail of the distribution of BVM workloads.
- We design and evaluate a new approach to BVM scheduling, called *Audible*, and show that it achieves high system utilization while being able to enforce even stringent requirements on server capacity violations. Audible is a non-parametric statistical model based on observed empirical distributions. This approach makes Audible light-weight and workload independent, and obviates the need for training of machine learning models or tuning of magic parameters.
- For the benefit of others working in this area, we also outline lessons learned from our work, including "negative results" from alternative novel approaches, besides Audible, that we explored and why they failed.
- We provide an artifact with this paper that includes the first dataset of BVMs from a public cloud and the simulator we designed to evaluate different scheduling algorithms, making our results reproducible and providing a framework for future work in this area.

2 Background and Motivation

BVMs: A BVM is a new type of virtual machine that has dynamic CPU assignment over its life-cycle and is designed for workloads that burst on occasion. BVM offerings in both Microsoft and Amazon are defined by four parameters: the *baseline CPU*, the *bucket size*, the *peak CPU*, and the number of *initial tokens*. While a BVM is running it collects tokens at a constant rate, which is specified by the *baseline* parameter. The *bucket size* determines the maximum number of tokens that can be accumulated. Each token represents a credit allowing the BVM to use one core at full capacity for one minute. CPU credits can be used in fractions, e.g., half a credit can be spent to use a core for half a minute. Table 1 shows the BVM configurations offered by Microsoft and Amazon.

Peak CPU (CPU Core(s))	Baseline(s) (Token per Minute)	Bucket Size(s) (Token)	Initial Tokens (Token)
1	0.05, 0.1, 0.2	72, 144, 288	30
2	0.4, 0.6	576, 864	60
4	0.9	1296	120
8	1.35	1944	240
12	2.02	2909	360
16	2.70	3888	480
20	3.37	4860	600

Table 1. *BVM Configurations at Microsoft and Amazon.*

VM Scheduling: VM scheduling typically involves three levels of decision making. The first level chooses which one of many clusters an incoming VM should be placed on. This decision is often based on load balancing or geographical considerations. The next step is to narrow down the servers in the chosen cluster to a set of *candidate* servers, based on a number of considerations. One of the most important constraints and the focus of this paper is to determine whether a candidate server has sufficient free capacity to host the VM. We call a candidate server a *valid* candidate if it has sufficient free capacity to host the incoming VM. (Other considerations include for example the type of hardware requested by the incoming VM or placement restrictions based on fault domains – these are orthogonal to our work and not topic of this paper). The third level of decision making determines which valid candidate server to place the VM on. This could be based on a heuristic like worst-fit for load balancing, best-fit for tighter packing or random selection. If there is no valid candidate server (i.e. no server in the cluster has sufficient capacity) the VM placement request is rejected.

Goals: The focus of our paper is to provide methods to increase server utilization in a public cloud platform while enforcing bounds on server capacity violations. At the core of this problem is determining for a given BVM and candidate server, whether the candidate server is a valid candidate, i.e. given the other BVMs already running on the server can we place the new BVM on this server without creating a significant risk of future server capacity violations. In the context of BVMs a capacity violation refers to a situation where the aggregate demand of all the VMs within the boundaries of their available tokens and Peak CPU Usages exceeds the capacity of the server. In such cases at least some BVMs have to be throttled despite having accumulated tokens, which violates the SLO for a BVM and negatively impacts customer

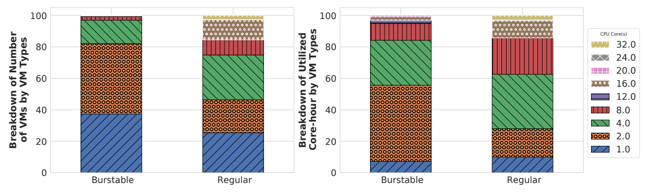

Figure 1. *Breakdown of the number of VMs and CPU-hours based on VM sizes (number of virtual cores) for burstable VMs (left) and regular VMs (right).*

satisfaction. Cloud providers therefore strive to keep the rate of capacity violations very low.

A resource allocator for BVMs needs to ensure that the rate of capacity violations stays within a limit that the provider has specified for that cluster or server, i.e., it needs to enforce an upper bound on the fraction of time periods that a server is allowed to experience a capacity violation, while also allowing for maximal utilization of server resources.

VM placement decisions need to be made conservatively as poor decisions are hard to correct later: VM migration is expensive and negatively impacts customer experience and is therefore avoided if at all possible. At the same time overly conservative decisions will hurt utilization.

3 BVM Workload Study

In this section, we present the first study of Burstable Virtual Machines (BVMs) production workloads. Our study is based on data collected on production machines at Microsoft Azure over a one-week period in 2021, as well as an older BVM trace and traces of regular VMs. The dataset comprises records from 4 million BVMs, that include, for example, the average and maximum CPU utilization, logged at five-minute intervals, as well as information regarding BVM configurations and the creation/termination events.

3.1 CPU utilization of BVMs

We begin by categorizing the BVMs in our dataset based on the number of virtual CPUs they have. Figure 1 shows the breakdown for burstable VMs (left) in comparison with regular VMs (right). Each graph shows both the frequency of each VM size as well as a breakdown of the aggregate CPU-hours consumed by VMs of each size. We observe that BVMs tend to be small. For example, nearly 20% of BVMs have only 1 virtual core compared to 5% of regular VMs. That is good news from a scheduling point of view as summarized in the lesson below:

Lesson 1: As the majority of BVMs are small, a server will be able to host a large number of them. Multiplexing across a large number of VMs creates potential for oversubscription.

Next, we consider how BVMs make use of the CPU resources allocated to them. Microsoft Azure currently allocates 2X the BVM's baseline for all BVM types, which on average corresponds to 38% of the BVM type's specified CPU peak. We convert the average CPU usage measured for each BVM in our dataset into a percentage of the total CPU allocated for the BVM. Figure 2 shows the resulting CDF, as well as the corresponding CDF for regular VMs. We observe that VMs and BVMs utilize only a small fraction of the CPU resources allocated to them. The BVM in the median uses only 7.5% of its allocated CPU and even the BVM in the 95th percentile uses only 38.9% of its allocated resources.

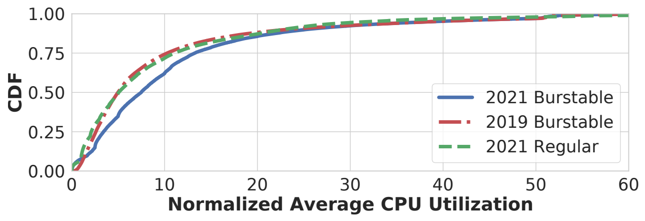

Figure 2. *CDF of Average CPU Utilization, normalized by the size of the (B)VM and weighted according to the lifetime of each VM. The vast majority of BVMs greatly underutilizes resources.*

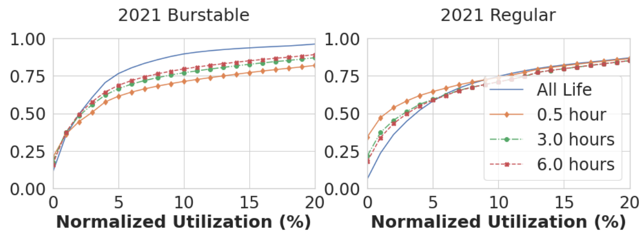

Figure 3. *CDF of CPU utilization at different stages of a BVM's lifecycle. We observe that CPU utilization is higher early in life compared to the rest of the lifetime.*

Lesson 2: BVMs greatly underutilize the CPU resources allocated to them further increasing the potential for oversubscription of resources.

