Venice: Reliable Virtual Data Center Embedding in Clouds

Qi Zhang, Mohamed Faten Zhani, Maissa Jabri, Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo, Canada.
{q8zhang, mfzhani, mjabri, rboutaba}@uwaterloo.ca

***Abstract*—Cloud computing has become a cost-effective model for deploying online services in recent years. To improve the Quality-of-Service (QoS) of the provisioned services, recently a number of proposals have advocated to provision both guaranteed server and network resources in the form of Virtual Data Centers (VDCs). However, existing VDC scheduling algorithms have not fully considered the reliability aspect of the allocations in terms of (1) hardware failure characteristics on which the service is hosted, and (2) the impact of individual failures on service availability, given the dependencies among the virtual components. To address this limitation, in this paper we present a technique for computing VDC availability that considers heterogeneous hardware failure rates and dependencies among virtual components. We then propose Venice, an availability-aware VDC embedding framework for achieving high VDC availability and low operational costs. Experiments show Venice can significantly improve VDC availability while achieving higher income compared to availability-oblivious solutions.**

I. Introduction

Cloud computing has become an attractive model for deploying online service applications in recent years. In a typical cloud computing environment, the Cloud Provider (CP) who owns the physical infrastructure (i.e., data centers) offers resources to one or more Service Providers (SPs). In turn, each SP uses the offered resources to deliver services to end users over the Internet. Traditionally, CPs offer resources in terms of Virtual Machines (VMs) without considering the bandwidth requirements between VMs. In practice, this model has generated numerous concerns related to the network performance, security and manageability. Motivated by this observation, recently a large number of research proposals advocate to offer resources in the form of Virtual Data Centers (VDCs). Also known as a virtual infrastructure, a VDC consists of VMs connected through virtual switches, routers and links with guaranteed bandwidth. This allows SPs to achieve better performance isolation and Quality of Service (QoS) for their applications, while allowing CPs to make informed traffic engineering decisions.

One of the key challenges associated with VDC management in Cloud data centers is the VDC embedding problem, which aims at finding a mapping of VMs and virtual links to physical components (e.g., servers, switches and links) to achieve the following objectives: (1) maximizing the total revenue generated from the embedded VDC requests, (2) minimizing request scheduling (i.e., queuing) delay, which refers to the time a request spends in the waiting queue before it is scheduled, and (3) minimizing the total energy consumed by the data center. As this problem is $\mathcal{NP}$-hard, various heuristics have been proposed in the literature to solve this problem [4], [11], [18], [5]. However, one aspect of the problem that has not been carefully addressed is the reliability of the resulting VDC embeddings. In particular, many Internet services have high availability requirements, because a service outage can potentially incur high penalty in terms of revenue and customer satisfaction. For example, it has been reported that in 2010 business in North America has lost $26.5 billion in revenue due to service downtime. Furthermore, when business critical systems are interrupted, the estimated ability to generate revenue is reduced by 29% [1]. As a result, improving service availability has become a critical concern of today's CPs [7], [15], [16].

Despite its importance, however, achieving availability-aware VDC embedding is a nontrivial problem for several reasons. First, a single service often consists of multiple virtual components (e.g., VMs and virtual links) that may have complex dependencies. For example in a 3-tier web application that consists of a web server, an application server and a database server, if the application server fails, the entire service becomes unavailable regardless of the availability of the web and database servers. Thus, it is necessary to capture the dependencies among virtual components in the VDC availability model. Second, recent analysis on data center hardware reliability [14], [9], [10], [13] has shown that physical data center components have non-uniform failure characteristics in terms of failure rates, impact and repair costs. Thus, given a particular VDC embedding, it is a nontrivial problem to evaluate the quality of the embedding in terms of service availability. Consequently, it is difficult to design an embedding algorithm that finds the optimal trade-off between VDC availability, total revenue and operational cost.

To address these challenges, in this paper we propose Venice, a framework for A**V**ailability-aware **E**mbeddi**N**g **I**n **C**loud **E**nvironments. Specifically, we present a technique for evaluating the availability of VDC embeddings. Using this technique, we study the availability-aware VDC embedding problem in a Cloud computing environment where each SP specifies the overall service availability requirement in addition to resource requirements. We then present a VDC embedding algorithm which aims at maximizing the total revenue of

978-1-4799-3360-0/14/$31.00

the CP while minimizing the total penalty incurred due to hardware failures and service unavailability. Experiments show Venice significantly improves VDC availability and achieves higher income compared to availability-oblivious solutions.

The rest of the paper is organized as follows: Section II surveys related work on data center failure characterization and reliable VDC embedding. In Section III, we present our technique for computing VDC availability. Section IV describes the architecture of Venice and its components. Section V provides the mathematical formulation of the availability-aware VDC embedding problem. The proposed availability-aware embedding algorithm is described in Section VI. We provide simulation results in Section VII. Finally, we draw our conclusions in Section VIII.

II. Related Work

A. Understanding Failure Characteristics in Data Centers

Several recent studies have reported failure characteristics in cloud data centers. The main finding is that these data centers often comprise heterogenous equipments (e.g., physical machines, switches) [17] with skewed distributions of failure rates, impact and repair time [14], [9], [10], [13]. In this section we provide a summary of these heterogenous characteristics.

Failure rates are heterogenous across physical components. Vishwanath et al. [13] have analyzed failure characteristics of $100,000$ servers across multiple Microsoft data centers over a duration of 14 months. They discovered that server unavailability is often caused by hard disk, memory and raid controller failures, with hard disks being the most dominant source of server failures (i. e., accounts for 78% of total failures). They have also reported that the number of server failures is often correlated with the number of hard disks that the server contains. Furthermore, a server that has experienced a failure is likely to experience another failure in the near future. This results in a skewed distribution of server failure rate. On the other hand, for network equipment, Gill et al.[9] reported that the failure rates of different equipment can vary significantly depending on their type (servers, Top-of-Rack (ToR) switches, aggregation switches, routers) and model. In particular, Load Balancers (LBs) have high probability of failure (over 20%), whereas the failure probability of switches is often very low (less than 5%). Furthermore, the failure rates are unevenly distributed. For example, the number of failures across LBs are highly variable with a few outlier LBs experiencing more than $40\times$ more failures over the one-year period.

