ECHO: Efficiently Overbooking Applications to Create a Highly Available Cloud

Parisa Rahimzadeh*, Youngbin Im*, Gueyoung Jung†, Carlee Joe-Wong‡ and Sangtae Ha*
* University of Colorado Boulder, {parisa.rahimzadeh, youngbin.im, sangtae.ha}@colorado.edu
† AT&T Labs-Research, gjung@research.att.com
‡ Carnegie Mellon University, cjoewong@andrew.cmu.edu

***Abstract*—Ensuring high availability for applications despite unpredictable cloud component failure events is a well-known problem in managing cloud infrastructure. One proposed solution uses a VM redundancy approach, reserving cloud resources for backup VMs that can substitute for primary ones in case of a failure event. However, this solution decreases the cloud resource utilization, since the backup resources usually remain idle. In this paper, we propose ECHO, a cloud resource management system that overbooks these backup VMs by optimizing the overbooking rate tradeoff between maximizing the cloud resource utilization, and thus maximizing the cloud provider's revenue; and improving application availability, thus satisfying users. Specifically, ECHO first obtains the optimal overbooking rate required to achieve a cloud provider's desired resource utilization level. It then computes the optimal (required) number of backup VMs that are required to maintain a given application availability level. Our extensive experimental and simulation results show that using ECHO can increase the number of accepted applications with satisfied availability by about 30%, while increasing the defined resource utilization at the same time.**

I. Introduction

Providing high availability for cloud services despite unpredictable failures is one of the most important challenges of cloud computing [1]: downtime on cloud-hosted applications due to these failures can lead to significant revenue losses and customer dissatisfaction, and may discourage businesses from utilizing cloud infrastructure [2]. Many research studies [1], [3]–[7] and commercial solutions [8], [9] have tackled this problem. An often-proposed solution is to use replication, or redundancy, in the virtual machines (VMs) provisioned for a given application, thus offering an application seamless recovery after a failure. With this approach, application VMs are divided into two groups: primary VMs[1] actively run to provide the required services, and backup VMs take over from a primary VM when it fails. Yet this approach harms cloud resource utilization: backup VMs can waste limited cloud resources, as they are mostly idle during normal operation.

We solve this problem by overbooking backup VMs. Replication-based solutions can utilize different types of backup VMs based on their state synchronization level, including Hot, Warm, and Cold backups [6]. These can be further divided into dedicated backups, which can only replace their own primary VM, and shared backups, where any backup instance can replace any primary VM of that application[2]. We consider a stateful system that uses updated Hot shared backup VMs. Thus, each application's backup VMs only need to synchronize their states with the primary VMs of that application. When not activated, they will then only require a portion of the resources (CPU or memory) used by a primary VM. Thus, we can use backup VM overbooking, in which many backup VMs of different applications share the resources of a single host, in order to increase cloud resource utilization.

Overbooking can indeed improve cloud resource utilization, which benefits a cloud provider's revenue. However, too much overbooking may decrease application availability, which hurts cloud tenants' quality of experience (QoE). For instance, the application availability will decrease if two backup VMs on the same host need to be activated at the same time, while the available resources are sufficient for only one activated backup. Realizing backup VM overbooking as a practical solution [10] thus requires us to navigate this tradeoff, which involves two variables: the overbooking rate, and the number of backup VMs for each application. We find the optimal overbooking rate to achieve a cloud provider's desired resource utilization threshold, thus protecting the provider revenue. And we quantify the optimal number of backup VMs for each application to guarantee a user-specified application availability, ensuring each cloud tenant's QoE (cf. Fig. 2).

Heterogeneous application requests and correlated failures in the cloud hierarchy make finding the right overbooking rate and number of backup VMs challenging. Since applications with different numbers of primary VMs and different requested availability may need to activate their co-placed backup VMs simultaneously, application availability is hard to compute.

In this paper, we propose ECHO (Efficient Clouds with High availability through Overbooking), a cloud resource management system that optimally overbooks backup VMs by considering both the cloud provider's desired resource utilization and each tenant's application availability. It accepts application requests including metadata expressing their required numbers of primary VMs and application availability. It then decides the number of backup VMs in order to provide that application's required availability. While ECHO currently

[1]Primary/backup VMs are also known as active/standby VMs, respectively.

[2]Note that each application has its own primary and backup VMs, and in the shared backup VM case, any backup VM of a specific application can take over any primary VM of that application.

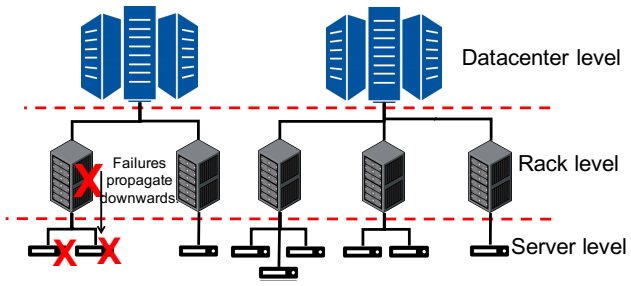


Fig. 1: Resource hierarchy in the datacenter infrastructure. Failures propagate from datacenters to racks to servers.

takes cloud resources to be CPU cores, it can be applied to other types of resources, e.g., memory.

Our contributions in this paper include:

- We propose ECHO, a cloud resource management system that optimally overbooks backup resources to increase the cloud resource utilization of an $n+k$ application, i.e., one with n active and k backup VMs (Section II).
- To build ECHO, we first study the tradeoff between cloud resource utilization and application availability. We obtain the optimal overbooking rate that maximizes application availability, while satisfying a cloud provider's desired utilization level (Section III).
- We then consider ECHO from the cloud tenant's perspective of ensuring a given application availability. We provide an algorithm to find the required number of backup VMs to achieve a given application availability in the hierarchical datacenter structure, given the provider's overbooking rate (Section IV).
- In deriving the right number of backup VMs that tenants should specify, we provide the first analytical framework to evaluate application availability, considering correlated component failures in typical datacenters (Section IV-A).
- We have used both experiments and simulations to evaluate the performance of ECHO. Our results confirm that ECHO can increase the number of accepted applications with satisfied availability by about 30%, and increase the defined resource utilization at the same time. These results validate that our algorithm realizes significant performance improvement and can scale to run in realistically-sized clouds (Section V).

We emphasize that ECHO may be used with any existing VM placement algorithm (e.g., [10]). Finding the optimal placement scheme complements our work on the optimal overbooking scheme, and is out of the scope of this paper.