Next we look at how stable a BVM's CPU usage is over its lifetime. Figure 3 shows the CDF of the average CPU utilization during the first 0.5, 6 and 12 hours of lifetime compared to the entire lifetime for BVMs and VMs. The figure reveals a distinct trend: BVMs' CPU utilization is higher during the first hours of lifetime compared to the rest of the lifetime. The same trend does not exist for regular VMs. The reason is likely that BVMs are instantiated with a set of initial tokens, which allow them to run at a higher utilization in the beginning. Static allocation methods that assume a fixed resource usage throughout a BVMs lifetime will likely be wasteful of resources (or underestimate resources during the early life of a BVM).

Lesson 3: Efficient resource allocation methods need to be aware of the dynamically changing CPU demands of BVMs, in particular their higher resource usage early in life.

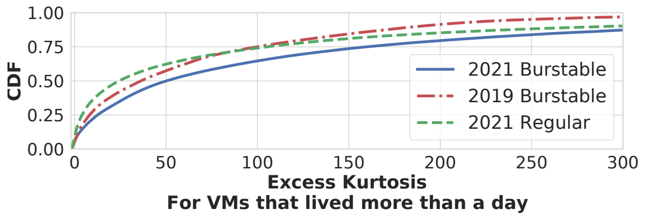

Figure 4. *CDF of the Kurtosis of CPU utilization of burstable versus regular VMs. Kurtosis is a measure of the "tailedness" of a probability distribution.*

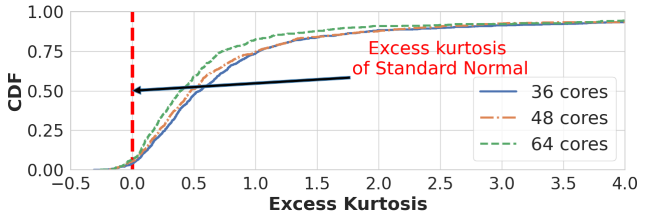

Figure 5. *Analyzing the skewness in the CPU usage of servers of different sizes packed with burstable VMs. The excess kurtosis of a normal distribution is 0, and positive values indicate right-skewness in the aggregate CPU usage.*

One aspect that makes tight packing of VM workloads hard is high variability and spikes in resource usage. Since burstable VMs are designed for workloads with time-varying CPU demands a natural question is how much burstier their workloads are than regular VMs. Towards this end, Figure 4 plots the CDF of the kurtosis of the BVM workloads and the regular VM workloads in our datasets. Kurtosis is a measure of the "tailedness" of a probability distribution. We observe significantly higher kurtosis for the burstable VM workloads, in particular for the more recent dataset. The mean and median kurtosis of BVMs in the 2021 BVM dataset is 147 and 50, respectively, compared to a mean of 71.46 and median of 36.23 for regular VMs, and this trend is rapidly shifting towards more burstiness as is evident when comparing the 2021 dataset with that of 2019.

Lesson 4: BVM workloads exhibit higher skewdness, which is a measure of distribution asymmetry, than regular BVM workloads. A VM scheduler needs to be able to handle this additional burstiness.

Finally, we pay some special attention to the characteristics of the distribution of the resource usage of an aggregate of BVMs. One common approach to oversubscribing resources is to rely on a Gaussian approximation of the aggregate resource usage of a group of VMs using the Central Limit Theorem (CLT). While some of the assumptions of the CLT might not always hold in practice (independent and identical distribution of the resource usage of different BVMs on a server) previous work has found approximation based on the CLT to be useful for scheduling datacenter workloads [4, 8, 14, 16].

In contrast, we find that while the normal distribution captures the overall shape of the CDF well, the fit is not tight at the tail of the distribution. We take a closer look at the tail behavior of the resource usage distribution of BVM aggregates in Figure 5. For this figure we repeatedly packed servers of different sizes (36, 48, 64 cores) with BVMs and compared the excess kurtosis of the resulting CPU usage distribution with that of a normal distribution. (The excess kurtosis of the normal distribution is zero, and positive values for excess kurtosis indicate right-skewness of a distribution). We see that in nearly 93% of the servers the aggregate CPU usage exhibits a rightward skew, with some servers exhibiting a significant deviation from the expected skewness of a normal distribution. This discrepancy highlights a limitation in the Gaussian model employed by CLT in effectively representing extreme events on the right side of the tail, specifically rare bursts in aggregate usages. These bursts that the tail of the distribution represents might be critical when trying to enforce stringent upper bounds on server capacity violations.

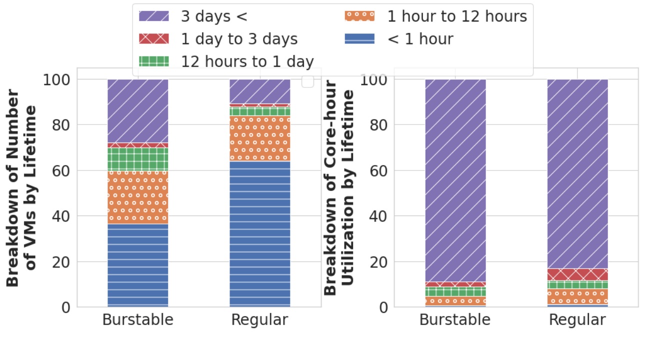

Figure 6. *Breakdown of the number of VMs and CPU-hours based on VM lifetimes for burstable (left) and regular VMs (right).*

Lesson 5: Approximations of server CPU usage based on the Central Limit Theorem might not be sufficiently accurate to enforce stringent requirements on the server capacity violations, where the tail of the distribution matters most.

3.2 Lifetime

Next we analyze the lifetime (duration) of BVMs. Figure 6 breaks down the number of VMs and the associated CPU-hours based on the lifetime of the VM, for burstable VMs and regular VMs. We observe that for both burstable and regular VMs the short-running VMs make up the majority of all VMs, however long-running VMs account for the majority of CPU hours. For example, while burstable VMs longer than a day

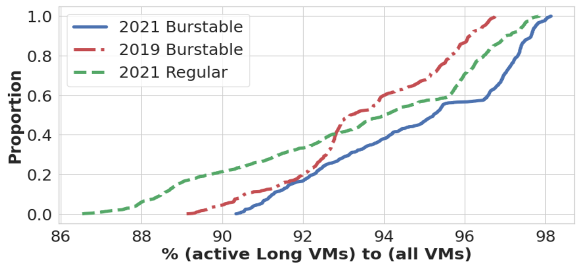

Figure 7. *CDF of the percentage of all running VMs that are long-running (longer than a day) at each 5-minute sampling time.*

make up less than 30% of all BVMs, they account for more than 90% of total CPU hours.

The observation above has implications on what the breakdown of VMs, based on lifetime, looks like if we take a snapshot of a data center at a given point in time. Figure 7 shows the CDF of the percentage of all active BVMs that are *long* (lifespan of more than a day) at any given sampling interval in our data set (5-minute intervals). We observe that both on average and in the median long BVMs constitute around 94% of all active BVMs and there is relatively little variation.

Lesson 6: While the majority of all VMs are short-running, at any given point in time the vast majority of VMs running in a cluster will be long-running VMs. Because of the different CPU usage characteristics of short and long VMs it is important to realistically reflect this ratio in simulations.