Failures have heterogenous impact and repair times. While server failures can take up to hours to fix, certain network failures can be fixed within seconds [9]. In general, most of the network failures can be mitigated promptly using simple actions [14]. However, certain failures can still cause significant network downtime. For example, although more than 95% of network failures can be fixed within 10 minutes, the worst 0.09% of failures can take more than 10 days to resolve [10]. Even though LB failures only cause packet loss over short periods of time, failure of ToR switches can cause significant downtime of all the servers in the rack. Interestingly, it has also been reported that correlated equipment and link failures are generally rare. For example, Gill et al. analyzed the correlations among link failures and found that more than 50% of link failures are single link failures, and more than 90% of link failures involve less than 5 links [9]. Similarly, Greenberg et al. [10] reported that most of the device failures are small (i.e., involve less than 4 devices).

In summary, these analyses show that (1) there is a significant heterogeneity in data centers in terms of failure rates and repair times; (2) It is unlikely to see large correlated failures. We believe these observations not only suggest that VDC embedding schemes should consider the heterogenous hardware reliability characteristics when placing mission-critical service applications, but also provide insights on how to estimate VDC availability in Cloud data centers.

B. Reliable Virtual Infrastructure Embedding

Due to the importance of providing high service availability in Cloud environments, recently there is a trend towards designing reliable embedding schemes for Cloud data centers. For instance, Xu et al. [15] proposed a resource allocation scheme for provisioning VDCs with backup VMs and links. However, their solution does not consider the availability of physical machines and links. Yeow et al. [16] provided a technique for estimating the number of backup VMs required to achieve the desired reliability objectives. However, they do not consider the availability of virtual links and assume that machines have an identical failure rate. Bodik et al. [7] proposed an allocation scheme for improving service[1] survivability while mitigating the bandwidth bottleneck in the core of the data center network. Their scheme improves the fault tolerance by spreading out VMs across multiple fault-domains while minimizing the total bandwidth consumption. However, this approach does not consider the heterogenous failure rates of the underlying physical equipment.

III. Computing VDC Availability

In this section, we study the problem of computing VDC availability in the presence of heterogenous hardware failure characteristics and VM dependencies. In our model, a VDC consists of multiple VMs connected by virtual links. Certain VMs may form a *replication group*, in which each VM can operate as a backup if another VM in the same group fails. In this case, the replication group is available as long as one of the VMs in the group is available [2]. VM replication is commonly used in cloud applications not only for reliability, but also for load balancing purposes [7]. In our model, each VDC request captures (1) the topology and resource requirements of virtual components (e.g., VMs, virtual switches) and virtual links, (2) the sets of virtual components that form replication groups, and (3) the overall VDC availability objective.

[1] A service defined in [7] is a set of VMs that execute the same code.

[2] Our solution can be generalized to handle the cases where replication group is available as long as m of the total n VMs are available.

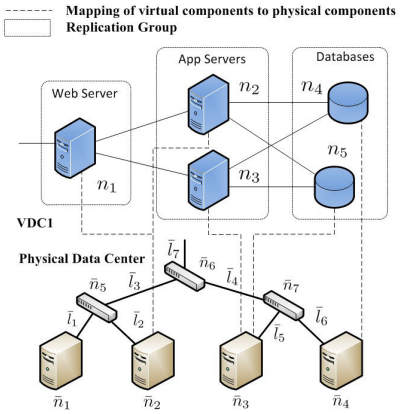


Fig. 1: Embedding of a 3-tier Application (VDC1)

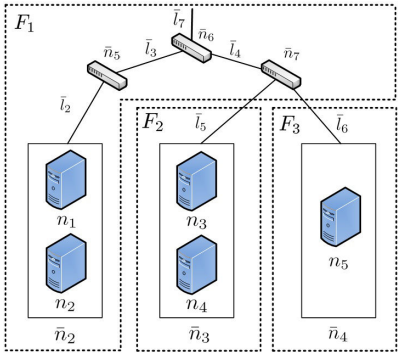


Fig. 2: Analyzing the Availability of VDC1

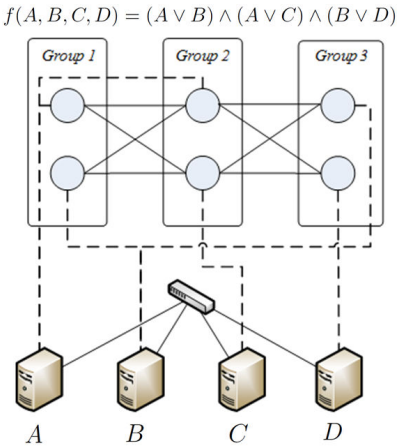


Fig. 3: Example Illustrating Theorem 1

To illustrate our approach, consider an example 3-tier web application modeled as VDC1 shown in Figure 1. It consists of a single web server n_1, two application servers n_2 and n_3 and two database servers n_4 and n_5. The two application servers n_2 and n_3 can provide backup for each other, and thus form a replication group. Similarly, n_4 and n_5 also form a replication group. In our example, the 3-tier web application provides only one type of service that requires coordination among all 3 tiers. Lastly, each virtual link is embedded along the shortest path between corresponding VMs.

