Virtual Redundancy for Active-Standby Cloud Applications

Gueyoung Jung[1], Parisa Rahimzadeh[2], Zhang Liu[2], Sangtae Ha[2], Kaustubh Joshi[1], and Matti Hiltunen[1]

[1]AT&T Labs – Research [2]University of Colorado Boulder

***Abstract*—VM redundancy is the foundation of resilient cloud applications. While active-active approaches combined with load balancing and autoscaling are usually resource efficient, the stateful nature of many cloud applications often necessitates 1+1 (or 1+n) active-standby approaches. Keeping the standbys, however, could result in inefficient utilization of cloud resources. We explore an intriguing cloud-based solution, where standby VMs from active-standby applications are selectively overbooked to reduce resources reserved for failures. The approach requires careful VM placement to avoid a situation where multiple standby VMs activate simultaneously on the same host and thus cannot get the full resource entitlement. Indeed today's clouds do not have this visibility to the applications. We rectify this situation through ShadowBox, a novel redundancy-aware VM scheduler that optimizes the placement and activation of standby VMs, while assuring applications' resource entitlements. Evaluation on a large-scale cloud shows that ShadowBox can significantly improve resource utilization (i.e., more than 2.5 times than traditional approaches) while minimizing the impact on applications' entitlements.**

I. Introduction

The active-standby redundancy is one of the oldest [1] yet most ubiquitously used design patterns for both fault tolerance and disaster recovery (DR) of modern computer systems. It is parameterized as 1+n redundancy in which one of n cold, warm, or hot spares takes over upon the failure of the single active primary. A range of values of n are common, from 1+1 for disaster recovery, 1+2 to maintain redundancy during lengthy repairs and upgrades, (1+1) + (1+1) (effectively, 1+3) multi-site designs in which an 1+1 standby site backs up another 1+1 primary site, to general 1+n systems [2] [3].

Unfortunately, inefficient utilization of resources is standby redundancy's Achilles heel. Keeping the standbys idle (except for synchronizing state) during normal operation results in 50 % (for 1+1 systems) or more of a system's peak capacity being wasted. The active-active systems in which all replicas are utilized during normal operation can eliminate wastage, but such designs are practical mostly when replicas are stateless, or contain state that can be shared (e.g., key-value stores). For most other stateful systems, standby redundancy continues to be a viable option despite its limitations.

In this paper, we make the observation that the idle wastage of standby redundancy could be mostly eliminated by overbooking standbys from multiple independent cloud applications onto the same hosting resources. However, while doing so, we must minimize the chance that multiple standbys should be activated at the same time on the same host due to common mode failure affecting their active primaries. This situation would lead to performance degradation or even inability of the standby to take the active role due to resource shortage (i.e., losing its entitlement). Fortunately, unlike traditional DR providers who do not know the failure modes of their customers, cloud computing platforms have visibility and control over virtual machine (VM) placement and failure modes. Thus, through careful placement of active and standby VMs followed by judicious selection of which standby VM to activate, a cloud platform can minimize the probability of concurrent activation of multiple standby VMs on the same host. Note that such redundancy-driven overbooking can only be done by the platform – tenants do not have the cross application visibility nor the infrastructure topology visibility necessary.

We propose ShadowBox, a novel multi-datacenter VM placement scheduler that uses these observations to overbook standby VMs from multiple independent applications efficiently with minimal loss in their resource entitlement guarantees and availability by co-optimizing the datacenter, rack, and server level placement of active and standby VMs. On active VM failure, ShadowBox can also select which standby VM to activate if more than one standby VMs are available. Using the physical topology of the hierarchical cloud infrastructure that consists of servers, racks, and data centers, ShadowBox can identify common mode failures that can impact multiple VMs at the same time. Importantly, ShadowBox uses additional application information compared to traditional cloud schedulers; it uses metadata from the applications identifying which VMs are part of an active-standby cluster and the entitlement assurance rate (EAR) for the application. The metadata can additionally include the placement diversity level (i.e., either cross-servers, cross-racks, and cross-data centers replications that lead to different levels of availability).

Despite its close philosophical relationship to the vast literature on parity-coded storage systems such as [4], we are unaware of any other work that has explored these ideas in the context of VM scheduling on public or private clouds. Previous cloud overbooking studies such as [5] and [6] have exclusively focused on mixing complementary workloads or leveraging time-of-day workload patterns, but these methods are often brittle due to inherent workload unpredictability (especially, the initial application placement time) and are

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

usually shunned by public cloud providers. In contrast, ShadowBox's redundancy-driven overbooking has predictable worst case behavior (i.e., resource failure probabilities) that can be used for the cloud providers to guarantee a certain service level agreement (SLA) for their customers.

Dealing with the redundancy with shared resources has also been tackled in shared-backup path protection in optical networks, such as in [7] and [8]. However, the nature of the hierarchical cloud structure leads to the complex dependencies between host failures that make the problem more challenging.

Finally, while ShadowBox is applicable to a traditional cloud infrastructure having a small number of large sites or even within a single cloud site, it becomes an especially powerful way to achieve DR in a network distributed cloud, in which the cloud provider has hundreds of small (edge) sites. We evaluate ShadowBox on a distributed cloud topology inspired by a Tier-1 network provider's cloud, and show that it is able to eliminate more than 2.5 times of the wasted standby resources, compared to the state-of-the-art approaches, while preserving the applications' resource entitlement requirements.

This paper makes the following contributions:

- We propose the core concept of redundancy-driven overbooking in the cloud environment, and introduce ShadowBox, an architecture enabling such overbooking (Section II).
- We propose placement rules (Section IV-A) that maximize the cloud resource utilization, while minimizing the impact on applications' entitlements.
- We develop optimization algorithms (Section IV-B and IV-C) to perform redundancy-aware standby VM placement and activation across multiple cloud datacenters in a hierarchical infrastructure.
- We implement ShadowBox and integrate with OpenStack (Section V) to show its effectiveness.
- We evaluate ShadowBox on a realistic cloud topology and show a dramatic decrease of resource wastage (Section VI).