4 Adapting Traditional Schedulers to BVMs

In this section, we describe three different approaches for scheduling BVMs that we use as baselines. Each of the three approaches is representative of a larger family of algorithms. The first approach is currently used by Microsoft Azure for scheduling their BVMs and is similar to the predictor used by Borg [28, 29]. This approach essentially decides on a fixed factor and oversubscribes CPU resources by that factor. The other two approaches are approaches that have been used in different contexts and that we adapted and optimized for the use with BVMs. One is based on the approach used by Microsoft Azure for their regular VMs [7]. The other one is based on the general idea of estimating a server's aggregate CPU usage using the Central Limit Theorem (which has been successfully used in other resource allocation problems [4, 8, 14]), but with a few optimizations that we add based on our observations in Section 3.

.

4.1 Scheduling BVMs based on fixed oversubscription ratios

This heuristic is currently being used by Microsoft Azure to schedule BVMs and is similar to the predictor used by Borg [28, 29]. The scheduler takes as input parameter (specified by the operator) a coefficient and reserves for each BVM its baseline multiplied by the coefficient. When placing a new VM, a candidate server is considered valid only if the coefficient multiplied with the sum of the baselines of BVMs already allocated on the server and the new BVM is not larger than the number of cores on the server.

The advantage of this approach is its simplicity as it relies on only one parameter (the oversubscription coefficient) and requires only information that is part of the BVM configuration (its baseline), rather than measurement data.

The downside of this approach is that it does not allow a provider to directly specify a threshold on server capacity violations. Instead the provider indirectly controls server capacity violations through the choice of the coefficient. A more conservative coefficient reduces server capacity violations, but also limits achievable server utilization.

We experiment with two versions of this approach:

Microsoft Azure baseline: In this version we set the coefficient to 2, as this is the value currently used at Microsoft Azure.

Oversubscription Oracle: In this version we (unrealistically) assume a clairvoyant oracle that chooses for a given threshold on server capacity violations and a given workload (trace) the smallest coefficient that ensures capacity violations below the threshold. We implement this oracle-aided approach by applying binary search for a given trace to identify the smallest coefficient that keeps server capacity violations just below the specified threshold. While this oracle-aided approach is clearly not achievable in practice it enables us to establish an upper bound for any outcome achievable with the general approach of fixed oversubscription ratio.

4.2 A baseline scheduler from the world of regular VMs: Adapting Resource Central

The other baseline approach is an adaptation of the scheduling algorithm employed by Microsoft Azure for regular VMs, as described in two existing papers [7, 12]. At a high-level, its placement decisions rely on an ML engine, called Resource Central [7], which predicts (among other things) the 95th-percentile of CPU utilization of the VM to be placed. A candidate server is considered *valid* only if the sum of the 95th-percentile predictions of VMs already allocated and the new VM are less than the server capacity.

We have optimized the original approach described in [7] for the BVM environment based on extensive experiments as follows. Instead of predicting the 95th-percentile of CPU utilization as the upper bound of one of four buckets (0–25%, 25–50%, etc.) our implementation utilizes eight more fine-grained buckets. Each BVM configuration is divided into four equally sized buckets ranging from 0 to the baseline value, and another four equally sized buckets between the baseline and peak CPU utilization. In addition, we equip this approach with an oracle that provides perfect predictions of which bucket a VM falls into. While this is clearly not achievable in practice it provides us the best possible results that can be achieved with this method (without being limited by prediction accuracy). We refer to this method as **RC**.

4.3 Adapting CLT-based methods for scheduling BVMs

The central limit theorem (CLT) tells us that the sum of n i.i.d. random variables approaches a Gaussian distribution with mean and variance equal to the sum of the means and variances of the n variables as n approaches infinity. Unlike the two baseline heuristics we presented, an approach based on the CLT allows us to directly determine whether a server is a valid candidate as follows:

Assuming we have an estimate of the mean and variance of the VMs running on a server and the mean and variance of an arriving VM we can use the Gaussian distribution to determine the probability that their combined CPU usage will exceed server capacity and create a violation. If that probability is below the specified violation threshold the server is a valid candidate. The following equation shows this probability. In this equation S_n is a random variable representing the aggregate CPU usage and C is the capacity of the server. This equation shows that we can calculate the mean and variance of the S_n by summing the mean and variances of the n active BVMs on this server. ϵ is the limit on the rate of server capacity violations.

$$P(S_n > C) < \epsilon \quad \text{where} \quad \mu_{S_n} = \sum_{i=1}^{n} \mu_i \quad \text{and} \quad \sigma^2_{S_n} = \sum_{i=1}^{n} \sigma^2_i$$

While not all the assumptions the CLT relies on might hold in practice, previous work [4, 8, 14] has found approaches based on the CLT to be effective in a different, but related context, for co-locating long-running data center tasks while limiting capacity violations.

Based on extensive experimentation we designed an approach for scheduling BVMs using the CLT as follows.

First, we find that it is important to use a conservative estimate for the mean and variance of a new incoming BVM, because of Lesson 3 in Section 3: resource utilization is higher early in a BVM's life. We determine this conservative estimate by computing the 95th percentile of the mean and variance of BVMs with the same configuration based on historical data.

Second, after a VM has been running for some time, i.e. its resource usage has stabilized and we have actual utilization data for this BVM, we update the estimate of its mean and variance based on observed usage data. We use the first 6 hours of usage data for a conservative estimate. We refer to this method simply as **CLT**.

5 Audible: Adaptive Utilization-driven Burstable VM Placement

This section presents Audible, a novel algorithm for Adaptive Utilization-driven Burstable VM Placement. We begin by detailing Audible's key design ideas, which are based on the lessons learned in Section 3 and the use of non-parametric statistics. Next we describe the algorithm's workflow in more detail and within the context of a BVM scheduler. We then discuss the data collection processes that support Audible's functionality. Lastly, we examine the adaptations required to integrate Audible into existing scheduler frameworks.

5.1 Audible Design: Insights from BVM Workload Analysis

Our design of Audible is guided by several of the lessons presented in Section 3. First, based on Lesson 5 we are wary of relying on CLT-based estimates of the aggregate CPU utilization as they risk underestimating the tail of the aggregate demand distribution. Second, based on Lesson 3 and 4 we are steering clear of the one-size-fits-all approach of a fixed oversubscription ratio, as BVM workloads are highly variable over time and across BVMs. Instead we rely on the observed empirical aggregate CPU utilization distribution on each server. Finally, based on Lesson 3 we make conservative assumptions about the CPU demands of arriving BVMs.

At a high level, when deciding whether a server is a valid candidates Audible approximates the distribution D_{total} of the aggregate CPU demand of the BVMs currently running on the server and the distribution D_{new} of the CPU demands of the incoming BVM to be placed. It then estimates the distribution of aggregate CPU demands that would result from placing the new VM on this server by convolution of D_{new} and D_{total} (reflecting the sum of the two corresponding random variables).

Audible approximates D_{total}, i.e. the aggregate CPU demand of currently running BVMs, using the empirical distribution of the server's CPU measurements over the last 24 hours that is recorded at 5-minute intervals at each server at Microsoft Azure.

Audible approximates D_{new} based on historical data for BVMs of the same configuration (same baseline and peak parameters). Particularly, we used the empirical CPU demand distribution of the 95th percentile highest demanding BVMs from the same configuration, when CPU demands are ranked based on the sum of their mean and standard deviation.

Based on the above approximations of D_{total} and D_{new}, Audible estimates the probability of server capacity violations if the new BVM were to be added to the server based on the joint demand distribution of the new BVM and the aggregate demands of all the currently running BVMs using the convolution of D_{total} and D_{new}. Audible deems a server a valid server only if the estimated probability of capacity violation is below a predefined acceptable threshold ϵ on violations.