In our model, we define $A_{\bar{n}i}$ and $A_{\bar{l}i}$ as the availability of physical machine hosting server $\bar{n}_i$ and physical link $\bar{l}_i$ respectively, for $i \in \{1, ..., 7\}$. In general, the availability of a physical component j is computed as:

$$A_j = \frac{MTBF_j}{MTBF_j + MTTR_j} \quad (1)$$

where $MTBF_j$ and $MTTR_j$ correspond to the *Mean Time Between Failures* and the *Mean Time To Repair* of component j respectively [6]. Both $MTBF_j$ and $MTTR_j$ can be obtained from historical failure and maintenance records of component j. Our goal is to determine the availability of VDC1 based on the availability of the physical components. In our example, the service is available if there exists a path from the web server to the database server where every component (physical nodes and links) along the path is available. However, the replication of application servers and database servers makes a direct evaluation of service availability a difficult task. To address this issue, we break down the possible failures into a set of *failure scenarios* S. A failure scenario is a specific failure configuration in which a few physical components have failed. We can compute the availability A^s_{VDC} of the VDC in each failure scenario $s \in S$, and then combine them to obtain the VDC availability using conditional probability. Even though there is a large number of failure scenarios to consider in the general case, in our example we can categorize the failure scenarios into a small number of cases, each describing a set of scenarios. Specifically, define $F_1 = \{\bar{n}_2, \bar{n}_5, \bar{n}_6, \bar{n}_7, \bar{l}_2, \bar{l}_3, \bar{l}_4, \bar{l}_7\}$, $F_2 = \{\bar{n}_3, \bar{l}_5\}$ and $F_3 = \{\bar{n}_4, \bar{l}_6\}$, as shown in Figure 2, the failure scenarios that affect VDC1 can be divided into 3 cases:

Case 1 (c_1): At least one component in F_1 is unavailable. This case occurs with probability $P(c_1) = 1 - \prod_{i \in F_1} A_i$. In this case, the service is unavailable as the web server is not reachable, thus $A^{c_1}_{VDC} = 0$.

Case 2 (c_2): All the components of F_1 are available, but at least one component in F_2 is unavailable. This occurs with probability $P(c_2) = \prod_{i \in F_1} A_i (1 - \prod_{i \in F_3} A_i)$. The service availability in this case is determined by the availability of components in F_3 (i.e., $\bar{l}_6$ and $\bar{n}_4$). Thus, $A^{c_2}_{VDC} = \prod_{i \in F_3} A_i$.

Case 3 (c_3): All the components of F_1 and F_2 are available. This occurs with probability $P(c_3) = \prod_{i \in F_1 \cup F_2} A_i$. In this case the service is available, thus $A^{c_3}_{VDC} = 1$.

The availability of VDC1 can now be computed as:

$$A_{VDC1} = \sum_{i=1}^{3} P(c_i) A^{c_i}_{VDC}. \quad (2)$$

Even though this approach of computing VDC availability by breaking down failures into failure scenarios is intuitive, it cannot be directly applied in practice. The reason is that given n physical components on which the VDC is embedded, and each component can either be available or unavailable, there are $O(2^n)$ possible scenarios to be considered in the worst case. In fact, the following result show that computing VDC availability optimally is not a viable option.

Theorem 1. *There is no polynomial time algorithm for computing VDC availability unless* $P = NP$.

Proof: We show that the problem of computing VDC availability can be reduced from the counting monotone 2-satisfiability problem (#MONOTONE-2SAT) [8]. Specifically, a boolean expression $f(\cdot)$ is in 2-Conjunctive Normal Form (CNF) if $f(\cdot)$ is a conjunction of multiple clauses, and each clause is a disjunction of at most two input variables. Given a 2-CNF boolean expression $f(\cdot)$ that does not contain any negated variables, the #MONOTONE-2SAT problem asks how many input sets for which $f(\cdot)$ evaluates to true.

Algorithm 1 Computing VDC Availability

$P \leftarrow \bar{N} \cup \bar{L}$ the set of physical components on which VDC is embedded
$A_{VDC} = 0$
for $i = 0$ **to** L **do**
  for all $S \in \{T \subseteq P : |T| = L\}$ **do**
    if Service is available when all nodes in S fail **then**
      $A_{VDC} \leftarrow A_{VDC} + \prod_{i \in S}(1 - A_i) \prod_{i \notin S} A_i$
$A_{sample} = 0$
for $i = 1$ **to** $N_{samples}$ **do**
  Randomly draw k virtual components S
  if Service is available when all components in S fail **then**
    $A_{sample} \leftarrow A_{sample} + \prod_{i \in S}(1 - A_i) \prod_{i \notin S} A_i$
$A_{VDC} \leftarrow A_{VDC} + A_{sample}/N_{samples}$
return A_{VDC}

The reduction works as follows: given a 2-CNF boolean expression $f(\cdot)$ that contains N input variables, we construct a replication group for each clause that contains two virtual nodes. The replication groups are connected in series, such that every virtual node is connected to both virtual nodes in the subsequent replication group. The physical topology of data center is a star topology where each physical machine corresponds to an input variable in $f(\cdot)$, and all the servers are connected to a central switch. Furthermore, all physical machines and links have infinite capacity. Virtual nodes are then embedded in the physical nodes by mapping each variable in each of the clauses to the corresponding machine that represents this variable. Figure 3 provides an example to illustrate this procedure. The input expression $f(A, B, C, D) = (A \vee B) \wedge (A \vee C) \wedge (B \vee D)$ can be represented as a VDC with 3 replication groups connected in series that are embedded in 4 physical machines. Finally, we assume each physical machine has availability 0.5, the links and switches have availability 1. Since each physical machine representing a variable has equal probability to be available and unavailable, every input argument set is equally likely to occur in the setup. Therefore the availability of the VDC multiplied by 2^n will give exactly the number of input sets that satisfies $f(\cdot)$, thus solving #MONOTONE-2SAT optimally. Since #MONOTONE-2SAT belongs to the complexity class of $\#P$-complete [8] for which no polynomial time algorithm exists unless $P = NP$, the result follows. ■

Theorem 1 indicates that computing VDC availability is a difficult problem even for simple star topologies. Thus, it is necessary to develop fast heuristics for computing VDC availability. One naïve solution is to leverage the fact that the probability of observing k physical components fail simultaneously is low. For example, assume all physical components have availability $\geq 95\%$, the probability of seeing 3 physical components fail simultaneously is at most $(1 - 95\%)^3 \leq 0.015\%$. This implies that considering failure scenarios that involve at most 2 failed physical components simultaneously can already provide an accurate lower bound of the actual VDC availability. However, although this approach works well for small VDCs, it fails to produce accurate estimation for large VDCs. This is because the remaining $\sum_{k=3}^{n} \binom{n}{k}$ cases can still contribute to a large fraction of scenarios for a large value of n, rendering this approach ineffective in this case.