II. System Overview and ECHO design

In this section, we provide an overview of the considered scenario in this paper and ECHO's design. We consider a common datacenter system, where cloud providers provide computing infrastructure as a service (IaaS) for the cloud tenant's requested applications over time. The datacenter system has a hierarchical infrastructure, with physical servers distributed across datacenters. Figure 1 shows an example hierarchy, with servers grouped into datacenter racks. Fail-stop failure events,

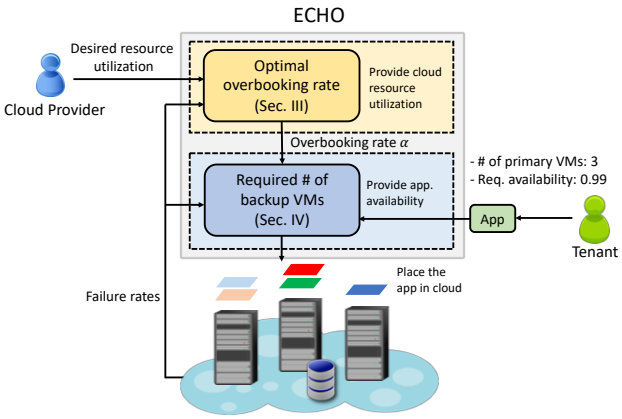


Fig. 2: ECHO design.

e.g., power failures, hardware failure, etc., can occur at each level, and failures propagate downwards to lower levels. For example, when a datacenter failure event happens, all of this datacenter's racks (and all of the racks' servers) are unavailable until the failure is repaired. To differentiate the failures at each level, we define MTBF_{DC}, MTBF_{R}, and MTBF_{S} as the mean times between failures at the datacenter, rack, and server levels, respectively. Higher levels have rarer failure events and higher MTBFs. The mean time to repair a failure is defined as MTTR. We formally define the *status* of each cloud component as either "working" or "failed". Each primary VM needs one dedicated CPU, and each backup VM needs $1/\alpha$ fraction of the server's CPU, i.e., α number of backup VMs can share one CPU: in the absence of failures, backup VMs only need to sync state, using much fewer resources than fully running primary VMs. We call α the *overbooking rate*, which our proposed system ECHO computes to satisfy the cloud provider's desired resource utilization.

Figure 2 shows ECHO's design. ECHO is a cloud management system for the cloud provider and collects cloud statistics (e.g., different cloud components' failure/recovery rates) periodically. It takes these cloud statistics and the cloud provider's desired resource utilization as inputs to find the optimal overbooking rate of backup VMs (Section III). ECHO then takes each application's metadata (including the number of its primary VMs and a required application availability[3]) along with the cloud statistics to determine the number of backup VMs needed for this application (Section IV). Each primary/backup VM is then placed on a physical server.

III. ECHO for the cloud provider

In this section, we will explain how ECHO finds the required optimal overbooking rate (α^*) for the cloud provider's desired resource utilization. In order to find the optimal overbooking rate, we note that a higher value of α means more backup VMs packed in one CPU, and thus higher resource utilization. However, hosting more backup VMs in one server will hurt the application availability, requiring more backup VMs per application and leading to fewer applications

[3]We formally define application availability in Section IV-A.

accepted to the cloud. To quantify this tradeoff, we formally define the *resource utilization* U:

Definition 1. The *resource utilization* of the cloud is the average number of VMs per CPU core on its servers. For n number of primary and k backup VMs, the cloud resource utilization is $U(k) = \frac{((C-1)\alpha+1)(n+k)}{((C-1)\alpha+1)n+Ck}$.

Because the exact resource utilization depends on the number of primary and backup VMs of different applications (n and k respectively), and since $U(k)$ is increasing in k, here we assume that applications use the minimum required backup VMs (k^*) for a typical number n of primary VMs, to guarantee an application's requested availability.

In order to do formalize this assumption, we first introduce a formal definition of availability.

Definition 2. The *availability* of an application with n primary and k backup VMs is the ratio of the time that at least n VMs are functional over the total application operation time.

Consider an application a with $n^{(a)}$ primary VMs that has a required availability of $\gamma^{(a)}$. If there is no overbooking and all primary VMs are in different datacenters, we can obtain the minimum required number of backups. The VM failures will be independent of each other and occur with probability $P_f = p_d + p'_d(p_r + p'_r p_s)$, where p_d, p_r, and p_s, are failure probabilities of a datacenter, a rack, and a server, respectively; the working probabilities are denoted as $p'_x = 1 - p_x$. In order for the application to be available, more than $n^{(a)} - 1$ VMs should be working, leading to an availability

$$\gamma^{(a)} = 1 - F(n^{(a)} - 1, n^{(a)} + k^{(a)}, P'_f), \tag{1}$$

where $F(i, M, P)$ is the CDF of the Bernoulli random variable with parameters (M, P) at i. The minimum required number of backups is the solution to the two equivalent optimization problems

$$\min k, \text{ s.t. } 1 - F(n^{(a)} - 1, n^{(a)} + k, P'_f) \geq \gamma^{(a)}, \tag{2}$$

$$\min k, \text{ s.t., } \sum_{i=n^{(a)}}^{n^{(a)}+k} \binom{n^{(a)}+k}{i} P_f^{n^{(a)}+k-i} P_f'^{i} \geq \gamma^{(a)}. \tag{3}$$

To see that (3) can be solved, we note that the function $F(i, M, P)$ is decreasing in M, so $1-F(n^{(a)}-1, n^{(a)}+k, P'_f)$ is increasing in k. Since $\lim_{k\to\infty} 1 - F(n^{(a)} - 1, n^{(a)} + k, P'_f) = 1$ and $0 < \gamma^{(a)} < 1$ the optimization problem in (3) always has a solution k^*.

Now since $U(k)$ is increasing in k, if $U(k^*) \geq u$, where k^* solves the optimization problem in (3), the real cloud resource utilization value will be larger than the desired utilization u.

Proposition 1. *The optimal α in terms of application availability, which also satisfies a required utilization threshold u, is*

$$\alpha^* = \left\lceil \frac{u(n + Ck^*) - n - k^*}{(C-1)(n + k^* - un)} \right\rceil, \tag{4}$$

where k^* solves the optimization problem in (3).

Proof. A smaller α leads to better application availability, so the optimal α in terms of application availability, which also satisfies a required utilization threshold u, is the solution to

$$\min \alpha, \text{ s.t., } \frac{((C-1)\alpha+1)(n+k^*)}{((C-1)\alpha+1)n + Ck^*} \geq u, \tag{5}$$

where k^* is the answer to the optimization problem in (3) with $n^{(a)} = n$ and $\gamma^{(a)} = \gamma$. Since $\frac{((C-1)\alpha+1)(n+k^*)}{((C-1)\alpha+1)n+Ck^*}$ is increasing in α, we can solve it to find $\alpha^* = \lceil \frac{u(n+Ck^*)-n-k^*}{(C-1)(n+k^*-un)} \rceil$. □

IV. ECHO FOR TENANTS

In order to find the required number of backup VMs for the tenant application, we need to evaluate the availability of an application with a determined placement of its primary and backup VMs. Our proposed solution finds the sufficient number of backup VMs to guarantee an application's requested availability, which first places the minimum required number of backup VMs (found with solving the optimization problem in (3)). We then find the application availability at the given overbooking rate and continue to add backup VMs until it exceeds the required availability, $\gamma^{(a)}$.

A. Application Availability Model

In this section, we present a model for application availability in the presence of backup VM overbooking.

Considering the application availability definition in Definition 2, in order to find the availability of an application, we need to find the probability that the number of its VMs that are "functional" is at least the number of primary VMs. For a primary VM to be functional, the corresponding server should be "working". On the other hand, a backup VM is only functional if the host server is "working" and can allocate one CPU to it. If a server has C CPUs, it can host up to C primary VMs (one CPU for one primary VM). Each CPU can host up to α backup VMs. Therefore, if the server has C cores and there are p primary and b backup VMs assigned to this server, the maximum number of activable backups (b') must satisfy $p + \frac{b-b'}{\alpha} + b' \leq C$, since p number of primary VMs and b' number of activated backup VMs need a whole CPU, while α number of $b - b'$ unactivated backups can share one CPU. So

$$b' = \left\lfloor \frac{C\alpha - p\alpha - b}{\alpha - 1} \right\rfloor. \tag{6}$$

Since at least one backup should be able to be activated, the maximum number of backup VMs that can be placed on this server is $(C-p-1)\alpha+1$. Thus as α increases, the maximum number of competing backup VMs increases linearly.