In other words if the following condition is satisfied.

$$P(X > C) < \epsilon, \text{ where } X \sim (D_{\text{total}} * D_{\text{new}})$$
$$\Leftrightarrow \int_C^{\infty} (D_{\text{total}} * D_{\text{new}})\,(x)\,dx < \epsilon$$

To implement Audible each server keeps track of the CPU utilization measurements over the past 24 hours. Assuming a measurement interval of 5 minutes (as used by Microsoft Azure) this corresponds to 288 data points. The server periodically precomputes the convolution for each BVM type and the recent window of CPU measurements to determine the maximum BVM type it could host. Computing the convolution on this limited number of data points poses minimal overheads. The information of the largest BVM type that the server can accept could then be relayed to the scheduler, e.g. by piggy-backing on the regular heartbeat messages.

5.2 Audible Workflow

Algorithm 1 outlines the core components of Audible's workflow for allocating BVMs.

In the offline phase, the algorithm maintains a conservative estimate of the CPU usage distribution—or Probability Mass Function (PMF)—for each BVM configuration by analyzing historical CPU usage data for BVMs of each configuration (available in a database called *BVMConfigsUsage*), as described in Section 5.1. The PMFs for all configurations are stored in a hash table referred to as *confPMFs* in the pseudo code.

In the online phase, the algorithm schedules a set of BVMs by obtaining for each BVM a list of candidate servers through *getCandidateServers* (provided by accessing a separate scheduler module that considers for example hardware requirements of the BVM and load balancing heuristics, see for example [12]) and then using Audible's logic to determine for each candidate server whether it is a *valid* candidate for the BVM: the algorithm computes the convolution of the estimated conservative PMF of the BVM (obtained from *confPMFs* through *getConfPMF*) and the PMF of the server (obtained from *ServerUsage* through *getServerPMF*). The algorithm places the BVM on the first server where the chance that the combined load of server and new BVM exceeds server capacity is below the specified threshold. In the event that none of the servers in the list of candidate servers are capable of hosting the BVM, the system will reroute the BVM to a different cluster.

Algorithm 1 Audible Scheduling workflow

```
Require: $BVMConfigsUsage$: Database of historical data on CPU
    usage for each BVM configuration. $ServerUsage$: Database of
    most recently reported CPU usages for each server
    Offline part
 1: Initialize $confPMFs \leftarrow []$
 2: for each $group$ in $BVMConfigsUsage$ do
 3:     $sorted \leftarrow sort(group)$           ▷ sort based on std+avg
 4:     $temp \leftarrow getTopPercent(sorted, 5)$
 5:     $confPMFs.append(createPMF(temp))$
 6: end for
    Online part
 7: for $bvm$ in $newBVMs$ do
 8:     $bvmPMF \leftarrow getConfPMF(bvm, confPMF)$
 9:     $servers \leftarrow getCandidateServers(bvm)$
10:     for $server$ in $servers$ do
11:         $serverPMF \leftarrow getServerPMF(server, ServerUsage)$
12:         $jointPMF \leftarrow convolve(serverPMF, bvmPMF)$
13:         if $checkLoad(jointPMF) < threshold$ then
14:             $placeVM(bvm, server)$
15:             $updateStatus(server)$
16:             break           ▷ BVM placed, move to next BVM
17:         end if
18:         $resubmit(bvm)$     ▷ if not placed, resubmit to different
    cluster
19:     end for
20: end for
```

5.3 Sourcing Data for Audible

Audible depends on two main data sources. The first data source tracks historical CPU usage across all BVM configurations to create a detailed distribution table for each configuration. This is the information that Audible accesses through the *createPMF* method. Collecting this data requires a monitoring system capable of recording and updating the CPU usage distribution for each BVM configuration according the historical data. This process can be performed offline. The second required data source involves the real-time CPU usage distribution for each server, identified in the pseudo code as *serverPMF*. This data can be gathered by utilizing the regular heartbeat signals emitted by servers. Both types of data are data that are commonly collected and stored by cloud platform management systems.

5.4 System Modifications for Audible Integration

The following modifications to the allocator are necessary for incorporating Audible into Azure's existing framework [12]. The allocator should designate a server as "ready" for VM allocation only if its status has been recently updated, including the real-time CPU distribution data, thus allowing Audible to access the server's recent CPU usage distribution. Moreover, the allocator should be able to access the CPU usage distribution for each BVM configuration. Once Audible finalizes its analysis, its decisions are incorporated into the allocator rules. An example rule might state, "Server x can accommodate BVM configurations with baseline usage up to 1.35". It is crucial to recognize that while these guidelines are informed by Audible's analysis for CPU oversubscription, they are applied in conjunction with other criteria that consider other resources like storage and network bandwidth during the allocation process.

Complexity analysis: Evaluating Audible's deployability necessitates understanding the complexities it introduces, both in its offline and online parts. In the offline part, the objective is to refine CPU usage data for each BVM configuration through the analysis of the highest 5% CPU usage patterns from past records. This task requires calculating the CPU usage distribution among a representative batch of historical BVMs, where both computational and space complexity depends on the size of data for the historical BVMs. We saw in our experiments that this size can be relatively be small, in the order of a few hundreds of BVMs.

In the online part, the major computational effort for Audible comes from the convolution operation, with a complexity of O(n^2), where n indicates the precision level in calculating a VM's CPU usage distribution, equating to 100C bins for a BVM possessing a peak CPU capability of C. The online space complexity is limited to recording the CPU usage distribution for each server, requiring 100S bins for a server equipped with S cores, to capture full representation of server CPU usage distribution. Collectively, these complexity considerations demonstrate practicality of Audible, and substantiate its viability for production implementation.

6 Experimental Setup

6.1 The trace-driven simulator

We implemented an event-driven simulator in Python that replays VM arrivals based on our VM traces. We implement in our simulator a version of each of the algorithms described in Section 4. The simulated allocator uses these algorithms to determine whether a server is a *valid* candidate for an arriving VM, i.e. whether the VM can be placed on the server while staying within the specified limit for violations. In our experiments we use the 2021 VM dataset, since it is more recent, and we experiment with servers that have either 36, 48 or 64 cores. Servers with 48 cores are currently the most common server configuration at Microsoft Azure. We parameterize the algorithms with threshold of 0.5%, 1%, 2.5% and 5% on the rate of server capacity violations.

Our simulator also needs to support policies for choosing which server of the pool of *valid* candidate servers that was identified by our methods above to place a VM on. These policies are orthogonal to our policies for identifying valid servers and we have implemented in our simulator different standard bin packing methods, including best-fit, worst-fit and random. All results in this paper are based on a policy that selects a server from the least busy 10% of valid servers currently available.

If at any time the aggregate CPU demands of the BVMs placed on a server exceed server capacity, we conservatively assume that the excess demand accumulates and the backlog is being executed once there is sufficient server capacity available (rather than shedding the excess demand). The local resource manager on a server would decide in such a situation, which VMs get throttled. In practice, different policies exist, for example based on fairness considerations. The metrics we report in this work are not affected by the policy used for deciding which VM to throttle and as such those policies are orthogonal to our work.

6.2 Reaching Steady State

A key aspect of any simulation setup is to ensure that the simulation has reached steady state at the point when measurements are being taken, i.e. results are not being influenced by transient behavior during the warm-up period of the simulator. We found this aspect to be particularly critical when simulating BVM cluster schedulers. As we have observed in our analysis of field data in Section 3 (Lesson 5), at any point in time about 90–98% of all currently-running BVMs are long-running BVMs despite the fact that the majority of the arriving VMs are short-running VMs. We also observe that long running and short running VMs differ in their resource usage characteristics, so it is important to accurately reflect this ratio in simulation. When starting a simulation, initially the VMs running will be mostly short-running VMs, as they make up the majority of VM arrivals. In all our experiments we make sure that the simulation runs long enough to reach steady state (where around 94% of running BVMs are long-running) before starting to take measurements. We are emphasizing this point as a word of caution for others performing research in this area as we found that results can be significantly different when the system is not carefully warmed up.