To address this limitation, we resolve to use sampling techniques. The idea is to improve the estimation by sampling over the remaining 2^n possible failure scenarios. Let $s \in \{0,1\}^n$ denote a sample failure scenario over n physical components, and let $P(s)$ denote the probability that s occurs. Define S as the failure scenarios that involve less than k simultaneous component failures, and let $\bar{S}^k = \{0,1\}^n \backslash S^k$. Suppose the samples drawn by the sampling algorithm is $N \in \bar{S}^k$, a lower bound on VDC availability can be computed as

$$A_{VDC}^{lower} = \sum_{s \in S^k} P(s) A_{VDC}^s + \sum_{s \in N} A_{VDC}^s \cdot P(s) \qquad (3)$$

This estimate is better than the naïve solution before. However, it still requires a large number of samples to be accurate. In this case, we use a statistical technique called *importance sampling* [3]. Let $\bar{P}(s)$ denote the probability that s is drawn in $\{0,1\}^n \backslash S^k$, and define $w(s) = \frac{P(s)}{\bar{P}(s)}$. It is easy to see that

$$\begin{aligned} A_{VDC} &= \sum_{s \in S^k} P(s) A_{VDC}^s + \sum_{s \in \bar{\mathbf{S}}^k} P(s) A_{VDC}^s \\ &= \sum_{s \in S^k} P(s) A_{VDC}^s + \sum_{s \in \bar{\mathbf{S}}^k} \bar{P}(s) A_{VDC}^s \cdot \frac{P(s)}{\bar{P}(s)} \\ &\approx \sum_{s \in S^k} P(s) A_{VDC}^s + \frac{1}{|N|} \sum_{s \in N} A_{VDC}^s w(s) \qquad (4) \end{aligned}$$

Thus, if we draw samples randomly from $\{0,1\}^n \backslash S^k$ according to probability density function $\bar{P}(s)$, we can estimate the A_{VDC} as the sample mean of the VDC availability value in each scenario weighted by $w(s)$. The purpose of computing the availability for S^k separately is to ensure these important samples are considered. For simplicity, we chose $\bar{P}(s)$ to be uniform over $\{0,1\}^n \backslash S^k$ in our implementation. It is easy to see that both estimations approach the true VDC availability as we increase the sample set $\bar{S}^k$ towards $\{0,1\}^n \backslash S^k$.

Lastly, even though so far our discussion has been focusing on the cases where a VDC is either available or unavailable in given failure scenario (e.g., $A_{VDC}^s \in \{0,1\}$), it is straightforward to generalize it to the cases where *partial availability* is considered (e.g., $A_{VDC}^s \in [0,1]$). Partial availability is useful when a failure does not shut down the service, but rather reduces the overall service quality. It is clear that equation (3) and (4) can be generalized to handle this case as well.

IV. SYSTEM ARCHITECTURE

Leveraging the technique for computing VDC availability in the previous section, we describe Venice, a framework for providing availability-aware VDC embedding, as shown in Figure 4. Specifically, the *Monitoring Module* is responsible for monitoring and detecting failures in the physical infrastructure. The *Reliability Analysis Module* is responsible for characterizing the availability of the data center components based on the statistics provided by the monitoring module. Finally, the *VDC Scheduler* is responsible for embedding each

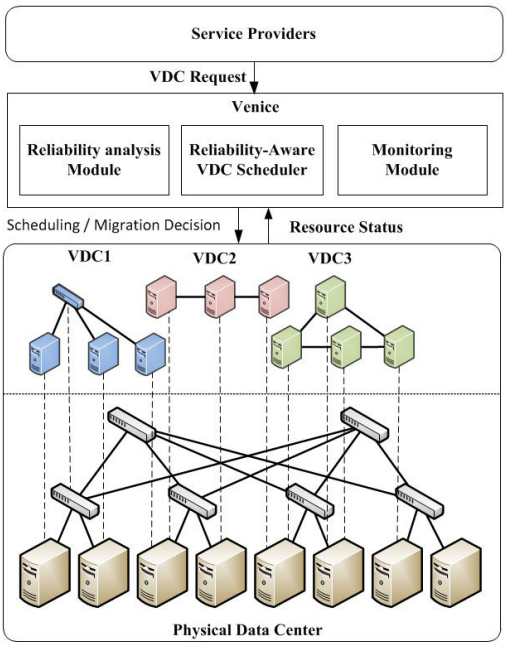


Fig. 4: Venice Architecture

VDC in the data center. If there is no feasible embedding for a VDC, the request is kept in a scheduling queue until the SP decides to withdraw it. The VDC scheduler also uses VM migration to improve the availability of high priority VDCs.