II. System Design

ShadowBox is designed to provide Infrastructure-as-a-Service (IaaS) in multi-tenancy environments. As shown in Fig. 1, we consider a common architecture which is broadly applicable for any cloud management systems. Specifically, ShadowBox puts application placement requests from tenants to the request queue and schedules them in the cloud, based on the provisioning policy specified in each request while monitoring the cloud resource status such as server, rack and datacenter failures.

However, to allow the cloud provider to overbook standby VMs on behalf of its users while assuring the expected availability for active-standby applications, we need to extend the placement request API. Fig. 1 highlights the API that can be expressed by cloud tenants. We simplify the API to make it easier to understand, while it is sufficient for the main problem tackled in this paper. For the redundancy-aware placement, in addition to the number of standbys for the

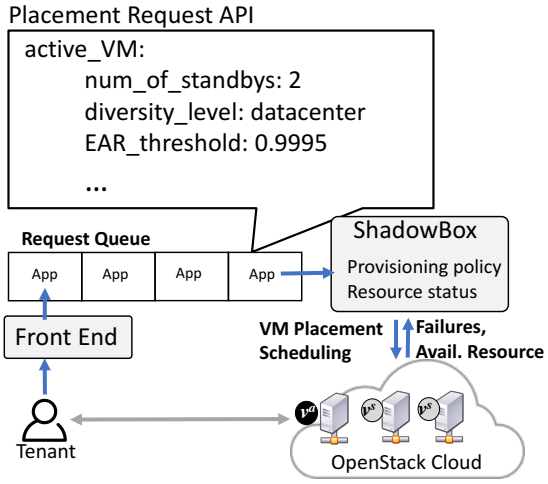

Fig. 1: ShadowBox architecture.

active VM, we emphasize that the API includes the expected availability assurance for the application, which is converted to the entitlement assurance rate (*EAR*) threshold. The EAR threshold is about how much the cloud tenant expect the availability and entitlement assurance for the given application when standbys are overbooked, and the formal definition is described in Section III-B. In our private cloud setup, tenants can estimate the application's availability based on the number of standbys and diversity. This is similar to the Amazon EC2 cloud, where tenants place their VMs in different availability zones (or datacenters) with the expectation of certain guaranteed availability [9]. However, this is typically done without the standby information and therefore, with no-overbooking. ShadowBox uses overbooking and checks how much it is guaranteed for tenants to obtain full resources upon resource failures to determine where to place or which standby VM to activate (e.g., a standby VM can be fully activated when its active VM fails). Moreover, ShadowBox allows tenants to specify even finer-grained diversity levels. For example, placing VMs on the same server or rack will reduce network latency but it hurts its availability upon failures.

III. Problem Statement

We assume that the placement of a cloud application is requested with one active and n standby VMs (i.e., 1+n active-standbys), and total M applications are requested to the cloud. Formally, the set of all VMs of an application A_i, $(i \in 1, ..., M)$ is defined as $A_i = \{v^a, v_1^s, v_2^s, ..., v_n^s\}$, where v^a is the active VM and v_j^s is the jth standby. All VMs of the application are placed in physical hosts, which are configured in a hierarchical cloud infrastructure of servers, racks, and datacenters. In the hierarchical cloud infrastructure illustrated in Fig. 2, a failure cascades down from the top-level. That is, when a datacenter (or a rack) fails, all the servers in that datacenter (or rack) also fail. In this structure, each server is connected to other servers through the top of rack switch (ToR) in the same datacenter or through core switches across datacenters.

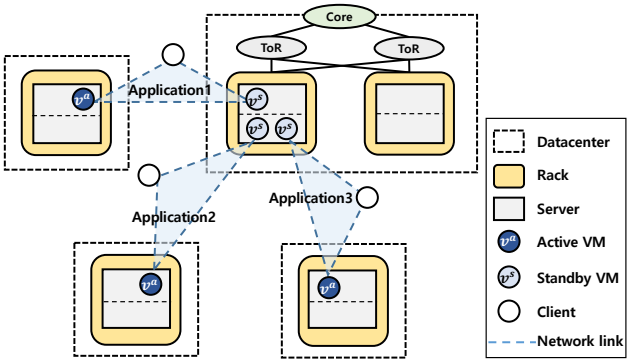

Fig. 2: Example of three 1+1 application placements.

Fig. 2 depicts the cloud infrastructure with four datacenters, where three 1+1 applications are placed. All three standby VMs are co-placed in a server, while their active VMs are placed in three different datacenters. In this example, we assume that each server has the capacity of 2 CPU slots[1] and each slot can be used for either one active VM or 2 standby VMs. Each client communicates with the active VM (v^a) of the application or the standby (v_j^s) if its v^a is not available.

A. Resource utilization

To utilize given cloud resources, our goal is to place as many applications as possible. To this end, we overbook m standby VMs of m different applications into one slot with a certain overbooking rate R.[2] Note that each active v^a is assumed to use the whole slot (i.e., no overbooking).

In the case of Fig. 2, R is set to 2, so up to 2 standby VMs can be placed in a slot and up to 4 standby VMs can be placed in a server. The number of slots used to place all 3 active and 3 standby VMs is 5, while without overbooking 6 slots would be required. To quantify this, the resource utilization is defined as follows.

Definition 1. Given the M' number of standby VMs of all applications placed in the cloud, the total number of slots used for all such VMs is measured as S. Then, the resource utilization is defined as $\frac{M'}{S}$ (i.e., the number of VMs allocated per slot).

In fact, the total number of standby VMs, which can be overbooked in a server, is restricted by failure probabilities of the cloud infrastructure (i.e., servers, racks, and datacenters). Specifically, when the servers hosting the original active VMs fail simultaneously, we need a certain number of standby VMs to be able to work as active VMs. Those newly activated VMs, however, must not be overbooked, since any active VM needs one whole slot (i.e., entitlement). We define the available number of slots for standby activations in a server s as B_s in the rest of this paper.