6.3 Performance Metrics

Each experiment consists of simulating 10 months of VM arrivals to reach the steady state (as explained in section 6.2), and recording the following metrics over the last week when steady state has been reached:
Server utilization: We record for each server the average CPU utilization in the *steady state*.
Server capacity violation Rate: For each server, the fraction of all *steady state* time-periods with a server capacity violation (BVM CPU demand exceeded server capacity).
Violation magnitude: The average CPU shortage value for each server at every time step throughout the *steady state*.

For each algorithm we run trace-based simulations with different arrival rates and determine the maximum VM arrival rate that each algorithm can sustain without VMs getting rejected (because no valid candidate server can be found). Once the maximum arrival rate is reached and VMs get rejected the scheduler will determine that the cluster is at capacity and stop routing VMs to that cluster. Therefore the cluster utilization during the maximum arrival rate is the highest utilization the cluster can achieve.

7 Evaluation

In this section, we are asking two questions: (1) which of the algorithms can reliably enforce limits on the rate of server

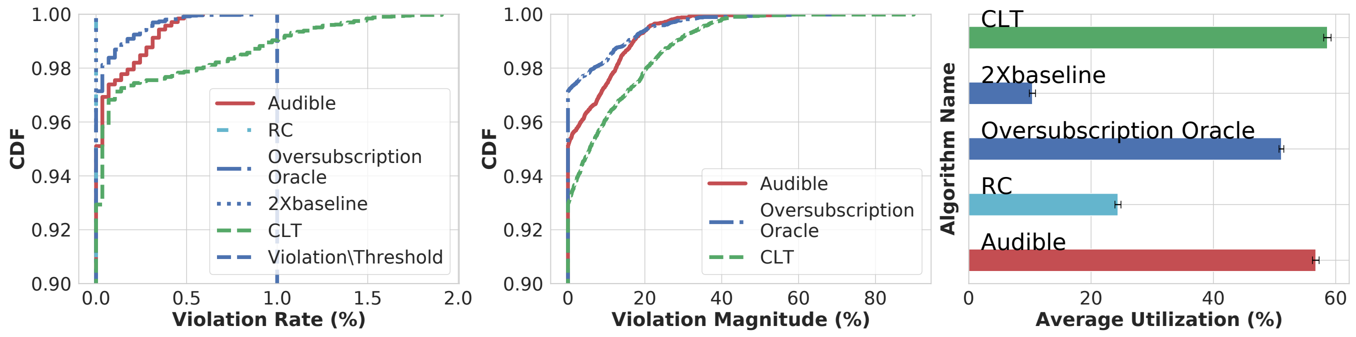

Figure 8. *Comparing server capacity violations and utilization for different algorithms for 48-core servers and a 1% threshold on violations. (Microsoft Azure-baseline and RC had no servers with capacity violations and hence don't appear in the violation plots.)*

capacity violations and (2) what is the utilization each algorithm can achieve. Our evaluation compares the five algorithms we introduced in Section 4: Microsoft Azure-baseline, Oversubscription-Oracle, RC, CLT, and Audible.

7.1 Baseline Experiments

In this section, we consider an experimental setup with 48-core servers (currently the most common server configuration at Microsoft Azure) and set a limit of 1% on the rate of server capacity violations. Figure 8 shows the results.

Server capacity violations The left-most graph shows the CDF of the server capacity violation rates across servers in the cluster for the different algorithms. We observe that the two approaches used in practice, Microsoft Azure-baseline and RC, are very conservative and as a result none of the servers ever experiences any capacity violations under them (and hence there is no corresponding line in the graph).

We also observe that under Audible no server ever comes close to reaching the 1% limit on capacity violations. The highest violation rate of any server is 0.45%, i.e. less than half the allowed limit. In contrast, under CLT a significant number of servers (2.5%) experience capacity violations above the threshold. The worst server under CLT sees a capacity violation rate of 1.9%. We note that these experiments are based on one-week periods. So over a longer time period even more servers under CLT are likely to violate the limit on capacity violations.

(Servers under Oversubscription-Oracle by definition do not exceed the violation threshold as the Oracle sets its parameters such that the threshold is always met.)

Server Utilization We make several interesting observations when looking at the server CPU utilization achieved by different approaches (Figure 8 (right)). First, the utilization achieved by Audible is slightly *higher* than that of even the Oversubscription-Oracle. This is despite the fact that Oversubscription-Oracle has the unfair advantage of clairvoyance and does experience higher violation rates (albeit by definition none above the limit). This shows that an approach based on a fixed oversubscription ratio, even when that ratio is optimally chosen, is not flexible enough to allow for maximum utilization. In contrast, Audible takes the recent behavior of BVMs running on a server into account when making placement decisions, while also reliably maintaining the limit on server violations (unlike CLT). Audible's utilization is also only slightly lower than CLT, which did violate the limit on server violations to achieve its high utilization. Due to their overly conservative nature the utilization under Microsoft Azure-baseline and RC is very low.

7.2 Exploration of other server configurations and violation thresholds

Figure 9 provides a summary of our results. Each column in Figure 9 corresponds to a different violation threshold, ranging from 0.5% to 5%. Each graph shows result for three different server configurations, 36, 48, and 64 cores. We omit results for Microsoft Azure-baseline and RC as they were not competitive with respect to utilization. For each unique experiment setting—defined by the algorithm, violation threshold, and server capacity—10 sets of experiments were conducted to capture the variation in the results, which is represented by the range bar on top of each bar in the figure.

Ability to enforce limits on capacity violations We first focus on the ability of the three algorithms to enforce the limit on server capacity violations. The second row of graphs in Figure 9 shows the fraction of servers that exceeded the limit on server capacity violations. We observe that Audible is able to keep server capacity violations below the required threshold for all of the 12 settings (the percentage of servers that exceed the violation limit is zero for all settings, hence no bars in the figure). In contrast, CLT leads to excessive capacity violations in 7 of the 12 scenarios CLT. Only when we relax the limit on the capacity violations to 5%, CLT is consistently able to enforce the limit on violations.

CLT struggles particularly for smaller server sizes. For example, for a 36-core server there are consistently servers that experience a violation rate close to 4%, independent of the target for the violation rate (see row 4 in the figure). Smaller