In a datacenter environment, different VMs of an application are assigned across a set of physical servers in a cloud hierarchy (cf. Fig. 1). Computing the application availability with backup VM overbooking in this hierarchy is difficult:

1) The status of different servers in a datacenter are interdependent. If a datacenter fails, all the servers in that datacenter will fail too. Similarly, a rack failure will cause a failure of all servers in that rack (Fig. 1).

2) With overbooking (i.e., $\alpha > 1$), the possibility of activation of a backup VM in an overbooked physical server depends on the status of other co-located backup VMs, which may serve other applications. Based on (6), there is a limit on the number of backups that can be activated simultaneously to work as a primary VM in a server.

We overcome these complexities in two steps. First, we find the joint probability distribution of servers' working/failed status in a hierarchical datacenter infrastructure (Section IV-B). We then map the server status to that of the application VMs, and thus application availability (Section IV-C).

B. Computing server status

To illustrate how we find the server status probabilities, we consider Fig. 3's example. We use N, with appropriate subscripts, to denote the number of racks corresponding to a given datacenter and servers to a given rack. There are two datacenters ($N = 2$), D_1 has two racks, $R_{1,1}$ and $R_{1,2}$ ($N_1 = 2$) and D_2 has one rack, $R_{2,1}$, ($N_2 = 1$). Each rack of D_1 has one server, $S_{1,1,1}$ and $S_{1,2,1}$, ($N_{1,1} = N_{1,2} = 1$), while the rack of D_2 has two servers, $S_{2,1,1}$ and $S_{2,1,2}$, ($N_{2,1} = 2$). We denote the datacenter, rack, and server failure probabilities as p_d, p_r, and p_s, respectively; the working probabilities are denoted as $p'_x = 1-p_x$. Given the mean times between failures as defined in Section II, we find that $p_d = \frac{\text{MTTR}}{\text{MTBF}_{\text{DC}}+\text{MTTR}}$, $p_r = \frac{\text{MTTR}}{\text{MTBF}_{\text{R}}+\text{MTTR}}$, and $p_s = \frac{\text{MTTR}}{\text{MTBF}_{\text{S}}+\text{MTTR}}$.

As an example, suppose that we want to obtain the probability of the server status vector of $z = [1;0;0;0]$. Each element of z represents the working (1) or failed (0) status of each server in the order of their index (i.e., from left to right in Fig. 3). In other words, $\mathbb{P}(z)$ is the probability that server $S_{1,1,1}$ is working and all others are failed. To find $\mathbb{P}(z)$, we note that datacenter failure events are independent, so $\mathbb{P}(z) = \mathbb{P}([1;0]_1)\mathbb{P}([0;0]_2)$. Each subscript on the datacenter-specific status sub-vector represents a datacenter (e.g., $[1;0]_1$ represents datacenter 1's servers).

First, to find $\mathbb{P}([1;0]_1)$, i.e., the status of servers in the first datacenter, we know that D_1 must be working (with probability p'_d), because otherwise both servers in this datacenter would be failed. The status of D_1's racks then become independent. Thus, $\mathbb{P}([1;0]_1) = p'_d\mathbb{P}([1]_{1,1})\mathbb{P}([0]_{1,2})$, where the subscripts on the status sub-vectors now represent both datacenter and rack indices. For $S_{1,1,1}$ to be working, $R_{1,1}$ and $S_{1,1,1}$ should be working, which occurs with probability $\mathbb{P}([1]_{1,1}) = p'_r p'_s$. For $S_{1,2,1}$ to be failed, either $R_{1,2}$ or $S_{1,2,1}$ needs to be failed. $R_{1,2}$ is failed with probability p_r and if $R_{1,2}$ is not failed, the server $S_{1,2,1}$ must be failed with probability p_s. So $\mathbb{P}([0]_{1,2}) = p_r + p'_r p_s$. Thus, $\mathbb{P}([1;0]_1) = p'_d(p'_r p'_s)(p_r + p'_r p_s)$.

Second, to obtain $\mathbb{P}([0;0]_2)$, D_2 may be failed with probability p_d. If D_2 is working, $R_{2,1}$ may be failed. If D_2 and $R_{2,1}$ both are working, both $S_{2,1,1}$ and $S_{2,1,2}$ servers must be failed with probability p_s^2. Thus, $\mathbb{P}([0;0]_2) = p_d + p'_d(p_r + p'_r p_s^2)$. Finally, $\mathbb{P}(z) = \big(p'_d(p'_r p'_s)(p_r + p'_r p_s)\big)\big(p_d + p'_d(p_r + p'_r p_s^2)\big)$.

We now consider a general cloud structure with N datacenters. Datacenter i has N_i number of racks, and rack j of datacenter i has $N_{i,j}$ number of servers. The status vector of all servers is $z = [z_1, z_2, ..., z_N]$, where z_i is the status vector of servers in datacenter i. Similarly, $z_i = [z_{i,1}, z_{i,2}, ..., z_{,i,N_i}]$, where $z_{i,j}$ is the status vector of servers in the rack j of datacenter i. Similarly, $z_{i,j} = [z_{i,j,1}, z_{i,j,2}, ..., z_{i,j,N_{i,j}}]$. The probability of the generalized status vector in an arbitrary datacenter structure can be obtained based on Theorem 1.

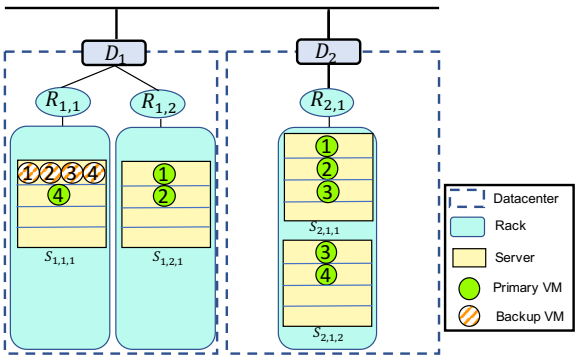


Fig. 3: An example of cloud hierarchy and VM assignment

Theorem 1. *The probability of the work/fail status vector of servers can be obtained by*

$$\mathbb{P}(z) = \prod_{i=1}^{N}\Bigg\{p_d\mathbf{1}(z_i = \mathbf{0}) + p'_d\prod_{j=1}^{N_i}\Bigg\{p_r\mathbf{1}(z_{i,j} = \mathbf{0}) + p'_r p_s^{N_{i,j}-x_{i,j}}{p'_s}^{x_{i,j}}\Bigg\}\Bigg\}, \tag{7}$$

where $\mathbf{1}$ *is the indicator function and* $\mathbf{0}$ *is the all-zero vector.* $x_{i,j}$ *is the number of working servers in the rack* j *of datacenter* i *(i.e.,* $\sum z_{i,j}$*).*