In the example shown in Fig. 2, we set B_s to 1, indicating that only one standby VM v^s can be fully activated at a time, in case of the corresponding active VM v^a failure. Thus, up to 3 standby VMs can be placed in a server.

B. Application EAR

An application A_i is not available if the server of an active VM (v^a) fails along with all servers running its standby VMs. Moreover, it is not fully available if the server of an active VM v^a fails and not one of the standby VMs can be assured for its resource entitlement due to the overbooking. For instance, Fig. 2 shows the example in which only one standby VM can be fully activated at a time (B_s is 1) and 3 standby VMs are co-placed in a server. In this case, if two servers hosting two active VMs fail simultaneously, not both standby VMs can be activated. In other words, not both can be assured for their entitlement. To quantify this, the entitlement assurance rate (EAR) of an application is defined as follows, and its formal equation will be described in Section IV-A3.

Definition 2. EAR of application A_i is the probability that its active VM (v^a) or one of its standby VMs (v_j^s) is assured for the entitlement for each time period upon failures.

We focus on improving EAR when we overbook the standby VMs. Intuitively, it is reasonable to avoid possible contention on a slot between applications when B_s VMs fail (or, B_s standby VMs activate) simultaneously. Basically, our approach is to ensure independent failures between applications by reducing the placement overlap among standby VMs of different applications.

C. Placement problem

Our approach aims to maximize the resource utilization (Def. 1) in the hierarchical cloud infrastructure, while meeting the EAR (Def. 2) threshold T_i of each application A_i. Importantly, EAR is affected by the number of available slots per server to activate standby VMs (i.e., B_s), the placement overlap, and the diversity in the hierarchical cloud infrastructure. This will be precisely analyzed in the following section.

We assume that the standby VM v_j^s will run active only until the original active VM v^a is repaired. We also assume that there is some upper bound on the number of standby VMs overbooked on a server, since they consume some amount of CPU cycles and bandwidth to sync the states with their active VMs. This will be further discussed in Section V.

IV. ShadowBox Approach

In this section, we analyze ShowdowBox's placement rules and their implications on the EAR and resource utilization. Then, we describe (a) a VM placement algorithm that takes into account different applications' EARs, and (b) an algorithm that decides which standby VM(s) to activate upon an application's active VM failure.

A. Placement rules

For the analysis in this section, we first define the failure probability of a cloud component as $p_x = \frac{\text{MTTR}}{\text{MTBF}_x+\text{MTTR}}$, where MTBF_x is the mean time between failure for x level in the

[1]While CPU overbooking is most common, other resources such as memory and storage are also candidates for overbooking.

[2]Other terms such as oversubscription and overcommit which have been used in other literature have the same meaning.

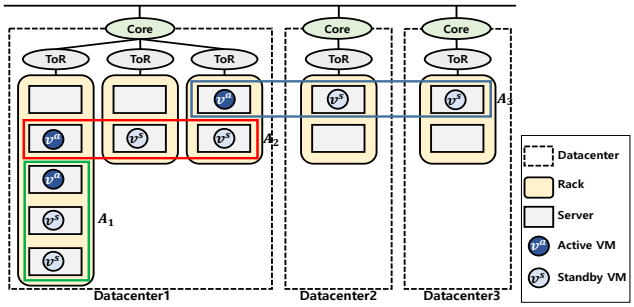

Fig. 3: Examples of different placement diversities (datacenter, rack, and server levels).

cloud hierarchy (i.e., x is a datacenter, rack, or server) and MTTR is the mean time to repair.

1) Placement diversity: Given the diversity requirement in the ShadowBox API (Section II), our approach places VMs of the same application into different cloud components to increase the chance of at least one VM of the application still running upon each failure. Fig. 3 illustrates three different diversity placements of 1+2 (i.e., 1 active and 2 standbys) applications. All VMs of application A_1 are placed in different servers in a rack for the server level diversity. Similarly, All VMs of A_2 are placed in different racks in Datacenter$_1$, and all VMs of A_3 are in different datacenters for the rack and datacenter level diversities, respectively.

While we can assume that failures are independent in the same level of the cloud hierarchy, servers' failures are *dependent across levels*. For example, if Datacenter$_1$ fails, all racks and servers in the datacenter are not accessible. In Fig. 3, A_1 and A_2 are considered not available if Datacenter$_1$ fails. Meanwhile, A_3 is still available, since one of its standby VMs can be activated in either Datacenter$_2$ or Datacenter$_3$.

Considering this *dependency*, we obtain the probability that each application cannot be assured for the entitlement (i.e., $1-EAR$) based on each cloud infrastructure's failure probability. For instance, the probability of a single VM failure in the cloud hierarchy is $p_d+p'_d(p_r+p'_r p_s)$, where p_d, p_r, and p_s are failure probabilities of datacenter, rack, and server, respectively, and $p'_d = 1-p_d$ and $p'_r = 1-p_r$. This implies that a single VM failure happens if either the host datacenter, the host rack, or the host server fails.

By extending this, now we calculate the probabilities of an application not being available for the datacenter (γ_d), rack (γ_r), and server (γ_s) level diversity, respectively.

The datacenter diversity rule for the (1+n) application will place 1 VM in each datacenter (1 active VM on 1 datacenter and n standby VMs in n datacenters). Therefore γ_d corresponds to the cases of failure of all datacenters, all racks, or all servers, where each VM is hosted.

$$\gamma_d = (p_d + p'_d(p_r + p'_r p_s))^{1+n} \tag{1}$$

With the rack level diversity, γ_r is either (a) the datacenter fails or (b) 1+n racks (or 1+n servers) hosting all VMs fail.

$$\gamma_r = p_d + p'_d(p_r + p'_r p_s)^{1+n} \tag{2}$$

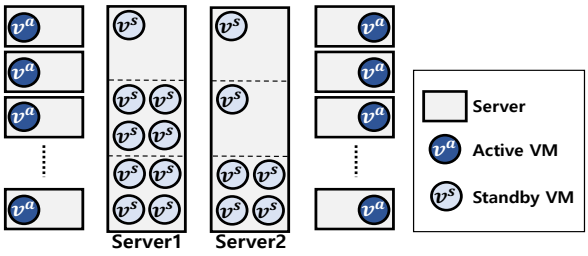

Fig. 4: Examples of 1+1 applications. Two servers in the middle has different number of slots for standby activations ($B_1 = 1$ for Server$_1$ and $B_2 = 2$ for Server$_2$).