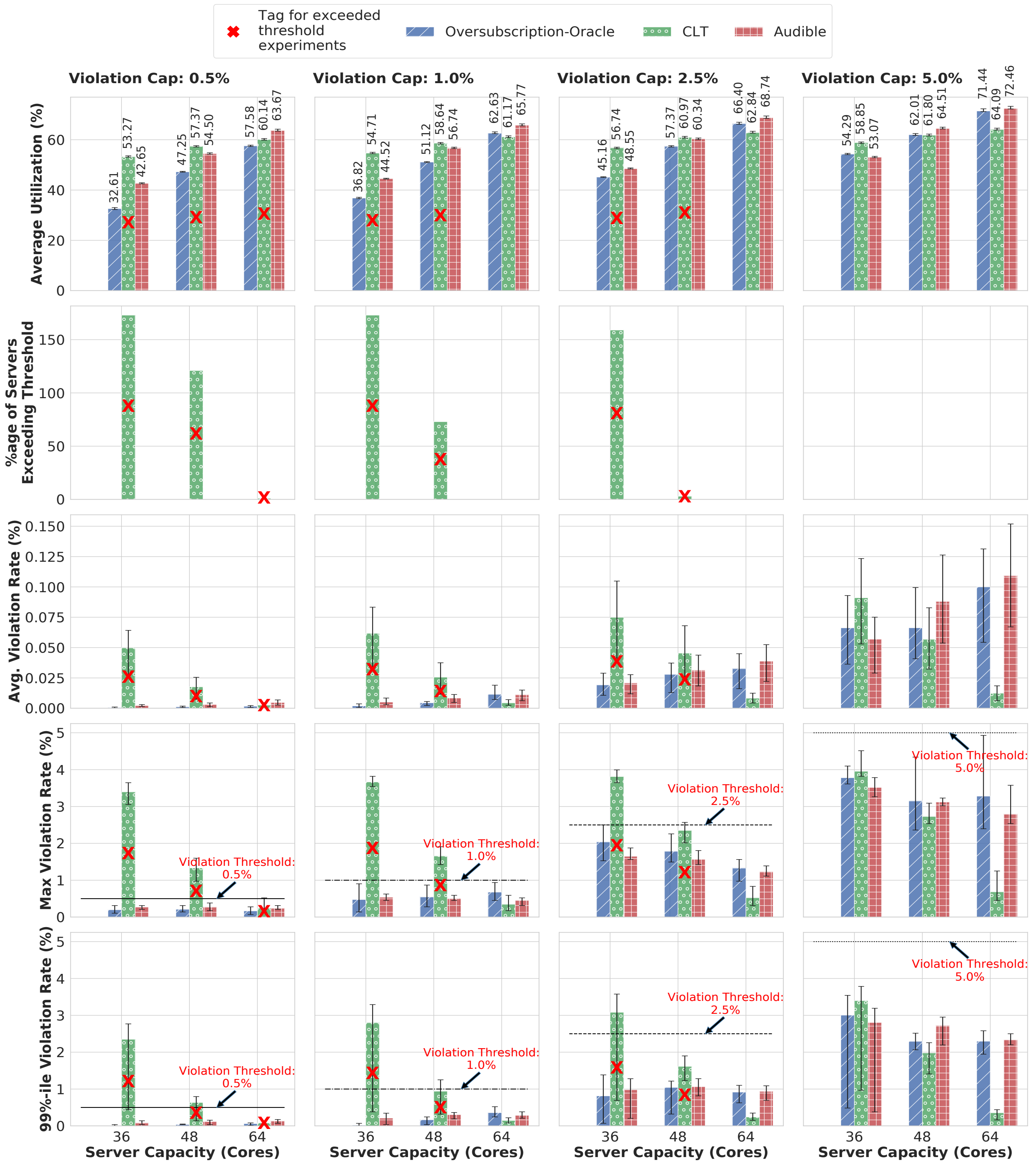

Figure 9. *Comparing server capacity violations metrics and utilization for for different server configurations and violation thresholds.*

servers can host fewer VMs and multiplexing resource usage over a smaller number of VMs increases the chance that estimates based on the central limit theorem are not accurate. CLT also struggles for more stringent thresholds on the capacity violations. For thresholds of 0.5% CLT cannot meet targets for any of the server sizes. The reason is likely that at low thresholds it becomes more important to estimate the tail of the aggregate resource distribution accurately. We have seen in Lesson 5 that the Gaussian distribution struggles to capture the tail of BVM resource usage.

The Oversubscription-Oracle experiences no servers that exceed violation threshold, since the clairvoyant oracle chooses the oversubscription ratio for each scenario such that no server experiences violations above the threshold. When looking at the server with the highest violation rate (see the fourth row of the graphs in Figure 9), we see that under the Oversubscription-Oracle the violation rate of this server is quite close to the specified threshold. In comparison, Audible maintains a safe margin from the threshold, with the worst server seeing violations rates of around 50% of the threshold. Also the average violation rate (see Row 3 in Figure 9) is lower under Audible than under Oversubscription-Oracle.

Ability to maximize utilization We begin with a comparison of the utilization under the Oversubscription-Oracle versus Audible. Interestingly, we observe that, across all settings, Audible is able to achieve *higher* utilization levels than the Oversubscription-Oracle. This is despite the fact that the Oversubscription-Oracle (unrealistically) works with perfect a-priori knowledge of the optimal oversubscription ratio for each of the settings, i.e. the ratio that maximizes utilization while still meeting capacity violation targets.

The fact that Audible beats the Oversubscription-Oracle consistently with respect to utilization is particularly impressive given that Audible keeps violations rates quite low compared to the Oversubscription-Oracle. Intuitively, Audible needs to keep a safety margin on violation rates as it needs to deal with statistical uncertainty, while the Oversubscription-Oracle is clairvoyant and can afford to push the violations and therefore its utilization to the limit.

When comparing the utilization of CLT and Audible we only consider the 5 settings where CLT was actually able to meet violation thresholds. (Setting where CLT did not meet the threshold are marked with a red star and are excluded for a fair comparison.) We observe that Audible achieves higher utilization than CLT in 3 of the 5 settings.

BVM quality of experience We also looked at the impact on BVM quality of experience on servers that experienced capacity violations. Table 2 shows what percentage of VMs active on a server during a capacity violation are still able to achieve at least 4 times their baseline CPU, even though they are not able to achieve their peak. Under Audible the percentage of VMs that is still able to burst to at least $4X$ their baseline during a server capacity violation is higher than for CLT and Oversubscription-Oracle.

Algorithm	36 cores	48 cores	64 cores
Oversubscription-Oracle	99.98%	99.76%	99.75%
CLT	98.04%	99.14%	99.74%
Audible	100%	100%	99.91%

Table 2. *Average percentage of active VMs capable of bursting up to 4 times their baseline across various algorithms.*

7.3 Summary of Evaluation Results

Audible consistently achieves higher utilization than other methods while also meeting stringent violation thresholds. Compared to approaches currently used in practice, Audible increases cluster utilization by more than a factor of 2.5.

In contrast, approaches based on a fixed oversubscription ratio barely achieve utilization levels comparable to Audible, even in their (unrealistic) best case scenario with perfect knowledge of the optimal oversubscription ratio. We conclude that approaches based on a fixed oversubscription ratio, even when that ratio is optimally chosen, are not flexible enough to allow for maximum utilization.

Another standard approach, relying on the CLT for approximating aggregate server utilization, is not consistently able to enforce threshold on server capacity violations. We will look into reasons for this shortcoming in the next section. In addition, CLT does also not result in higher utilization than Audible. Finally, in our design of CLT-based approaches we found that they are more sensitive to design and implementation choices. In combination, this makes CLT-based approaches less attractive than Audible.

8 Lessons learned

Can we pinpoint the underlying causes of violations?

We are interested in identifying the key factors contributing to the CPU bursts involved in server capacity violations. When closely examining the server state during capacity violations, we observe that 40-43% of the observed bursts originate from larger VMs (with more than 8 cores). In essence, a marginal subset of the overall active VMs, constituting around 1-2% of all VMs (VMs with 12, 16, or 20 cores), causes considerable performance degradation during bursting events. The reason is that when these large VMs burst they have the potential to dominate overall server utilization. In BVM workloads, which have a higher tendency towards burstiness (recall Lesson 4), bursts of large VMs create risk of server capacity violations. While Audible is able to capture this risk, the CLT approach is not.

Why did CLT fail?