C. Computing application (VM) status

In the second step of our analytical model, we consider one application to find its availability for a specific placement of its VMs. Assume a specific VM assignment (A) to different servers in a cloud structure, where A is a function mapping each VM to its host server in the cloud. We consider an application a with $n^{(a)}$ primary VMs, $P^{(a)} = \{P_1^{(a)}, ..., P_{n^{(a)}}^{(a)}\}$, and $k^{(a)}$ backup VMs, $B^{(a)} = \{B_1^{(a)}, ..., B_{k^{(a)}}^{(a)}\}$. Furthermore, elements of the vector $x^{(a)} = [x_{\text{p}}^{(a)}, x_{\text{b}}^{(a)}]$, which has size $n^{(a)} + k^{(a)}$, represent the functional (1) or nonfunctional (0) status of the VMs of the considered application a. Here $x_{\text{p}}^{(a)}$ denotes the status of the primary and $x_{\text{b}}^{(a)}$ the backup VMs. If at least $n^{(a)}$ out of all VMs are functional, the application is available. Therefore, the availability of application a can be obtained as

$$\gamma^{(a)} = 1 - \sum_{\substack{x^{(a)} \\ \sum x^{(a)} < n^{(a)}}} \mathbb{P}(x^{(a)}). \tag{8}$$

We first suppose there is no overbooking (i.e., $\alpha = 1$). In this case, the (non)functional status of both primary and backup VMs of an application means their host servers are working or failed, respectively. Thus, we can simply use the

A function to relate the server status z to VM status $x^{(a)}$. If $x^{(a)}(P_i^{(a)}) = 0(1)$, then $z(s) = 0(1)$, where $s = A(P_i^{(a)})$.

Proposition 2. *If $\alpha = 1$, the availability of application a is*

$$\gamma^{(a)} = 1 - \sum_{\substack{x^{(a)} \\ \sum x^{(a)} < n^{(a)}}} \mathbb{P}(z), \tag{9}$$

where z is the server status vector and $z(s) = x^{(a)}(A^{-1}(s))$ and $\mathbb{P}(z)$ is obtained by Theorem 1.

Given this result, we can consider the general case when there is overbooking (i.e., $\alpha > 1$). In this case, the working status of a backup server does not necessarily represent the backup VM's functionality. So, each server status vector z can lead to multiple functional/nonfunctional VM status vectors x for application a: other applications may have already activated their backups on a given server, which could prevent application a from activating its own backup on that server[4].

We illustrate our approach with an example assuming that applications (1)-(4) with 2 primary and 1 backup VMs are assigned to the servers as shown in Fig. 3. For $z = [1; 1; 1; 0]$, all applications are available. Because of the working status of servers $S_{1,2,1}$ and $S_{2,1,1}$, applications 1 and 2 have two functional primary VMs. Since $S_{2,1,2}$ is failed, applications 3 and 4 each have a single nonfunctional primary VM, so they have to activate their backup VMs assigned to the working server $S_{1,1,1}$, which contains one primary VM of application 4 as well.

Based on (6), the maximum number of activable backup VMs in the server $S_{1,1,1}$ is $b' = 2$, so both applications can activate their backup VMs. Thus in this example, the given $z = [1; 1; 1; 0]$ implies $x^{(1)} = x^{(2)} = [1, 1, 0]$, and $x^{(3)} = x^{(4)} = [1, 0, 1]$ with probability 1 ($\mathbb{P}(x^{(i)}|z) = 1, \forall i \in \{1, 2, 3, 4\}$). All applications are available.

Now consider the general cloud structure. Given the relationship between the z and $x^{(a)}$ vectors, we let $Z[x^{(a)}]$ denote the set of all server status vectors z such that $\mathbb{P}(x^{(a)}|z) > 0$, i.e., the VM status vector $x^{(a)}$ is feasible; and $X^{(a)}$ the set of all application status vectors with $\sum x^{(a)} < n^{(a)}$ (i.e., the application cannot activate enough backups)[5]. The application availability in (8) then becomes

$$\gamma^{(a)} = 1 - \sum_{x^{(a)} \in X^{(a)}} \Big[\sum_{z \in Z[x^{(a)}]} \mathbb{P}(x^{(a)}|z)\mathbb{P}(z) \Big]. \tag{10}$$

In order to solve (10), $\mathbb{P}(z)$ is obtained by Theorem 1. The problem then reduces to finding the set $Z[x^{(a)}]$ and $\mathbb{P}(x^{(a)}|z)$, $\forall z \in Z[x^{(a)}]$. Given the VM status dependencies induced by overbooking backup VMs for different applications on the same servers, obtaining a closed-form solution is hard. For each overbooked backup VM, we must consider all cases in which other applications with co-located backup VMs may need to activate them. Since this space is large, we propose to randomly sample it in Algorithm 1, collecting enough samples to ensure that we cover all cases. Each case[6] forces a specific functional/nonfunctional status for those applications' primary VMs, which allows us to update the servers' status vectors (z) in Algorithm 2. We then obtain a z vector to add to the set $Z[x^{(a)}]$. To find $\mathbb{P}(x^{(a)}|z)$, we obtain the empirical probability that application a can or cannot activate its backup VM, considering that all applications are equally likely to activate their backup VMs.

Algorithm 1 Availability of application (a)

for $x^{(a)} \in X^{(a)}$ **do**
  $Z[x^{(a)}] \leftarrow []$
  while $Z[x^{(a)}]$ changes **do**
    $p \leftarrow 1$
    for $i \in \{1, ..., n^{(a)}\}$ **do**
      $s \leftarrow A(P_i^{(a)})$
      $z(s) \leftarrow x^{(a)}(P_i^{(a)})$
    end for
    for $i \in \{1, ..., k^{(a)}\}$ **do**
      $s \leftarrow A(B_i^{(a)})$
      $c(s) \leftarrow \{a' | \exists k; A(B_k^{(a')}) = s\}$
      if $x^{(a)}(B_i^{(a)}) = 1$ **then** ▷functional backup VM
        $z(s) \leftarrow 1$
        $[b'', -] \leftarrow \texttt{rand}([0 : |c(s)|], 1)$
        if $b'' \geq b'$ **then**
          $p \leftarrow p \times \frac{b'}{b''+1}$
        end if
      else if $x^{(a)}(B_i^{(a)}) = 0$ **then** ▷nonfunctional backup VM
        either
          $z(s) \leftarrow 0$
        or
          $z(s) \leftarrow 1$
          $[b'', -] \leftarrow \texttt{rand}([b' : |c(s)|], 1)$
          $p \leftarrow p \times (1 - \frac{b'}{b''+1})$
        end if
      $[c_n(s), c_d(s)] \leftarrow \texttt{rand}(c(s), b'')$
      $z \leftarrow \texttt{updatez}(c_n(s), c_d(s))$
    end for
    add z to $Z[x^{(a)}]$ and $\mathbb{P}(x^{(a)}|z) \leftarrow p$
  end while
end for
return $1 - \sum_{x^{(a)} \in X^{(a)}} \Big[\sum_{z \in Z[x^{(a)}]} \mathbb{P}(x^{(a)}|z)\mathbb{P}(z) \Big]$

The detailed proposed pseudocode is presented in Algorithm 1. For each $x^{(a)} \in X^{(a)}$, we find the $z \in Z[x^{(a)}]$ vectors and corresponding $\mathbb{P}(x^{(a)}|z)$ by considering all possible states for each application a's VMs:

- If a primary VM $P_i^{(a)}$ is functional (respectively nonfunctional), i.e., $x^{(a)}(P_i^{(a)}) = 1(0)$, the hosting server, $s = A(P_i^{(a)})$, should be working (failed), $z(s) = 1(0)$.
- If a backup VM $B_i^{(a)}$ is functional (i.e., $x^{(a)}(B_i^{(a)}) = 1$), the hosting server, $A(B_i^{(a)})$, should be set to working ($z(s) = 1$). To account for backup activations by co-located applications, denoted by the set $c(s)$, we note that if the number of co-located applications that need their backups (denoted by the set $c_n(s)$), b'', is less than b' (i.e., $b'' < b'$), application a can activate its backup. If $b'' \geq b'$, the probability that application a can activate its backup is $\frac{b'}{b''+1}$. We randomly select the applications

[4]We assume applications randomly choose which of their available backups to activate whenever they need one.