With the server level diversity, γ_s is either (a) the datacenter fails or (b) the rack fails, or (c) all 1+n servers fail.

$$\gamma_s = p_d + p'_d(p_r + p'_r p_s^{1+n}) \tag{3}$$

2) Standby overbooking: Next, we analyze the impact of overbooking on the resource utilization and EAR.

Assume the server s has total C cores and the overbooking rate per core is R. With B_s and $B_s^{\max}$, where B_s is the number of standby activations and $B_s^{\max}$ is the maximum number of overbooked VMs, the following should satisfy

$$B_s + \frac{B_s^{\max} - B_s}{R} \leq C. \tag{4}$$

Therefore,

$$B_s^{\max} \leq CR - B_s(R-1). \tag{5}$$

Consequently, the resource utilization (based on $B_s^{\max}$) linearly decreases as allowing more standby VMs to be activated (B_s) (while it increases the EAR).

Fig. 4 illustrates this tradeoff with the two servers hosting all standby VMs for 1+1 applications. Assume R is set to 4, and C is 3. Based on (5), Server$_1$ can host up to 9 standby VMs with $B_1 = 1$. In this case, Server$_1$ can assure the entitlement (i.e., a whole slot) of only one standby VM (v_j^s) upon active VM (v^a) failure. On the other hand, Server$_2$ can host only up to 6 standby VMs with $B_2 = 2$.

We can consider two possible failure cases for the 1+1 application A_i in Fig. 4. The first case is that the server hosting v^a of A_i fails and the server hosting its v^s (e.g., Server$_1$) also fails. The second case is that when the server hosting v^a of A_i along with the host of v^a of another 1+1 application fails, where the standby VMs of both applications are hosted in Server$_1$. Then more than two standby VMs in Server$_1$ will *contend* to be activated due to $B_1 = 1$.

The probability of the first case depends on the placement diversity of A_i and γ_d in (1), γ_r in (2), and γ_s in (3). If we assume application A_i has n_i number of VMs, with no-overlap assumption, the contention probability in the second case for A_i is approximately

$$\gamma = p'_d p'_r p'_s \; q_i^{n_i} \sum_{l=1}^{\binom{B-1}{B_s}} \prod_{A_y \in S_l} q_y^{n_y} \prod_{A_y \notin S_l} (1 - q_y^{n_y}). \tag{6}$$

The term $p'_d p'_r p'_s$ is the probability that the server hosting the standby VM (v^s) of A_i works, and $q_i^{n_i}$ is the probability that

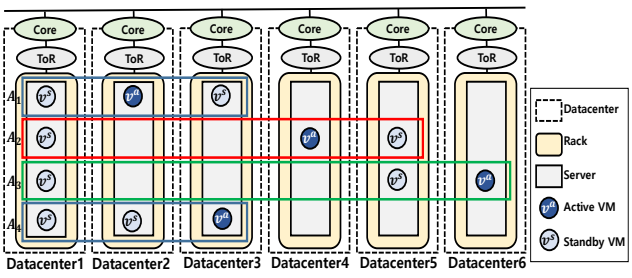

Fig. 5: Example of placement overlaps among four 1+2 applications across datacenters.

servers hosting the remaining n_i VMs of A_i fail, knowing that the host server of (v^s) is working. Here, the probability q_i depends on the diversity level of A_i. That is, if the diversity is server level, n_i servers hosting n_i VMs of A_i are in the same datacenter and the same rack with v^s, which is not failed with probability $p'_d p'_r$. Thus, q_i is p_s. Similarly, in the rack level diversity, q_i is $p_r + p'_r p_s$, and in the datacenter level, is $p_d + p'_d(p_r + p'_r p_s)$. To find the probability that B_s out of other $B-1$ applications need their standby VMs, we need to consider all $\binom{B-1}{B_s}$ cases. In each case l, S_l is the set of applications that need their VM. The lower bound on the probability that all applications in S_l need their standby VM is $\prod_{A_y \in S_l} q_y^{n_y}$. Similarly, the lower bound for the probability that the remaining applications do not need their VMs is $\prod_{A_y \notin S_l}(1-q_y^{n_y})$. Note that we could consider that more than B_s standby VMs contend for B_s slots caused by more VM failures; however, such probabilities are negligible compared to (6).

3) Placement overlap: Finally, we analyze the impact of the placement overlap on the application EAR. First, the placement overlap delta for two application A_i and A_j, $\delta_{i,j}$, is the number of VMs of application A_i, which do not share a host server with any VMs of application A_j. The placement overlap delta for an application A_i is defined as the minimum of the placement overlap delta of application A_i and all other applications $A_j, j \neq i$. In other words, $\Delta_i = \min \delta_{i,j}, \forall j \neq i$.

In Fig. 5, we illustrate how four 1+2 applications' (A_1 to A_4) placements are overlapped with each other in 6 datacenters with the datacenter level diversity. As it can be seen in this figure, $\delta_{1,2} = \delta_{1,3} = 2$, and $\delta_{1,4} = 0$, so $\Delta_1 = 0$.

Let's assume $B_s = 1$ (i.e., only 1 standby VM can be activated at a time) for the server in Datacenter$_1$ in Fig. 5. The failure on entitlement assurance of application A_1's standby VM caused by another application activating its standby VMs depends on Δ_1. Suppose that the server in the Datacenter$_1$ is working, but 3 servers in the Datacenter$_2$ to Datacenter$_4$ are failed. The standby VM of A_1 in Datacenter$_1$ will contend with the standby VM of A_4 to be activated, since $\delta_{1,4} = 0$. On the other hand, in this situation, the standby VM of A_1 does not contend with the standby VM of A_2 in this server, since $\delta_{1,2} = 2$ and another standby VM of A_2 can be activated in Datacenter$_5$.