From a theoretical perspective, satisfying the Central Limit Theorem (CLT) for an aggregate random variable involves fulfilling two sets of conditions regarding its random components. The first condition pertains to probabilistic independence of co-located VMs. A scheduler could try to increase independence, e.g. by avoiding co-location of VMs from the same user, however diurnal patterns in workload might still create correlations. The second condition relates to either the random components being identically distributed, or none of the components have a significantly large variance [9]. In practice obviously the random components are not identically distributed due to the variations in configuration and workload among co-located instances. In the case of BVMs, we observed that the variance condition is also not satisfied mainly due to CPU usage bursts from large BVMs.

To demonstrate and validate that the variance condition is primarily influenced by CPU usage bursts from large BVMs, leading to an inadequate Gaussian approximation of the aggregate, we test the Gaussian approximation hypothesis for the aggregate distribution twice. First, for a randomly selected sets of BVM components, and second, for the same sets with the CPU usage bursts of large BVMs limited to their baseline. In each case, we calculate the P-value for the samples from the 99th percentile tail of the distribution. Remarkably, the P-values unequivocally support the earlier discussion, as the aggregate without bounded CPU usage bursts from large BVMs exhibits a notably low P-value.

Also, recall that the probability distribution the aggregate CPU usage for a server exhibits a pronounced right skew, i.e. the tail extends beyond standard normal distribution limits (Lesson 5). For BVM schedulers, the lesson is clear: the consideration of non-Gaussian behavior in the tail distribution of aggregate CPU usage, especially during bursts, is critical.

Attempts at adapting the CLT approach: We performed extensive experiments trying to improve CLT's ability to meet violation thresholds. These attempts included, for example, the use of a "black-list", where a server is removed from consideration as a valid server (black-listed) for some period of time once it starts experiencing capacity violations; using more conservative initial estimates; obtaining more accurate estimates of mean and variance, steering large VMs away from servers that are already hosting other large VMs, adding a safety margin around large VMs, among others. In the end we found that (1) these fixes reduce the utilization that CLT can achieve, and/or are still limited in their ability to control server capacity violations and (2) that it is somewhat of a black art to find the right combination of fixes that will make CLT work for a given configuration. Compared to Audible which consistently works without tuning of any parameters and achieves high utilization the CLT approach seems unattractive in comparison.

Experiences with alternative approaches: We studied alternative approaches that seem to be a natural fit for scheduling BVMs, but in the end did not perform well and were therefore not included in our results. For the benefit of others working in this area we briefly summarize these "negative results" below:

- **Dual Token Bucket Models for BVM Scheduling:** One idea we explored is the use of Dual Token Bucket (DTB) models, inspired by network calculus, as BVMs can be represented as combinations of two token buckets. The challenge is to set bounds on cumulative CPU demands for multiple BVMs on a server. We considered two approaches: a conservative deterministic bound and a stochastic bound using Gaussian processes [10]. However, the deterministic bound proves overly cautious, assuming maximal CPU usage, leading to very low utilization. While the stochastic bound achieves higher utilization, around 10%, it remains low as it fails to account for empirical BVM behavior. Real-world observations show that BVMs, especially long-running ones, vary significantly. Despite offering a tighter bound, the analytical stochastic approach leads to very low utilization. In conclusion, the DTB model's dynamic resource assignment doesn't align well with BVMs' actual CPU usage, rendering their configuration inadequate as an indicator.
- **Configuration-Based Probabilistic Modeling:** Another idea we explored was to adapt the models of BVM resource usage that Jiang et al. [15] developed for a different purpose. Their specific models made some assumptions on CPU usage that we found to be unrealistic. Instead we tried to populate their models based on empirical CPU distributions. However, despite our best efforts the algorithms based on these models resulted in very high rates (25%) of server capacity violations. As observed earlier in this section, we found that also here large VMs emerged as the primary contributors to the violations. Another issue was the approach's struggle to adapt to variations in CPU usage distribution over a BVM's lifetime, particularly between short-running and long-running instances, leading to underestimated total CPU usage and frequent server resource violations.

9 Ongoing work: Generalizing Audible

While we designed Audible specifically for BVMs, we note that at its core Audible's approach is broadly applicable: its non-parametric method of using the convolution of observed empirical distributions means it makes no specific assumptions about the underlying workloads and their distributions. An obvious question is therefore whether Audible can also effectively oversubscribe regular virtual machines, rather than BVMs.

We have been exploring this question in on-going work and have made some promising observations. We find that Audible can greatly improve server utilization in data centers running regular VMs over existing approaches. However, for regular VM workloads we observe some rare cases (one in a thousand servers) where servers experience capacity violations above the specified threshold. We identified as the key reason the bigger (negative) impact that large VMs, i.e. VM types with large core counts, have when scheduling regular VM workloads. Large VMs are generally harder to handle in oversubscription scenarios than small VMs as one (or a few) large VMs have a higher chance of dominating the workload of a server than individual smaller VMs. This effect is exacerbated for regular VM workloads where large VMs are relatively more frequent (recall Figure 1) than for BVM workloads, and are also more likely to utilize all their cores simultaneously (something BVMs can only do when they have accumulated tokens). We're currently working on a generalization of Audible that uses a decay factor to keep some memory of the original conservative estimate of the violation probability for a newly arrived large VM. Initial results indicate that this simple extension of Audible can successfully enforce limits on server capacity violations and we expect it to still achieve high server utilization. Full results will be presented in an extended publication.

10 Related work

We are not aware of prior work on policies for data center resource allocation for BVMs. The BVM work that is closest is by Jiang et al. [15]. The focus of their work is different, as they aim to provide models that can be used for cloud provider revenue maximization. But aspects of their model could be used to determine the risk of server capacity violations for VM placement. We attempted to adapt their ideas for this purpose as described in Section 8, without success.

There are several papers that look at burstable VMs from the point of view of how applications can make use of BVMs. This includes, for example, work on how to use BVMs for auto scaling [3]; for in memory caches [30, 31]; for running bags of tasks (on a combination of burstable and spot instances) [27]; how to use burstable instances in mobile computing [25]; how to make distributed data processing frameworks aware of credits to run more efficiently inside burstable VMs [23]; how to control an application's resource usage, e.g. through throttling, so they make optimal use of burstable resources [2]; and how storage applications can make use of cheap IOPS through burstable storage [19].

There are also papers on choosing the most suitable BVM configurations for a given workload using application performance predictors [1, 20]. This work is complimentary to our work (which assumes a customer has already chosen the type of BVM for their workload).

When considering general work for regular VMs, there are a number of studies that focus on CPU usage predictions of individual VMs rather than server aggregates. For example, PRESS [11], CloudScale [24] and AGILE [18] predict usage over short time horizons in order to make decisions on VM migrations, or adjusting allocated resources accordingly. There are also a number of studies [5, 13, 17, 21, 22, 26] on resource prediction that use more heavy-weight statistical methods, e.g. time series-based analysis. Those approaches are more expensive in the amount of data and computation they require and not suitable for our problem setting.

The work on resource prediction that is closest to ours work is a recent paper by Bashir et al. [4]. Bashir et al. are interested in oversubscription of server resources through predictions of aggregate server utilization for machines in Google's data centers. They build a predictor that utilizes predictions from three methods: RC-based estimates, CLT-based estimates and fixed oversubscription ratio. We find that these methods do not work well for BVM workloads, likely due to differences in workload characteristics (recall Section 3) and the more stringent limits on violations.