V. Availability-aware VDC-Embedding

This section formally introduces our model for availability-aware VDC-embedding. We model a data center as a graph $\bar{G} = (\bar{N}, \bar{L})$. Let R denote the types of resources offered by each node (e.g., CPU and memory for servers). We assume each node $\bar{n} \in \bar{N}$ has a capacity $c^r_{\bar{n}}$ for each resource $r \in R$, and each link $\bar{l} \in \bar{L}$ has a bandwidth capacity $b_{\bar{l}}$. Furthermore, we define $\bar{s}_{\bar{n}\bar{l}}, \bar{d}_{\bar{n}\bar{l}} \in \{0,1\}$ as boolean variables that indicate whether $\bar{n}$ is the source and destination node of link $\bar{l}$, respectively. Similarly, we assume there is a set of VDC requests I, each request $i \in I$ asks for embedding a VDC $G^i = (N^i, L^i)$. We also assume each node $n \in N^i$ has a capacity c^{ir}_n for resource $r \in R$, and each link $l \in L^i$ has a bandwidth capacity b_l. We also define s_{nl} and d_{nl} as boolean variables that indicate whether n is the source and destination node of $l \in L^i$, respectively. Let $x^i_{n\bar{n}} \in \{0,1\}$ be a variable that indicates whether virtual node n of VDC i is embedded in substrate node $\bar{n}$, and $f^i_{l\bar{l}}$ be a variable that measures the bandwidth of edge $\bar{l}$ allocated for virtual link $l \in L^i$. To ensure the embedding does not violate the capacity constraints of the physical resources, the following constraints must be met:

$$\sum_{i \in I} \sum_{n \in N^i} x^i_{n\bar{n}} c^{ir}_n \leq c^r_{\bar{n}} \qquad \forall \bar{n} \in \bar{N}, r \in R \tag{5}$$

$$\sum_{i \in I} \sum_{l \in L^i} f^i_{l\bar{l}} \leq b_{\bar{l}} \qquad \forall \bar{l} \in \bar{L} \tag{6}$$

Furthermore, each link embedding must satisfy the flow constraint that the total outgoing flow of a physical node $\bar{n}$ for a virtual link L^i is zero unless $\bar{n}$ hosts either the source or the destination of virtual link i:

$$\sum_{\bar{l} \in \bar{L}} \bar{s}_{\bar{n}\bar{l}} f^i_{l\bar{l}} - \sum_{\bar{l} \in \bar{L}} \bar{d}_{\bar{n}\bar{l}} f^i_{l\bar{l}} = \sum_{n \in N^i} x^i_{n\bar{n}} s^i_{nl} b_l - \sum_{n \in N^i} x^i_{n\bar{n}} d^i_{nl} b_l \quad \forall i \in I, l \in L^i, \bar{n} \in \bar{N}. \tag{7}$$

Next, we need to consider node placement constraints. This constraint is used to specify that VMs are exclusively embedded in physical machines (i.e., not in switches). Define $\tilde{x}^i_{n\bar{n}} \in \{0,1\}$ as a boolean variable that indicates whether virtual node n can be embedded in physical node $\bar{n}$, the placement constraint can be captured by the following equation:

$$x^i_{n\bar{n}} \leq \tilde{x}^i_{n\bar{n}} \qquad \forall i \in I, n \in n, \bar{n} \in \bar{N} \tag{8}$$

We also need to ensure every $n \in N^i$ is embedded:

$$\sum_{\bar{n} \in \bar{N}} x^i_{n\bar{n}} = 1 \qquad \forall i \in I, n \in N^i \tag{9}$$

Lastly, we need to define $y_{\bar{n}}$ as a boolean variable that indicates whether physical node $\bar{n}$ is active. A physical node is active if it hosts at least one virtual component. This implies the following constraints must hold:

$$y_{\bar{n}} \geq x^i_{n\bar{n}} \qquad \forall i \in I, n \in N^i, \bar{n} \in \bar{N} \tag{10}$$

$$y_{\bar{n}} \geq \frac{1}{b_l} f^i_{l\bar{l}} \bar{s}_{\bar{n}\bar{l}} \qquad \forall i \in I, \bar{n} \in \bar{N}, l \in L^i, \bar{l} \in \bar{L} \tag{11}$$

$$y_{\bar{n}} \geq \frac{1}{b_l} f^i_{l\bar{l}} \bar{d}_{\bar{n}\bar{l}} \qquad \forall i \in I, \bar{n} \in \bar{N}, l \in L^i, \bar{l} \in \bar{L} \tag{12}$$

A. Migration

VM migration can be used to improve the overall quality of the embedding in terms of total revenue. However, in order to leverage VM migration for VDC embedding, it is necessary to consider the migration cost in terms of service disruption and bandwidth cost. Specifically, we treat migration cost as a one-time embedding cost. The one-time cost of embedding a node n of VDC i (which is currently embedded in node $\bar{m} \in \bar{N}$) in node $\bar{n} \in N$ is given by:

$$g^i_{n\bar{n}} = \begin{cases} mig(n, \bar{m}, \bar{n}) & \text{if } \bar{n} \neq \bar{m} \\ 0 & \text{if } \bar{n} = \bar{m} \text{ or } n \text{ is not embedded} \end{cases}$$

where $mig(n, \bar{m}, \bar{n})$ denotes the cost of migrating node n from node $\bar{m}$ to node $\bar{n}$. Thus, when n is already embedded but needs to be migrated from $\bar{m}$ to $\bar{n}$, the one-time embedding cost is equal to the migration cost. This cost is equal to zero when n is already embedded in the physical node $\bar{n}$ (i.e., $\bar{n} = \bar{m}$), or when the node n is embedded for the first time.