In our proposed placement algorithm, when placing application A_1 in each server s, the placement overlap delta of application A_1 and all other already placed applications in this server ($c(s)$) will be considered. Therefore, for application A_i we can define parameter $\Delta^{(i)}(s)$ for each server s based on

$$\Delta^{(i)}(s) = \min_{j \in c(s)} \delta_{i,j}. \quad (7)$$

Our proposed placement algorithm can consider $\Delta^{(i)}(s)$ as a metric to rank different candidate servers, and it only considers the servers with $\Delta^{(i)}(s)$ that meet a threshold. Obviously, when there is overlap in placement of standby VMs, the probability of contention of application A_i, (γ in (6), will increase.

Algorithm 1 Redundancy-aware VM placement.

```
1: A; # final placements of all applications
2: Δ* ← ∅; # Δ meets EAR threshold T
3: B* ← ∅; # no. slots meets EAR threshold T
4: (Δ*, B*) ← DeltaAndSlots(requirements)
5: for each v_i ∈ V # for each VM of given application
6:    for each s_j ∈ S # for each server
7:       if Diversity(A, v_i, s_j) not meet or
8:          CurrDelta(A, s_j) < Δ* or
9:          B_{s_j} < B* or # available slots of s_j less than B*
10:         Cap(B_{s_j}, R) > C then # s_j has no capacity
11:         Remove(s_j, S);
12:   if S = ∅ then break; # no server found
13:   S* ← Sort(S); # sort servers by Δ and B_s
14:   A ← A ∪ (S*[0] ← v_i); # get the largest Δ and B_s
```

B. VM placement

Although many known algorithms for the optimal VM placement have been proposed including [10] [11] [12] [13], our VM placement algorithm additionally deals with the 1+n application's entitlement assurance rate (EAR) as well as the resource utilization in a hierarchical cloud infrastructure. Specifically, our algorithm aims to overbook as many standby VMs in the hosting servers as possible, while taking into account its destructive effect on the application EAR.

To ensure the EAR threshold requirement, our algorithm attempts to keep the high values of both the number of available slots for standby activations (B_s) and the placement overlap delta (Δ) to avoid simultaneous activations of co-placed standby VMs. However, once many such 1+n applications are placed in the multi-tenancy cloud, reducing B_s and Δ are inevitable. Consequently, our algorithm places VMs of each arrived application (i.e., on a first come, first served basis), while reducing their placement overlap with already placed VMs of the other applications as much as possible, and choosing a server, which has the largest capacity among the candidate servers. B_s and Δ decrease as more applications are placed in the cloud infrastructure, until there are no more servers left to guarantee the EAR threshold of a new application.

Algorithm 1 summarizes our greedy VM placement procedure. In line 4, it first computes the minimum required placement delta (Δ^*) and available slots for activating each standby VM (B^*) to meet the given EAR threshold using equations (1), (2), (3), and (6). For each standby VM, the algorithm filters out servers that do not meet constraints in

Algorithm 2 Standby VM selection for activation.

```
1: A^f ← ∅; # a set of failed apps due to host failures
2: for each s_i ∈ S^f # for each failed server
3:    if s_i includes v^a s then # if s_i has active VMs
4:       A^f ← A^f ∪ applications of v^a s
5: S ← Avail(A^f); # get servers hosting standbys of A^f
6: S ← Sort(S); # sort servers by available slots
7: A^f ← Sort(A^f); # sort failed apps by EAR threshold
8: for each A_i ∈ A^f # for each failed app
9:    for each s_j ∈ S # for each server hosting standbys
10:      if s_j has any v^s of A_i then # if s_j has a standby of A_i
11:         ActivateStandby(v^s, s_j);
12:         DeductAvailableSlot(s_j);
13:         break;
14:   if Σ_{s_j∈S} B_{s_j} = 0 then break;
```

lines 7–10. It checks if the VM placement into s_j violates the diversity requirement (see Section IV-A1) with the prior placements in A (i.e., `Diversity` in line 7). Then, it checks if choosing s_j for the current v_i of an application does not meet the minimum required placement delta Δ^* in line 8. It also checks if the available slots for activating standby VMs of s_j is more than or equal to the minimum required slots B^* in line 9. Finally, it checks the capacity of s_j (`Cap` in line 10) as defined in (4). Note that this algorithm does not include the placement of active VMs, which is similar to the standby VM placement except it does not check the available slots for activation (i.e., line 9). If it cannot find out any candidate server, it exits. Otherwise, it chooses a server that has the largest Δ and B_s from the candidate list (lines 13–14).

The algorithm can also include other constraints such as network latency and bandwidth, if those requirements are given. For the network latency, the algorithm checks the latency requirement considering the latency between datacenters. We assume the latency will be met within the datacenters. ShadowBox also keeps monitoring the available bandwidths of each link and the minimum of all links between two nodes to check with the bandwidth requirements.

Although we use a greedy algorithm, the experiment results show that, compared to the other alternative approaches (Section VI), our proposed algorithm improves the resource utilization significantly, while keeping EAR.

C. Standby selection for activation

When a cloud component fails, many 1+n applications may be affected. For example, a datacenter failure triggers the situation that all racks and servers in the datacenter are inaccessible, making all active VMs hosted in those servers not available. Once ShadowBox captures such failed applications at runtime, it finds the servers of corresponding standby VMs of those failed applications. It activates them as active VMs and continue running those standby VMs until the repair is done. In fact, selecting standby VMs for activation may not be trivial, if many applications are affected from the failures and many standby VMs are overbooked.

Algorithm 2 presents a heuristic standby selection procedure. ShadowBox first captures all the affected applications (lines 1–4) and available servers which are hosting standby VMs of affected applications in line 5. Then, it sorts those servers by the number of available slots for standby activations (B_s) in descending order in line 6. It also sorts the affected applications by EAR threshold in descending order in line 7 (i.e., dealing with the most critical application first). For each affected application, it attempts to get a server that hosts any one of standby VM of the application in lines 8–10. If a server is found, the algorithm activates the standby VM in the server in line 11, and decrease the number of available slots of the server in line 12. ShadowBox stops its procedure once there is no available slots left in servers in line 14.