11 Conclusion

We provide the first study of resource allocation for BVMs, including the first study of BVM workloads on production machines at a major cloud provider. We derive lessons to guide resource allocation for BVMs and identify some key differences to regular VM workloads. We use those insights to design Audible, a new approach for scheduling BVMs based on non-parametric statistics, that is light-weight and workload independent, and obviates the need for training of ML models or tuning magic parameters. We show through extensive simulations driven by production traces that Audible achieves higher utilization than state-of-the art approaches while being able to enforce stringent requirements on server capacity violations. Given its effectiveness and its attractive features, it will be worth exploring the use of Audible in other resource management settings, such as regular VMs or general data center tasks.

As part of this paper, we are making available an artefact, which will allow others to reproduce and build on top of our work and which will hopefully foster research in this area. The accompanying artifact consists of two significant datasets and a simulator specifically designed to assess the "Audible" VM scheduling technique highlighted in our investigation. The datasets include two main collections of burstable VMs data: one from 2019, which encapsulates information from hundreds of thousands of VMs over a three-day period, and another from 2021 that documents the activities of millions of VMs across an entire week. Additionally, to support the replication of our study's findings, the entire Python codebase used in our research is made available. The codebase is accessible on GitHub at https://github.com/seyedali14/audible-artifact-asplos24. This complete package of datasets and simulation tools will hopefully provide a helpful foundation for further research on VM scheduling.

References

[1] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. CEDULE: A scheduling framework for burstable performance in cloud computing. In *IEEE International Conference on Autonomic Computing (ICAC)*,

pages 141–150. IEEE, 2018.

[2] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. It's not a sprint, it's a marathon: Stretching multi-resource burstable performance in public clouds. In Dejan S. Milojicic and Vinod Muthusamy, editors, *Proceedings of the 20th International Middleware Conference Industrial Track*, pages 36–42. ACM, 2019.

[3] Ataollah Fatahi Baarzi, Timothy Zhu, and Bhuvan Urgaonkar. Burscale: Using burstable instances for cost-effective autoscaling in the public cloud. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019*, pages 126–138. ACM, 2019.

[4] Noman Bashir, Nan Deng, Krzysztof Rzadca, David Irwin, Sree Kodak, and Rohit Jnagal. Take it to the limit: peak prediction-driven resource overcommitment in datacenters. In *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys)*, pages 556–573, 2021.

[5] R.N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. Workload prediction using arima model and its impact on cloud applications' QoS. *IEEE Transactions on Cloud Computing*, 2014.

[6] Maxime C. Cohen, Philipp Keller, Vahab Mirrokni, and Morteza Zadimoghaddam. Overcommitment in cloud services - bin packing with chance constraints. *Management Science*, 2019.

[7] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles (SoSP)*, pages 153–167. ACM, 2017.

[8] Nan Deng, Zichen Xu, Christopher Stewart, and Xiaorui Wang. Telltale tails: Decomposing response times for live internet services. In *Sixth International Green and Sustainable Computing Conference, IGSC*, pages 1–8. IEEE Computer Society, 2015.

[9] Rick Durrett. *Probability: Theory and Examples*. Cambridge, 4 edition, 2010.

[10] Paolo Giacomazzi, Luigi Musumeci, Gabriella Saddemi, and Giacomo Verticale. Analytical methods for resource allocation and admission control with dual-leaky-bucket regulated traffic. In *Proceedings of IEEE International Conference on Communications ICC*, pages 499–505. IEEE, 2007.

[11] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Proceedings of IEEE International Conference on Network and Service Management*, 2010.

[12] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E. Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. Protean: VM allocation service at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI*, pages 845–861. USENIX Association, 2020.

[13] S. Islam, J. Keung, K. Lee, and A. Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 2012.

[14] Pawel Janus and Krzysztof Rzadca. SLO-aware colocation of data center tasks based on instantaneous processor requirements. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC)*, pages 256–268. ACM, 2017.

[15] Yuxuan Jiang, Mohammad Shahrad, David Wentzlaff, Danny H. K. Tsang, and Carlee Joe-Wong. Burstable instances for clouds: Performance modeling, equilibrium analysis, and revenue maximization. In *2019 IEEE Conference on Computer Communications (INFOCOM)*, pages 1576–1584. IEEE, 2019.

[16] SeyedAli Jokar Jandaghi, Kaveh Mahdaviani, and Cristiana Amza. Virtual instance resource usage modeling: A method for efficient resource provisioning in the cloud. In *Proceedings of IFIP/IEEE IM 2017 Workshop: 2nd International Workshop on Analytics for Network and Service Management (AnNet)*, pages 917–922, 2017.

[17] A. Khan, X. Yan, S. Tao, and N. Anerousis. Workload characterization and prediction in the cloud: A multiple time series approach. In *Proceedings of IEEE Network Operations and Management Symposium*, 2012.

[18] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. AGILE: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proceedings of International Conference on Autonomic Computing (ICAC)*, 2013.

[19] Hojin Park, Gregory R. Ganger, and George Amvrosiadis. More IOPS for less: Exploiting burstable storage in public clouds. In Amar Phanishayee and Ryan Stutsman, editors, *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. USENIX Association, 2020.

[20] Riccardo Pinciroli, Ahsan Ali, Feng Yan, and Evgenia Smirni. CEDULE+: resource management for burstable cloud instances using predictive analytics. *IEEE Transactions on Network and Service Management*, 18(1):945–957, 2021.

[21] Olga Poppe, Tayo Amuneke, Dalitso Banda, Aritra De, Ari Green, Manon Knoertzer, Ehi Nosakhare, Karthik Rajendran, Deepak Shankargouda, Meina Wang, et al. Seagull: An infrastructure for load prediction and optimized resource allocation. *arXiv preprint arXiv:2009.12922*, 2020.

[22] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proceedings of IEEE International Conference on Cloud Computing*, 2011.

[23] Aakash Sharma, Saravanan Dhakshinamurthy, George Kesidis, and Chita R. Das. CASH: A credit aware scheduling for public cloud platforms. In Laurent Lefèvre, Stacy Patterson, Young Choon Lee, Haiying Shen, Shashikant Ilager, Mohammad Goudarzi, Adel Nadjaran Toosi, and Rajkumar Buyya, editors, *21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 227–236. IEEE, 2021.

[24] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of European Conference on Computer Systems (EuroSys)*, 2011.

[25] Bo Sun, Yuxuan Jiang, and Danny H. K. Tsang. When burstable instances meet mobile computing: Performance modeling and economic analysis. In *40th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 1179–1180. IEEE, 2020.

[26] X. Sun, N. Ansari, and R.Wang. Optimizing resource utilization of a data center. *IEEE Communications Surveys & Tutorials*, 2016.

[27] Luan Teylo, Luciana Arantes, Pierre Sens, and Lúcia Maria de A Drummond. Scheduling bag-of-tasks in clouds using spot and burstable virtual machines. *IEEE Transactions on Cloud Computing*, 11(1):984–996, 2021.

[28] M. Tirmazi, A. Barker, N. Deng, M.E. Haque, Z.G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes. Borg: The next generation. In *Proceedings of European Conference on Computer Systems (EuroSys)*, 2020.

[29] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at google with borg. In *Proceedings of European Conference on Computer Systems (EuroSys)*, 2015.

[30] Cheng Wang, Bhuvan Urgaonkar, Aayush Gupta, George Kesidis, and Qianlin Liang. Exploiting spot and burstable instances for improving the cost-efficacy of in-memory caches on the public cloud. In Gustavo Alonso, Ricardo Bianchini, and Marko Vukolic, editors, *Proceedings of the Twelfth European Conference on Computer Systems (EuroSys)*, pages 620–634. ACM, 2017.

[31] Cheng Wang, Bhuvan Urgaonkar, Neda Nasiriani, and George Kesidis. Using burstable instances in the public cloud: Why, when and how? *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):11:1–11:28, 2017.