[5]Because $n^{(a)}$ and $k^{(a)}$ are small in practice, $X^{(a)}$ is a fairly small set.

[6]By "case" we mean a situation where given sets of applications need or do not need their backup VMs

Algorithm 2 (`updatez` $(c_n(s), c_d(s))$)

Input: $z, c_n(s), c_d(s)$
Output: z
1: **for** $a' \in c_n(s)$ **do**
2: $[P_r^{(a')}, -] \leftarrow \texttt{rand}(P^{(a')}, 1)$
3: $z(A(P_r^{(a')})) = 0$
4: **end for**
5: **for** $a' \in c_d(s)$ **do**
6: $[P_r^{(a')}, -] \leftarrow \texttt{rand}(P^{(a')}, n^{(a')} - k^{(a')} + 1)$
7: $z(A(P_r^{(a')})) = 1$
8: **end for**
9: **return** z

that need or do not need to activate their backups using a uniform random partition.

- If a backup VM $B_i^{(a)}$ is nonfunctional (i.e., $x^{(a)}(B_i^{(a)}) = 0$), either the hosting server, $(A(B_i^{(a)}))$, is failed $(z(s) = 0)$, or the server is working, $(z(s) = 1)$; however, among all the other applications with co-located backup VMs in the server, at least b' number of applications need their backups to be activated. If $b'' \geq b'$ number of co-located applications need their backups to be activated, the probability that application a cannot activate its backup is $1 - \frac{b'}{b''+1}$. We sample these applications with a uniform random partition.

Finally, the z server status vector should be updated to incorporate the effect of the co-located backup applications, using the `updatez`$((c_n(s), c_d(s))$ function. This function updates the z vector based on the condition of co-located applications (set c), which need to activate ($c_n(s)$ set) or do not need to activate ($c_d(s)$ set) their backups. The pseudocode for the `updatez` function is presented in Algorithm 2. If application a' needs to activate its backup VM, we know that the server hosting one of its primary VMs is failed. So we select a random primary VM and set the corresponding index in z at 0. If application a' does not need its backup VM, then at least $n' - k' + 1$ number of its primary VMs are functional, so we set the hosting servers of randomly selected $n' - k' + 1$ primary VMs to be working. Finally, the updated z is returned to Algorithm 1 and is added to the set $Z[x^{(a)}]$.

To illustrate this algorithm, consider the scenario depicted in Fig. 3 again to find the availability of application 1 with this specific VM assignment. We have to consider each member of $X^{(1)}$ to find the corresponding $z \in Z[x^{(1)}], \forall x^{(1)} \in X^{(1)}$ vectors. Let us first consider $x^{(1)} = [0, 0, 0]$. We want to find the set $Z[x^{(1)} = [0, 0, 0]]$ and $\mathbb{P}(x^{(1)}|z), \forall z \in Z[x^{(1)}]$. For nonfunctional primary VMs, their hosting servers should be set to failed, $z(S_{1,2,1}) = 0$ and $z(S_{2,1,1}) = 0$. For the nonfunctional backup VM, either the host server is failed $(z(S_{1,1,1}) = 0)$, in which case $z = [0, 0, 0, -]$ is added to Z with $\mathbb{P}(x^{(1)}|z) = 1$; or the server is working $(z(S_{1,1,1}) = 1)$, but at least 2 applications in $c(s) = \{2, 3, 4\}$ need to activate their backups. As an example, assume that applications 2 and 3 need to activate their backups (so that $b'' = 2$), and thus that $c_n(s) = \{(3), (2)\}$ and $c_d(s) = \{(4)\}$. Based on Algorithm 2, both servers of application 4's primary VMs should be set to working, $z(S_{2,1,2}) = 1$ ($z(S_{1,1,1})$ is already 1). And we randomly select 1 primary VM of applications 2 and 3 and set their hosting servers to be failed ($z(S_{1,2,1})$ and $z(S_{2,1,1})$ are already 0). So $z = [1, 0, 0, 1]$ can be added to $Z[x^{(a)}]$ with $\mathbb{P}(x^{(1)}|z) = \frac{1}{3}$. The final results of using Algorithm 1 for this example are presented in Table I. After obtaining $Z[x^{(a)}], \forall x^{(a)} \in X^{(a)}$ and the corresponding probabilities, and assuming p_d, p_r, and p_s to be 0.0021, 0.0034, and 0.0089, respectively, we find the availability $\gamma^{(1)} = 0.9918$. We confirm the accuracy of this proposed availability calculation algorithm in Section V.

TABLE I: Algorithm 1 results for the example in Fig. 3

$x^{(1)}$	$Z[x^{(1)}]$		
$[0,0,0]$	$z=[0,0,0,-]$ $p(x^{(a)}\|z)=1$	$z=[1,0,0,1]$ $p(x^{(a)}\|z)=1/3$	$z=[1,0,0,0]$ $p(x^{(a)}\|z)=1/2$
$[0,0,1]$	$z=[1,0,0,1]$ $p(x^{(a)}\|z)=2/3$	$z=[1,0,0,0]$ $p(x^{(a)}\|z)=1/2$	
$[0,1,0]$	$z=[0,0,1,-]$ $p(x^{(a)}\|z)=1$	$z=[1,0,1,0]$ $p(x^{(a)}\|z)=1/2$	
$[1,0,0]$	$z=[0,1,0,-]$ $p(x^{(a)}\|z)=1$	$z=[1,1,0,1]$ $p(x^{(a)}\|z)=1/3$	$z=[1,1,0,0]$ $p(x^{(a)}\|z)=1/2$

V. Evaluation

We report the performance of ECHO in our experimental testbed and large-scale simulation.

A. Performance metrics

To assess the tradeoff between resource utilization and application availability, we define the *number of accepted applications with satisfied availability*, as a function of the utilization u chosen by the cloud provider and the required availabilities γ chosen by the applications. Accepted applications are simply the applications for which there are enough available resources in the cloud for them to host their primary and backup VMs. A higher resource utilization will lead to more accepted applications (those that are granted space for their primary and backups), and a higher availability will lead to more satisfied applications (those that are able to activate their backups when needed). We evaluate the performance of ECHO in terms of the following metrics: (1) the number of accepted applications with satisfied availability, (2) the percentage of satisfied applications, (3) cloud resource utilization, and (4) CPU resource utilization.

B. Baseline algorithms

As there is no similar work to ECHO, we compare the performance of ECHO with the following baseline algorithms used for backup VMs in the cloud:

w/o backup. No backup VMs are used for applications.

w/o overbooking. Backup VMs are used for primary ones, but not overbooked.