ShadowBox can validate if all failed applications can be assured for their entitlements by comparing the total number of available slots for standby activations of all servers, where standby VMs of failed applications are hosted, to the number of failed applications (i.e., $\sum_{s_j \in S} B_{s_j} \geq |A^f|$, where $|A^f|$ is the number of failed applications, and S is the set of servers hosting standby VMs).

V. Implementation

We have implemented ShadowBox as a redundancy-aware resource scheduler on top of OpenStack[3], which is a popular open source cloud platform. We have used the OpenStack Heat template[4] as the placement request format, and added key/value pairs to include additional information (i.e., the number of standbys, EAR threshold, diversity as shown in Fig. 1). The OpenStack platform has been deployed in each datacenter to orchestrate datacenter resources. It also periodically reports resource status to ShadowBox. The resource failures in each datacenter are also monitored and reported using Nagios[5].

To control the rack and server level diversities within a datacenter, we have added our own scheduling filter into the OpenStack Nova filter scheduler[6]. Our filter can select a server among all candidate servers based on the diversity requirements. We have also implemented a server agent module that runs on each server to control the overbooking per core using *cgroup* mechanism[7] in Linux. Basically, the agent controls the overbooking rate per core by pinning a given vCPU to a CPU core. For example, if the overbooking rate R is 4, it can allocate (pin) 4 vCPUs per core.

To determine the maximum number of standby VMs per server that consistently communicate with its active VM for state update, we have performed offline measurements using the virtual proxy server backed with a small database, which is one of the most CPU-intensive stateful applications in our private cloud. We have found that up to 1 Mbps bandwidth and $R = 4$ are enough for each standby VM. Then, we conservatively set the upper bound of overbooking standby VMs to 30 for ShadowBox to use up to 8 cores per server for the redundancy-aware placements. The upper bound can dynamically be changed when we consider various

[3]https://www.openstack.org
[4]https://docs.openstack.org/developer/heat/template_guide/hot_spec.html
[5]https://www.nagios.com
[6]https://docs.openstack.org/developer/nova/filter_scheduler.html
[7]https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

applications that have different performance characteristics. We are planning to employ other applications as our on-going work, but we consider the CPU-intensive application for the simplicity in this paper.

VI. Evaluation

For the evaluation, we experiment ShadowBox in our private distributed cloud. We also use simulations to accommodate failure injections for the long period (e.g., 20 years).

A. Experimental setup

We use our hierarchical cloud infrastructure consisting of 10 datacenters in the edge for the evaluation. There are three different datacenter types that have different resource capacities as shown in Table I.

TABLE I: The hierarchical cloud infrastructure.

datacenter type	large	medium	small
number of datacenters	1	6	63
number of racks per datacenter	30	8	4
number of servers per rack	8	8	8

All links between datacenters have 40 Gbps bandwidth, while all internal bandwidths (i.e., between top-of-rack (ToR) and core switch) in each datacenter are set to 100 Gbps. Each server has 8 CPU cores, 32 GB memory and 2 TB disk. Unless mentioned otherwise, the default overbooking rate per core is set to 4 (i.e., up to 4 single vCPU VMs can share a core).

B. Results

We show the results with two aspects, the resource utilization and the EARs in two different failure scenarios (i.e., Optimistic and Rampage). The **Optimistic** scenario sets MTTR and MTBFs based on hardware documentations used in our private cloud, while the **Rampage** scenario sets these values more realistically by considering software crashes. And two different threshold types (i.e., Stingy and Generous). **Generous** threshold has less EAR than **Stingy** for higher utilization. We also compare ShadowBox with two alternative approaches (i.e., No-Overbook and Overbook-Blindly). **No-Overbook** approach will not overbook standby VMs on servers, but will place VMs by following the diversity requirements. On the other hand, **Overbook-Blindly** approach will overbook standby VMs but will not consider placement delta (Δ) nor the number of standby activations upon failure (B_s).

1) Resource utilization: Fig. 6 shows the resource utilization (as defined in Def. 1) of ShadowBox and Overbook-Blindly is more than two times better than No-Overbook approach. Obviously, the resource utilization of No-Overbook is 1 because it does not overbook standby VMs, and each standby VM consumes a whole core. This experiment also shows the impact of the placement delta Δ on the resource utilization. The resource utilization of the Overbook-Blindly approach is higher than ShadowBox since it overbooks as long as it finds available cores while ShadowBox should leave at least 1 core per server (Δ) available for possible failures. In the later experiments, we show that this Overbook-Blindly's high resource utilization comes at the cost of EAR degradation.

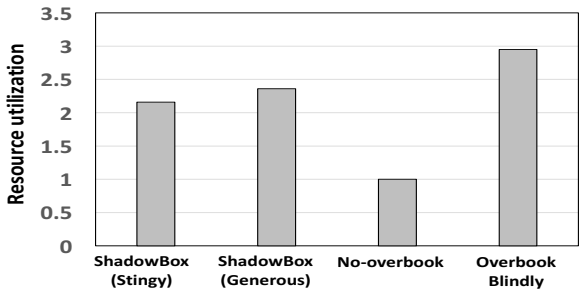

Fig. 6: Resource utilization comparison between ShadowBox and two different alternative approaches.

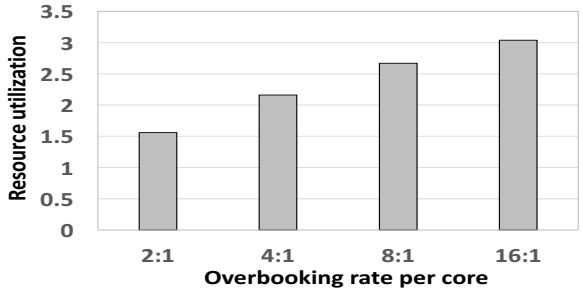

Fig. 7: Resource utilization with different overbooking rates per core.