B. Reliability Requirement

Let A_i denote the availability of VDC i, we define the SLA penalty due to resource unavailability as

$$C^{unavail} = \sum_{i \in I} (1 - A_i) \pi_i \tag{13}$$

where π_i is the unit SLA penalty due to resource unavailability. There is also the virtual resource restoration cost which includes the cost of restarting VMs and reconfiguring the

network devices. We can define the restoration cost for a failure of node $\bar{n}$ as

$$C_{\bar{n}}^{restore} = \rho_{\bar{n}} + \sum_{i \in I} \sum_{n \in N^i} x_{n\bar{n}}^i \lambda_n + \sum_{l \in L^i} f_{l\bar{l}}^i u_{\bar{n}\bar{l}} \lambda_l \quad (14)$$

$$C_{\bar{l}}^{restore} = \rho_{\bar{l}} + \sum_{i \in I} \sum_{l \in L^i} f_{l\bar{l}}^i \lambda_l \quad (15)$$

where λ_n and λ_l are the costs for restoring virtual node n and virtual link l, respectively. $F_{\bar{n}}$ and $F_{\bar{l}}$ are the node and link failure rates for node $\bar{n} \in \bar{N}$ and $\bar{l} \in \bar{L}$, respectively. We also define $u_{\bar{n}\bar{l}} = \max\{\bar{s}_{\bar{n}\bar{l}}, \bar{d}_{\bar{n}\bar{l}}\}$ as a boolean variable that indicates whether physical link $\bar{l}$ uses node $\bar{n}$, the total service unavailability cost is given by

$$\mathcal{C}_A = C^{unavail} + \sum_{\bar{n} \in \bar{N}} F_{\bar{n}} C_{\bar{n}}^{restore} + \sum_{\bar{l} \in \bar{L}} F_{\bar{l}} C_{\bar{l}}^{restore} \quad (16)$$

C. Optimization Problem Formulation

Let $p_{\bar{n}}$ represent the energy cost of an active node $\bar{n}$ (expressed in dollars). The goal of the reliability-aware embedding can be stated as finding an embedding that achieves

$$\min \sum_{\bar{n} \in \bar{N}} y_{\bar{n}} p_{\bar{n}} + \sum_{i \in I} \sum_{n \in N^i} \sum_{\bar{n} \in \bar{N}} \gamma_n x_{n\bar{n}}^i g_{n\bar{n}}^i + C_A \quad (17)$$

Subject to constraints (4)-(11). The first, second and third term represent the energy, migration, and unavailability costs, respectively. Here γ_n is a weight factor that controls the tradeoff between migration cost and other costs. This problem is clearly $\mathcal{NP}$-hard as it generalizes the bin-packing problem.

VI. VDC Embedding Algorithm

A. Reliability-Aware VDC Embedding Heuristic

This section describes the VDC embedding algorithm we proposed in Venice. There are two major issues we have to address in order to achieve availability-aware embedding. First, we want to differentiate incoming VDC requests so that the machines with high availability are allocated to those VDCs with high availability requirements. Second, high availability should not achieved at the expense of high resource usage. In particular, even though it is possible to improve VDC availability by spreading replicas across a large number of physical nodes, doing so can go against the goal of minimizing the number of active physical nodes (e.g., for minimizing bandwidth usage and energy consumption) [7]. Thus, we need to find a trade-off between these objectives.

To address the first challenge, our algorithm embeds a given VDC on machines with the least availability that can still attain the desired VDC availability requirement. As most of the machines (and links) have similar availability, they can be divided into distinct availability types (e.g. based on their actual type). Let $\mathcal{N}$ denote the number of availability types. The embedding algorithm proceeds in multiple trials. In the first trial, we use all the machines to embed the VDC. In each subsequent trial, we remove machines of the lowest availability type and use remaining machines to embed the VDC. This produces $\mathcal{N}$ different embedding solutions, and the one with the best cost is the one used for actual embedding.

To address the second challenge, we leverage the fact that availability is additive, as demonstrated in Section III. we first start with an initial embedding where only one VM in each replication group is embedded. We then compare the resulting availability with the desired VDC availability. If it is lower than the desired value, we select the next virtual node such that the embedding of this node can significantly improve VDC availability. This process repeats until the desired VDC availability is achieved, and subsequently the remaining virtual components can be embedded greedily without considering the VDC availability requirement.

We now describe our reliability-aware VDC embedding algorithm (depicted by Algorithm 2) in details. Upon receiving a VDC request i, the algorithm first separates the physical nodes into 2 lists based on whether they are active or inactive. The algorithm then runs $\mathcal{N}$ embedding trials, each considers one less availability type than the previous trial. In each trial, We sort virtual nodes in decreasing order of their size. Specifically, for each $n \in N^i$, we define its size as $size_n^i = \sum_{r \in R} w^r c_n^{ir}$, where w^r is a weight factor for resource type r. The intuition is that $size_n^i$ measures the difficulty of embedding n. Thus w^r is selected based on the scarcity of resource type $r \in R$.

After sorting all virtual nodes in N^i according to $size_n^i$, our algorithm then tries to embed each node in the sorted order, based on whether it is connected to any embedded nodes. For each selected node $n \in N^i$, define $L_n^i \subseteq L^i$ as the set of virtual links that have already been embedded and that are connected to n. Define $\sigma^l(l) \subseteq \bar{L}$, and $\sigma^n(l) \subseteq \bar{N}$ as the set of links and nodes in which the link l is embedded, respectively. The cost for embedding a node n on $\bar{n}$ becomes:

$$cost^i(n, \bar{n}) = \gamma_n(mig(n, \bar{m}, \bar{n}) + MigOther(n, \bar{n})) + F_{\bar{n}} \lambda_n + \sum_{l \in L_n^i} \left(\sum_{n' \in \sigma^n(l)} F_{n'} \lambda_n + \sum_{\bar{l} \in \sigma(l)} b_l + F_{\bar{l}} \lambda_l \right) \quad (18)$$