Min-backup method. For each application a, we place the minimum required number of backup VMs to achieve the application required availability $\gamma^{(a)}$, i.e., the solution to (3). This method, however, does not account for overbooking's effect on application availability.

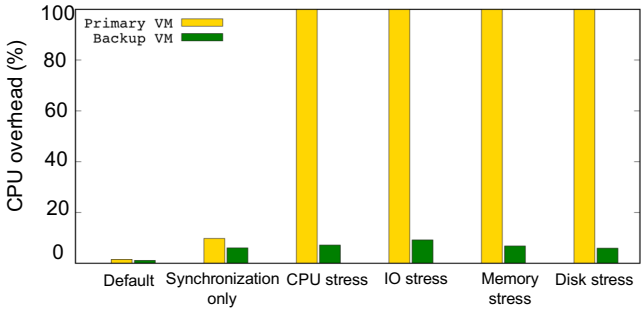


Fig. 4: CPU usages of the default VM process, when the state is synchronized, and with 4 different stress tests.

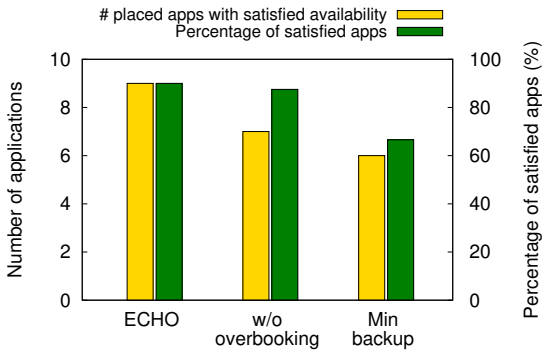


Fig. 5: Performance evaluation of ECHO with w/o overbooking and Min-backup methods in terms number of satisfied availability applications and ratio of satisfied applications.

C. Experimental Results

In our experimental evaluations, we first investigate the feasibility of VM overbooking. Then, we compare the performance of ECHO with other algorithms.

Feasibility of VM overbooking. To show the feasibility of VM overbooking, we show that the resources needed for synchronizing the backup VMs with the corresponding primary VMs are much smaller than the resources used by the primary VMs.

We use two machines for the experiment: one server running a primary VM and the other server running a backup VM. We run a Linux container in each VM. We use DRBD [11] for synchronizing the state of the container running in the primary VM. DRBD is a system for mirroring block devices such as VM images over the network [12]. We consider 4 different stress test scenarios: CPU, IO, memory, disk, by using *stress* [13] in the container of the primary VM, which is a tool that imposes load on systems. We measure the CPU usage of the VM process in each server for 120 seconds.

The average CPU usages for the default VM process, the state synchronization of the backup VM, and the stress test scenarios are illustrated in Figure 4. The backup VM's CPU usage for synchronization is significantly lower than that of the stress tested primary VM (which is 100%). This shows that the VM overbooking for backup VMs is feasible in practice. We can also observe that the synchronization overhead is not significantly affected by the application type.

Performance of ECHO. For a more realistic performance evaluation, we run an emulation experiment on a multi-core machine with a simple high-load application. We implement a cloud emulation software, which first reads the experiment scenario describing the servers and their CPU cores, cloud topology including racks and servers, applications and the locations of their primary VMs and backup VMs, and failure events. Then it creates the topology according to the scenario, runs the experiment, and reports the performance results such as VM activation times and CPU utilization. The topology of our experiment is one datacenter with 5 racks; each rack contains 3 servers. We use 2 CPU cores for each of our 15 servers. For a VM, we simply run an application process. For the application, we implement a CPU-consuming application that repeats a simple mathematical operation continuously, and stores the number of operations in a file. When the failure event happens, the application process is killed, and another application process is run on the backup VM, if any. Then the computation is resumed exactly from the point where the previous application process was running. For our implementation, we have used the cloud component failure parameters for the rampage scenario shown in Table III. We assume that the desired cloud availability is $u = 1.125$, so using Proposition 1, we find the required overbooking rate to be $\alpha = 3$.

We implement three different algorithms in our testbed; ECHO ($\alpha = 3$), w/o overbooking ($\alpha = 1$), and Min-backup ($\alpha = 3$). Fig. 5 presents the number of accepted applications with satisfied availabilities and the percentage of satisfied applications compared to the total number of placed applications, using these three algorithms. As it can be seen, using ECHO increases the number of satisfied applications by 29% and 50% compared to the w/o overbooking and Min-backup methods, respectively. Furthermore, the percentage of applications with satisfied availability is $\frac{7}{8}$ when w/o overbooking is used. Interestingly, while we expect the application availability to decrease when overbooking is used, ECHO achieves a slightly better percentage of applications with satisfied availability (90%). If the Min-backup method is used, the percentage would decrease to $\frac{2}{3}$.

Consistent with what expected, the found VM placement using ECHO has a cloud resource utilization $U = 1.125$. In addition, in our experimental testbed, the average CPU utilization using ECHO is 13% more than the average CPU utilization when there is no overbooking.

D. Simulation Results

To demonstrate that ECHO can scale to realistically sized clouds, in this section we evaluate the performance of the proposed algorithms using extensive simulations on larger topologies. First, we confirm the accuracy of our proposed algorithm to find the availability of applications with a known primary and backup VM placement. Then, we investigate the overbooking rate tradeoff in terms of applications' availability and cloud resource utilization.

TABLE II: Cloud structure parameters for considered small and large-scale scenarios.

	number of DC	racks per DC	servers per rack	C
small-scale	6	7	5	2
large-scale	small: 2	10	10	2
	medium: 2	30	20	
	large: 2	50	30	

TABLE III: Characteristics of our two failure states.

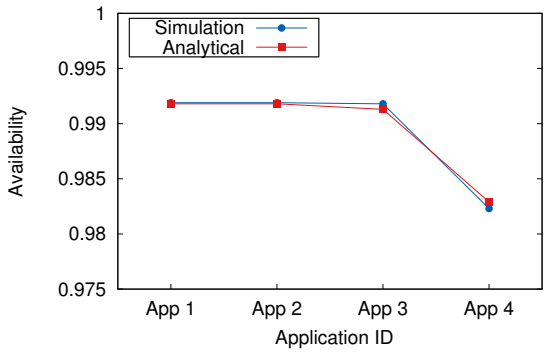
	MTTR	$MTBF_{DC}$, $MTBF_R$, $MTBF_S$	number of primary VMs (n)	Required availability (γ)
Optimistic	54	(130000,80000,30000)	1,2,3	0.99,0.995, 0.999,0.9995
Rampage	54	(26000,16000,6000)	1,2,3	0.99,0.995,0.999

Simulation setup. The cloud infrastructure parameters we have considered in our evaluations are based on the parameters of a commercial cloud provider [10]. We use two small-scale and large-scale cloud scenario configurations (Table II.)