To see the impact of the overbooking rate per core, we measure the resource utilization by increasing the overbooking rate. Intuitively, we expect the resource utilization increases as a core has more capacity for overbooking. Fig. 7 shows the result. We measure the resource utilization of ShadowBox with Stingy threshold. As expected, the resource utilization increases when doubling the overbooking rate. Interestingly, it does not increase dramatically, and even with 16:1, it reaches only around 3. This is mainly because ShadowBox carefully and conservatively manages the placement delta Δ to keep meeting the EAR thresholds. It tries to avoid the overlap as much as possible, and once it starts to overlap applications, it checks the thresholds by estimating the failure probability as described in Section IV-A3.

2) Application EARs: We measure the application EARs using the stochastic failure simulations as failures are not frequent enough over a short period of time, and it would be difficult to collect a statistically significant number of failures in such a cloud infrastructure.

Table II shows MTTR and MTBFs for the two failure scenarios (i.e., Optimistic and Rampage), where $MTBF_d$ is the datacenter MTBF, $MTBF_r$ is the rack MTBF, and $MTBF_s$ is the server MTBF.

TABLE II: Two failure scenarios (hours).

	$MTTR$	$MTBF_d$	$MTBF_r$	$MTBF_s$
Optimistic	54	130,000	80,000	30,000
Rampage	54	26,000	16,000	6,000

TABLE III: EAR thresholds under two failure scenarios with the expected availabilities

Redundancy & Diversity	Expected Availability		Generous EAR Threshold		Stingy EAR Threshold	
	Optimistic	Rampage	Optimistic	Rampage	Optimistic	Rampage
1+3 Server	0.998	0.994	0.995	0.985	0.997	0.990
1+4 Server	0.999	0.995	0.996	0.986	0.998	0.992
1+2 Rack	0.9996	0.998	0.999	0.990	0.9995	0.996
1+3 Rack	0.9999	0.999	0.9995	0.995	0.9996	0.998
1+1 Datacenter (DC)	0.99999	0.9998	0.99995	0.999	0.99997	0.9996
1+2 Datacenter (DC)	0.9999999	0.999997	0.9999995	0.99999	0.9999997	0.999995

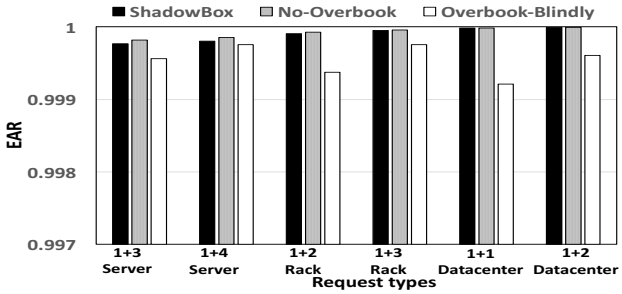

Fig. 8: Application EAR comparison between ShadowBox and two alternative approaches in the Optimistic failure scenario.

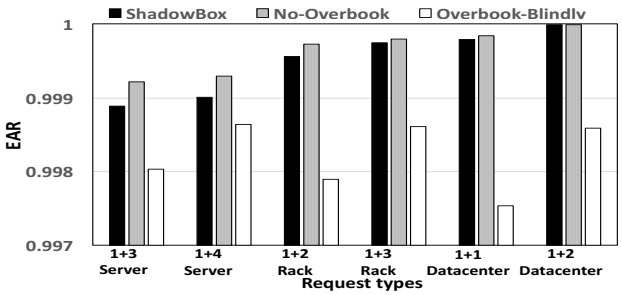

Fig. 9: Application EAR comparison between ShadowBox and two alternative approaches in Rampage failure scenario.

Fig. 8 shows the comparison results in Optimistic failure scenario. Overall, the differences among ShadowBox and the other two approaches are relatively small (except Overbook-Blindly). However, we can observe some important facts from the results. First, ShadowBox achieves competitive EAR with No-Overbook, even though ShadowBox overbooks VMs. This indicates the importance of choosing Δ in VM placement. By increasing Δ, ShadowBox increases the failure independency among applications.

As expected, the No-Overbook approach achieves the best EAR but with a cost of resource utilization. We can see ShadowBox result is close to No-Overbook and even closer in the higher diversity levels. This means ShadowBox can achieve the high EAR with better resource utilization.

In Rampage failure scenario in Fig. 9, the trend is same with Optimistic, but the difference between the approaches is clearly seen, especially, in lower diversity levels. ShadowBox still shows competitive EARs with No-Overbook while the gap is larger than in Optimistic failure scenario. and obviously better than Overbook-Blindly approach. Especially, in the higher diversity level, it is almost same with the No-Overbook approach. The gap in 1+3 server level diversity is the largest

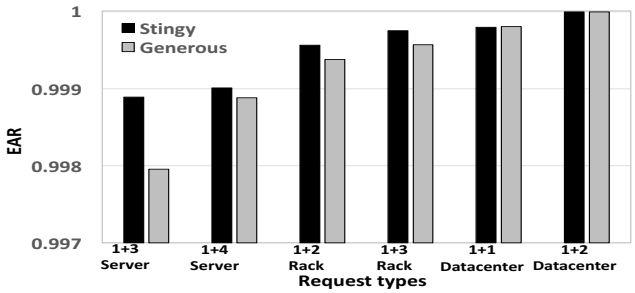

Fig. 10: Application EARs of two different EAR thresholds in Rampage failure scenario.

between ShadowBox and No-Overbook approach. However, the average EARs ShadowBox achieves are more than the expected availability (i.e., 0.994 obtained by (3)) as well as the EAR threshold (i.e., 0.990), thanks to the conservative placement approach of ShadowBox. Note that Overbook-Blindly even does not meet the expected availabilities because of many simultaneous resource failures making many applications being not available.

Table III shows the expected availabilities that are computed with (1) (2), and (3). ShadowBox attempts to meet EAR thresholds while placing VMs. For example, for the given 1+3 application with the server level diversity, its expected availability is 0.998 in Optimistic failure scenario (obtained by (3)). By overbooking 3 standby VMs on some servers, it also expects an additional EAR degradation up to 0.003 in the Generous threshold case. Therefore, its adjusted expected EAR is 0.995 (0.998 - 0.003).