where the last two terms capture the restoration and bandwidth cost of embedding n on $\bar{n}$. Note that as some virtual components may not have been embedded yet, the bandwidth and link restoration costs in equation (18) only include the links that have already been embedded. Finally, $MigOther(n, \bar{n})$ is the cost of migrating away the nodes on $\bar{n}$ in order to accommodate n on $\bar{n}$. Formally, we denote by $loc(\bar{n})$ the set of virtual nodes hosted on physical node $\bar{n}$. Let $mig(\tilde{n}, \bar{n})$ denote the minimum cost (including both the migration cost and service unavailability cost defined in equation (16)) for migrating away $\tilde{n} \in loc(\bar{n})$ to another node that has capacity to host $\tilde{n}$. As computing $mig(\tilde{n}, \bar{n})$ generalizes a minimum knapsack problem [12], we use a greedy algorithm to compute $MigOther(n, \bar{n})$. In particular, for a virtual node $\tilde{n} \in loc(\bar{n})$ that belongs to a VDC j, we compute a cost-to-size ratio $r_{\tilde{n}}$:

$$r_{\tilde{n}} = \arg \min_{\bar{n}' \in \bar{N}'} \left(\frac{mig(\tilde{n}, \bar{n}, \bar{n}')}{\sum_{r \in R} w^r c_{\tilde{n}}^{jr}} \right) \quad (19)$$

where $\bar{N}'$ is the set of nodes to examine for VM migration. Currently, we set $\bar{N}'$ to be the machines within the same rack

as $\bar{n}$. Then, we sort $loc(\bar{n})$ based on the values of $r_{\tilde{n}}$, and greedily migrate away $r_{\tilde{n}}$ in the sorted order until there is sufficient capacity to accommodate n on $\bar{n}$. The total migration cost of this solution produces $MigOther(n, \bar{n})$. If there is no feasible solution, we set $MigOther(n, \bar{n}) = \infty$. Lastly, once the embedding cost $cost^i(n, \bar{n})$ is computed for every $\bar{n} \in \bar{N}$, we embed n on the node with the minimum $cost^i(n, \bar{n})$.

This process repeats until we have embedded at least one component in each replication group. The algorithm then compares the availability of the current embedding with the desired VDC availability. If it is lower than the desired value, we find the next virtual node n such that the embedding of n on a new physical node $\bar{n}$ achieves the highest reduction in solution cost. Specifically, let A_i and $A_i(n, n')$ denote the current availability of VDC i and the availability of VDC i after embedding n on n', respectively. The cost for embedding node n can be computed as:

$$\bar{n} = \arg\min_{n' \in N} \left\{ cost^i(n, n') + (1 - A_i)\pi_i - (1 - A_i(n, n'))\pi_i \right\}$$

This process is repeated until the desired VDC availability is achieved, and subsequently the remaining virtual components can be embedded greedily using $cost^i(n, \bar{n})$ defined in equation (18). Finally, the algorithm terminates when either $cost^i(n^*, \bar{n}) = \infty$ (which indicates VDC i is not embeddable), or the embedding of VDC i actually hurts the net income (i.e., the cost is higher than the revenue for VDC i), in which case the request for VDC i should be rejected.

As for the running time of the algorithm, assume each physical node can host at most n_{max} virtual nodes, and the number of physical machine per rack is at most N_{rack}, the running time for computing $MigOther(n, \bar{n})$ (Line 11) is $O(|\bar{N}|n_{max}|N_{rack})$. Line 6 to 18 take $O(|N^i||N^i|)$ rounds of computing $MigOther(n, \bar{n})$, as we need to search for an embedding for each virtual node in N^i. Thus, the total running time of the algorithm is $O(|A_{TH}||N^i||\bar{N}|n_{max}N_{rack})$.

B. VDC Consolidation Algorithm

Since VDCs may come and leave, the initial embedding of VDCs can become suboptimal over time. Hence, it is possible to use migration to (1) consolidate the VMs in order to minimize bandwidth usage and energy consumption and to (2) improve the availability of embedded VDCs. For example, at night time when the data center is under-utilized, it is possible to consolidate VMs on a few physical machines with high availability to save energy, or reducing the service unavailability cost defined in equation (16) for certain VDCs. In Venice, the VDC consolidation is performed only when the arrival rate is low over a period of time (i.e., below a threshold λ_{th} requests per second over a duration of T minutes).

Our dynamic VDC consolidation algorithm is represented by Algorithm 3. The algorithm starts by improving the availability of VDCs using active machines, and then tries to reduce the number of active machines to minimize energy cost. In the first step, the algorithm identifies the top V VDCs with the highest unavailability cost, where V is a constant that can be controlled. For each identified VDC, the algorithm uses

Algorithm 2 Algorithm for embedding VDC request i

1: $\bar{M} \leftarrow$ active machines, $\bar{U} \leftarrow$ inactive machines, $M_1, ...M_{\mathcal{N}} \leftarrow$ availability groups in increasing order of availability, $BestCost \leftarrow \infty$
2: **for** $i \leftarrow 1$ **to** $\mathcal{N}$ **do**
3: $M_{th} \leftarrow \bigcup_{p=1}^{i} M_p \cap \bar{M}$
4: $N_{th} \leftarrow \bigcup_{p=1}^{i} M_p \cap \bar{U}\}$
5: $\bar{N}_{th} = M_{th} \cup U_{th}$
6: $S \leftarrow N^i$ with one node from each reliability group
7: **repeat**
8: $C \leftarrow$ nodes in S that are connected to embedded nodes.
9: **if** $C = \{\emptyset\}$ **then**
10: $C = \{S\}$
11: **for** each $\bar{n} \in \bar{N}_{th}$ in sorted order **do**
12: Compute embedding cost $cost^i(n^*, \bar{n})$ according to equation (18). If not feasible, set $cost^i(n^*, \bar{n}) = \infty$.
13: **if** $cost^i(n^*, \bar{n}) = \infty \forall \bar{n} \in \bar{N}_{th}$ **then**
14: Continue
15: **else**
16: Embed n^* on $\bar{n}$ with lowest $cost^i(n, \bar{n})$. $S \leftarrow S \backslash n^*$
17: **until** $S == \{\emptyset\}$
18: $S \leftarrow$ remaining nodes in N^i
19: **repeat**
20: Sort C according $size_n^i$ defined by equation (VI-A).
21: $n^* \leftarrow$ first node in C
22: **if** $A_{VDC} \geq$ required VDC **then**
23: $\bar{N}^{'} \leftarrow \bar{N}_{th}$
24: **else**
25: $\bar{N}^{'} \leftarrow \bar{N}_{th} \backslash \{$nodes where n^*'s siblings are embedded $\}$
26: **for** each $\bar{n} \in \bar{N}^{'}$ in sorted order **do**
27: Compute embedding cost $cost^i(n^*, \bar{n})$ according to equation (18). If not feasible, set $cost^i(n^*, \bar{n}) = \infty$.
28: **if** $cost^i(n^*, \bar{n}) = \infty \forall \bar{n} \in \bar{N}^{'}$ **then**
29: Continue
30: **else**
31: Embed n^* on $\bar{n} \in \bar{N}^{'}$ with lowest $cost^i(n, \bar{n})$. $S \leftarrow S \backslash n^*$
32: **if** Solution Cost $< BestCost$ **then**
33: $BestCost \leftarrow$ Solution Cost, $BestSolution \leftarrow$ current solution
34: **until** $S == \{\emptyset\}$
35: **return** $BestSolution$