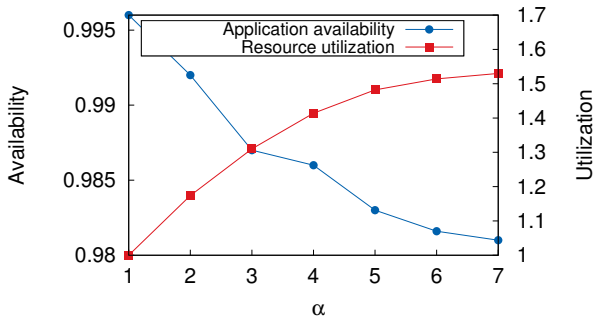
The two considered failure scenarios of the cloud components are also presented in Table III. The failure event processes of each cloud component are assumed to be Poisson processes [14] with the mean values presented in Table III. We consider two states for the cloud component failures: *Optimistic*, where the mean time between failures is based on the hardware specification of our private cloud, and *Rampage*, where failure events are more frequent, with parameters obtained based from previous studies [15], [16]. The duration of each simulation is set to 20 years and the resolution is 1 hour. The number of primary VMs of an application (n) and its required availability (γ) are chosen uniformly at random from the sets shown in Table III.

Validating our availability calculation. To confirm the accuracy of our proposed availability algorithm (Algorithm 1), we reconsider the example cloud scenario with four accepted applications (Fig. 3). We use Algorithm 1 to obtain the availability of these four accepted applications, and we compare the algorithm results with the simulation results in Fig. 6a. As can be seen in this figure, the simulation and analytical results match very well and the difference in application unavailability is less than 5% in the worst case.

Availability and resource utilization tradeoff. To study the tradeoff in increasing the overbooking rate in terms of application availability and the resource utilization, we consider a scenario where an example application has a fixed number of backup VMs. Figure 6b shows the availability of the example application considered, as well as the cloud resource utilization, versus the overbooking rate. With a larger overbooking rate, more backup VMs are placed in one server, thus increasing the cloud resource utilization. On the other hand, more backup VMs placed in one server means more applications competing for resources to activate their backup VMs in case of failures. So a specific application will have a lower probability of activating its backup VM, due to the limitation on the number of simultaneously activated backup VMs in a server. As a result, the availability of an application will decrease with increasing α. The resulting effects can be clearly seen in Fig. 6b.



(a) Availability of all four applications

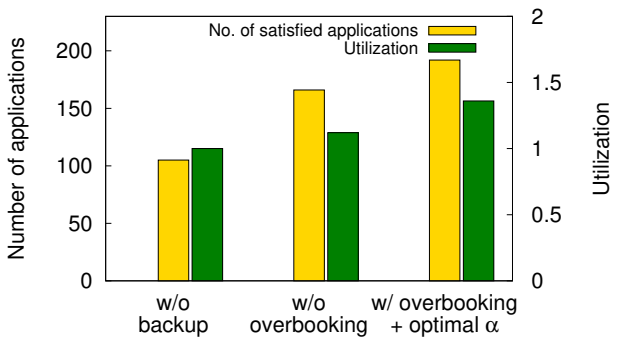


(b) Availability and utilization of an example application

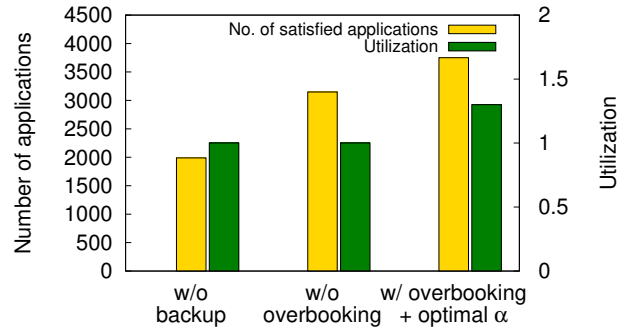
Fig. 6: (a): The simulation and analytical results of availability of four accepted applications in the example shown in Fig. 3 match well.
(b): Application availability and cloud resource utilization tradeoff versus the overbooking rate.

Performance of ECHO. We finally evaluate the performance of ECHO in terms of the number of accepted applications with satisfied availability. As explained earlier, this metric can take into account both availability and resource utilization, since a higher resource utilization will make room for more accepted applications, and a higher availability will lead to more satisfied applications. We therefore use this metric to consider both the application availability decrease and the cloud resource utilization increase due to overbooking backup VMs (i.e., $\alpha > 1$) in our proposed scheme.

Figure 7 summarizes our findings for the two considered scenarios (i.e., small and large-scale). Not only does ECHO increase the resource utilization by about 40% in both the small and large-scale scenarios, but the number of accepted applications with satisfied availability is also increased about 88% (respectively 82%) compared to no backup VMs and 20% (respectively 16%) compared to backup VMs without overbooking in the large (respectively small)-scale scenario. Overall, we find that *overbooking with the optimal overbooking rate significantly increases the number of applications with*



(a) Small-scale



(b) Large-scale

Fig. 7: ECHO (overbooking + optimal α) increases the number of applications with satisfied availability and cloud resource utilization with Optimistic failure rates.

satisfied availabilities as well as the cloud resource utilization.

Fig. 8 shows the number of accepted and satisfied applications over an increasing α, using ECHO with the Optimistic cloud component failure rates. For all values of α, almost all of the accepted applications achieve their required availability.

Fig. 9a shows the resource utilization of the Min-backup method and ECHO in the Rampage failure scenario. Obviously, increasing α allows more backup VMs to be hosted in one server increasing the resource utilization. Furthermore, since ECHO increases the number of applications' backup VMs, it increases the resource utilization compared to the simpler Min-backup method. As an example, compared to the no overbooking case, the resource utilization is increased by 30% and 48% where $\alpha = 5$ with the Min-backup and ECHO methods, respectively.

The resource utilization of Min-backup method and ECHO in the Optimistic failure scenario are shown in Fig. 9b. The increase in resource utilization due to increasing α or using the ECHO method is visible. By comparing Fig. 9b and Fig. 9a, we can see that the resource utilization is better in the

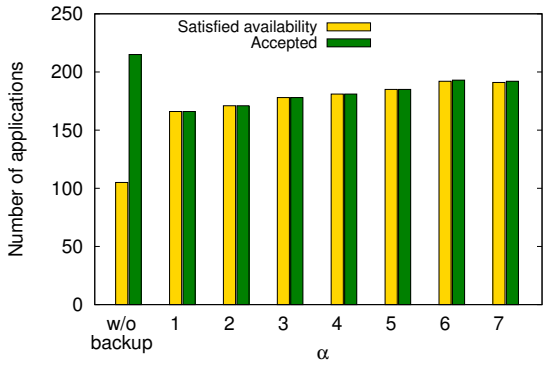
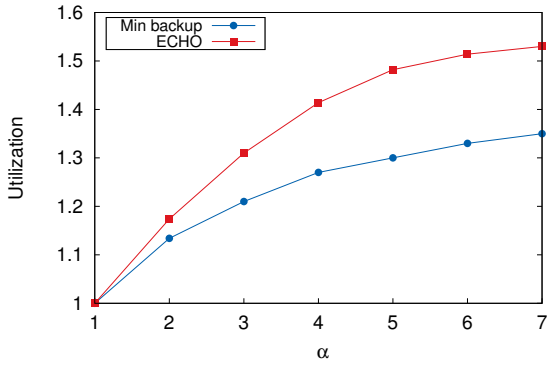
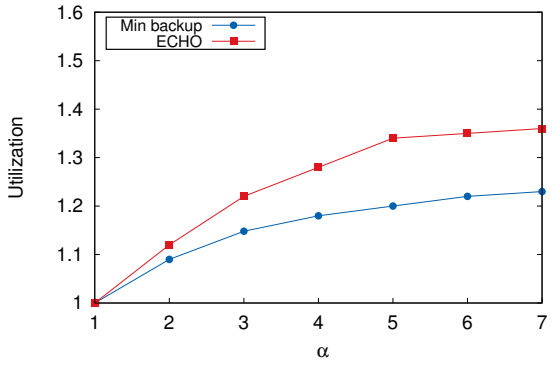


Fig. 8: Number of accepted applications and applications with satisfied availability using ECHO for small-scale scenarios and optimistic failure. For all $\alpha \geq 1$, almost all accepted applications have satisfied availability.