Fig. 10 shows the difference between two threshold types (i.e., Stingy and Generous) in the Rampage failure scenario. This indicates the same trends discussed in the above experiments. The average EARs ShadowBox achieves increases as the diversity level and the number of standby VMs increase for both Stingy and Generous thresholds. The average EARs meet even the expected availabilities in lower diversity levels (i.e., server and rack levels), but does not in the datacenter diversity cases. However, when considering the EAR thresholds, ShadowBox can meet the adjusted EARs. Stingy threshold leads to the higher EAR results because ShadowBox constrains the overbooking more in this case than in Generous case by more avoiding the placement overlap and leaving more number of slots to be used for activating standby VMs.

VII. Related Work

High Availability (HA) has been considered as one of challenging obstacles to the growth of cloud computing [14].

Redundancy and checkpointing. One of well-known mechanisms is the redundancy with either active-active or active-standby replication, which ShadowBox also tackles in this paper. Remus [15] offers a method to asynchronously propagate changed state to backup hosts. Another well-known mechanism is the checkpointing that is a technique to add fault tolerance into systems basically by saving snapshot of system state so restarting from that point in case of failure [16] [17] [18] [19]. ShadowBox can employ those mechanisms to improve the state updates between active and standby VMs.

Availability and performance tradeoff. Jung et al. [10] has proposed an algorithm to address the tradeoff between the application performance, especially network latency between replicas, and the application availability. Zhou et al. [20] proposes a VM placement optimization by placing standby VMs close to active VMs to reduce network bandwidth consumption, while ShadowBox aims at maximizing resource utilization while ensuring the high entitlement assurance and availability. Sharma et al. [21] combines energy efficiency and reliability for efficient resource provisioning and studies the tradeoff between them.

VIII. Conclusion

Cloud applications are often suffer from unpredictable cloud resource failures. The redundancy with standby VMs has been popularly used to address the high availability problem, but it could decrease the cloud resource utilization if it is not properly used. In this paper, we propose ShadowBox, which runs by the cloud provider and allows cloud tenants to specify their needs for the availability and the entitlement assurance of their applications. ShadowBox then overbooks multiple standby VMs on servers to improve the cloud resource utilization, while avoiding potential simultaneous activations of those standby VMs upon resource failures, to satisfy cloud tenants' availability and entitlement needs on their applications. We have addressed the tradeoff between the overbooking and entitlement assurance rate (EAR) with the algorithms that control the number of VMs of different applications to be co-placed in servers. Evaluations in our private cloud show that it significantly improve the resource utilization with a minimal loss of EAR.

IX. Acknowledgements

This work was partially supported by the NSF under Grant CNS-1525435.

References

[1] A. Avizienis, "Design of fault-tolerant computers," in *Proc. of Fall Joint Computer Conference*, 1967, pp. 733–743.

[2] R. van Renesse and F. B. Schneider, "Chain replication for supporting high throughput and availability," in *Proc. of USENIX Opearting Systems Design & Implementation*, 2004.

[3] "Mongodb chained replication," https://docs.mongodb.com/v3.2/tutorial/manage-chained-replication/.

[4] K. V. Rashimi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the Facebook warehouse cluster," in *Proc. of USENIX Hot Storage*, 2013.

[5] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: incorporating time-varying network reservations in datacenters," in *Proc. of ACM SIGCOMM*, 2012, pp. 199–210.

[6] 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 *Proc. of USENIX Operating Systems Design and Implementation*, 2016, pp. 755–770.

[7] D. A. Mello, J. U. Pelegrini, R. P. Ribeiro, and D. A. S. andH. Waldman, "Dynamic provisioning of shared-backup path protected connections with guaranteed availability requirements," in *Broadband Networks*, 2005, pp. 1320–1327.

[8] L. Zhou, M. Held, and U. Sennhauser, "Connection availability analysis of shared backup path-protected mesh networks," *Journal of Lightwave Technology*, vol. 25, no. 5, pp. 1111–1119, 2007.

[9] "Regions and availability zones," http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[10] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu, "Performance and availability aware regeneration for cloud based multitier applications," in *Proc. of IEEE Dependable Systems and Networks*, 2010, pp. 497–506.

[11] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement," in *Proc. of IEEE Service Computing*, 2011, pp. 72–79.

[12] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *Proc. of IEEE International Conference on Distributed Computing Systems*, 2011.

[13] H. Yanagisawa, T. Osogami, and R. Raymond, "Dependable virtual machine allocation," in *Proc. of IEEE INFOCOM*, 2013, pp. 629–637.

[14] 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, 2010.

[15] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High availability via asynchronous virtual machine replication," in *Proc. of USENIX Networked Systems Design and Implementation*, 2008, pp. 161–174.

[16] N. Limrungsi, J. Zhao, Y. Xiang, T. Lan, H. H. Huang, and S. Subramaniam, "Providing reliability as an elastic service in cloud computing," in *IEEE Communications*, 2012, pp. 2912–2917.

[17] M. S. Bouguerra, A. Gainaru, L. B. Gomez, F. Cappello, S. Matsuoka, and N. Maruyam, "Improving the computing efficiency of hpc systems using a combination of proactive and preventive checkpointing," in *IEEE Parallel and Distributed Processing*, 2013, pp. 501–512.

[18] B. Nicolae and F. Cappello, "Blobcr: Efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots," in *Proc. of ACM High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 34:1–34:12.

[19] S. Di, Y. Robert, F. Vivien, D. Kondo, C.-L. Wang, and F. Cappello, "Optimization of cloud task processing with checkpoint-restart mechanism," in *Proc. of ACM High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 64:1–64:12.

[20] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. Cheng, M. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Transactions on Service Computing*, 2016.

[21] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliability and energy efficiency in cloud computing systems," *J. Netw. Comput. Appl.*, vol. 74, pp. 66–85, 2016.