(a) Rampage failure



(b) Optimistic failure

Fig. 9: Resource utilization. More α means more backup VMs in each server and therefore better resource utilization.

Rampage scenario. In this scenario, applications need more backup VMs to achieve their required availability, although our considered applications in the Rampage failure scenario have less required availability on average. The average number of backup VMs for applications is shown in Fig. 10 for both failure scenarios. The average number of backup VMs

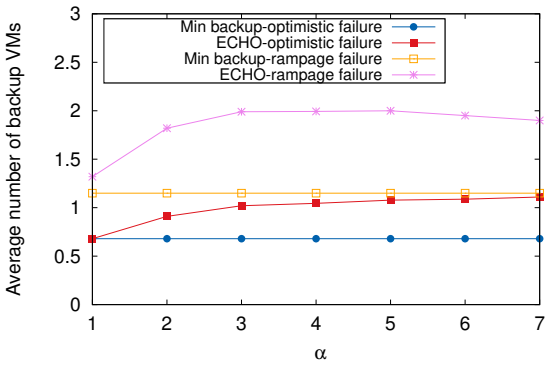


Fig. 10: The average number of backup VMs increases with α in the ECHO method.

is constant for different values of α when we use the Min-backup method, because this method places the minimum required number of backup VMs, independent of the value of α. However, with ECHO, as α increases, the average number of backup VMs that applications need increases. This is due to the additional competition in the servers hosting backup VMs.

VI. Related Work

Ensuring high availability despite resource failures is a crucial challenge in many types of failure-prone systems. One well-known solution uses redundancy, which several works have proposed for cloud computing systems [1], [3]–[7]. For instance, [4] uses virtualization and performs whole-system checkpointing to update the state of VMs asynchronously in the backup hosts. The authors of [7] use Linux containers and checkpoint the full-state of the containers to be used in the backup hosts, while [5] introduces ZORFU, a hierarchical system architecture for replication across datacenters.

Some prior works consider the placement of primary and backup VMs, e.g., focusing on the tradeoff between application performance and application availability [10], [17]–[19]. [20] proposes a placement algorithm for backup resources to decrease the cloud resource consumption when a failure event occurs. The authors of [10] propose a VM placement algorithm to increase the resource utilization by overbooking backup VMs of the single primary VM of each application. These placement algorithms can be complementary to our approach, which does not assume any specific placement algorithm.

Other works have shown that replicated services may harm cloud resource utilization due to the resources reserved for failure events. For instance, [21] and [22] are two previous cloud overbooking studies, which mainly focus on mixing complementary workloads or leveraging time-of-day workload patterns. However, the unpredictability of cloud applications can make these methods impractical.

VII. Conclusion

We study the overbooking rate tradeoff in a backup VM overbooking scheme to increase the cloud resource utilization and still improving application availability. To realize this idea, we propose ECHO, a system including two parts, where the first part obtains the optimal overbooking rate to realize the cloud provider's resource utilization, and the second part quantifies the required number of backup VMs for each application, to guarantee the cloud tenant's QoE.

In the second part, we present an analytical model to evaluate the application availability, assuming a specific number and placement for primary and backup VMs. This model considers the failure dependencies between different datacenter components as well as limitations on the number of simultaneously activated backup VMs. Next, the experimental and simulation results confirm the accuracy and effectiveness of our proposed solution.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] D. Pepijn, "Is your business ready for a catastrophic cloud failure?" 2017, https://www.business.com/articles/daan-pepijn-business-failures-in-the-cloud/.

[3] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, "Charm: A cost-efficient multi-cloud data hosting scheme with high availability," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 372–386, Jul. 2015.

[4] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High availability via asynchronous virtual machine replication," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08, 2008, pp. 161–174.

[5] J. W. Anderson, H. Meling, A. Rasmussen, A. Vahdat, and K. Marzullo, "Local recovery for high availability in strongly consistent cloud services," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 2, pp. 172–184, Mar. 2017.

[6] M. Nabi, M. Toeroe, and F. Khendek, "Availability in the cloud: State of the art," *J. Netw. Comput. Appl.*, vol. 60, no. C, pp. 54–67, Jan. 2016.

[7] W. Li, A. Kanso, and A. Gherbi, "Leveraging linux containers to achieve high availability for cloud services," in *IEEE International Conference on Cloud Engineering*, Mar. 2015, pp. 76–83.

[8] Stratus, "Easily supporting mission critical workloads in clouds," 2017, https://www.stratus.com/cloud-computing/.

[9] B. Jimerson, "Software architecture for high availability in the cloud," 2012, http://www.oracle.com/technetwork/articles/cloudcomp/jimerson-ha-arch-cloud-1669855.html.

[10] G. Jung, P. Rahimzadeh, Z. Liu, S. Ha, K. Joshi, and M. Hiltunen, "Virtual redundancy for active-standby cloud applications," in *IEEE Conference on Computer Communications (INFOCOM)*, 2018.

[11] "Docs LINBIT - Online Documentation Resource for DRBD," https://docs.linbit.com/, 2017.

[12] "DRBD - Linux-HA," http://www.linux-ha.org/wiki/DRBD, 2017.

[13] "stress(1): impose load on/stress test systems - Linux man page," https://linux.die.net/man/1/stress, 2017.

[14] H. Kim, Y. Ha, Y. Kim, K.-n. Joo, and C.-H. Youn, "A VM reservation-based cloud service broker and its performance evaluation," in *International Conference on Cloud Computing*, 2015, pp. 43–52.

[15] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: novel erasure codes for big data," in *Proceedings of the international conference on Very Large Data Bases*, 2013, pp. 325–336.

[16] X. Chen, C. D. Lu, and K. Pattabiraman, "Failure analysis of jobs in compute clouds: A google cluster case study," in *IEEE International Symposium on Software Reliability Engineering*, Nov. 2014, pp. 167–177.

[17] Y. Xia, M. Tsugawa, J. A. B. Fortes, and S. Chen, "Large-scale vm placement with disk anti-colocation constraints using hierarchical decomposition and mixed integer programming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1361–1374, May 2017.

[18] M. Mishra and U. Bellur, "Whither tightness of packing? the case for stable vm placement," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 481–494, Oct. 2016.

[19] L. Zhao, L. Lu, Z. Jin, and C. Yu, "Online virtual machine placement for increasing cloud providers revenue," *IEEE Transactions on Services Computing*, vol. 10, no. 2, pp. 273–285, Mar. 2017.

[20] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. Chang, M. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, Jan. 2017.

[21] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012, pp. 199–210.

[22] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, I. Goiri, and R. Bianchini, "History-based harvesting of spare cycles and storage in large-scale datacenters," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 755–770.