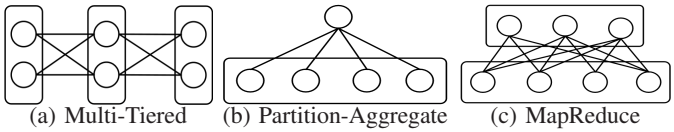
Algorithm 2 to compute a new embedding. The re-embedding is performed only if the new embedding improves the solution quality. This process repeats until all V VDCs have been examined. In the second step, the algorithm tries to reduce the number of active machines. It first sorts the physical nodes in increasing order of their utilizations. For each $\bar{n} \in \bar{N}$, we define the utilization $U_{\bar{n}}$ of $\bar{n}$ as the weighted sum of the utilization of each type of resources (e.g., CPU, memory, disk and network bandwidth):

$$U_{\bar{n}} = \sum_{r \in R} \sum_{i \in I} \sum_{n \in N^i : n \in loc(\bar{n})} \frac{w^r c_n^{ir}}{c_{\bar{n}}^r}, \qquad (20)$$

Once physical nodes are sorted, for each physical node we sort virtual nodes $n \in loc(\bar{n})$ according their size $size_n^i$. Let i denote the VDC that n belongs to. We then run Algorithm 2 on VDC i with physical nodes excluding $\bar{n}$. This will find an embedding where $\bar{n}$ is not used. Once all virtual nodes have been migrated, we compute the cost of the solution

Algorithm 3 Dynamic VDC Consolidation Algorithm

1: Let $\bar{S}$ represent the set of active machines
2: Sort VDCs in increasing order of $\mathcal{C}_A$
3: **for** $i = 1$ **to** V **do**
4: $cost(i) \leftarrow$ Cost of running Algorithm 2 on VDC i.
5: **if** $cost(i) \leq$ current cost **then**
6: Re-embed VDC i according to Algorithm 2
7: **repeat**
8: Sort $\bar{S}$ in increasing order of $U_{\bar{n}}$ according to equation (20).
9: $\bar{n} \leftarrow$ next node in $\bar{S}$, $S \leftarrow loc(\bar{n})$
10: Sort S according to $size_n^i$ defined in equation (VI-A).
11: **for** $n \in S$ **do**
12: $n \leftarrow$ next node in S, $i \leftarrow$ the VDC to which n belongs
13: Run Algorithm 2 on VDC i over $\bar{S} \backslash \{\bar{n}\}$.
14: $cost(\bar{n}) \leftarrow$ the total cost according to equation (17)
15: **if** $cost(\bar{n}) \leq p_{\bar{n}}$ **then**
16: Migrate all virtual nodes according to Algorithm 2
17: Set $\bar{n}$ to inactive
18: $\bar{S} \leftarrow \bar{S} \backslash \{\bar{n}\}$
19: **until** $U_{\bar{n}} \geq C_{th}$



(a) Multi-Tiered (b) Partition-Aggregate (c) MapReduce

Fig. 5: Example VDC Topologies

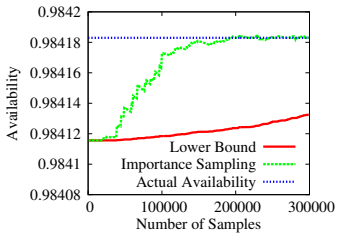


Fig. 6: Computing Availability

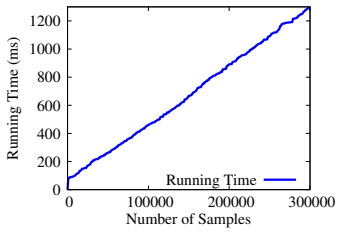


Fig. 7: Running Time

according to equation (17) and compare it to the energy saving, which is represented by $p_{\bar{n}}$. If the total saving is greater than the total cost of the solution, migration is performed and $\bar{n}$ becomes inactive. Otherwise, the algorithm proceeds to the next physical node $\bar{n}$ in the list until the cluster is sufficiently consolidated (i.e., all the machines in the cluster have reached a threshold C_{th}). This ensures the quality of the embedding will increase as the algorithm proceeds.

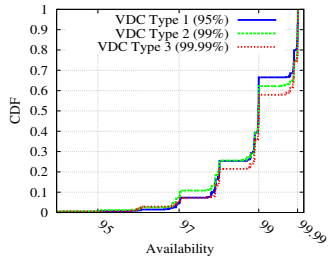
Finally, we analyze the running time of Algorithm 3. Line 2 takes $O(|I|\log(V))$ time by using a binary heap to find top V VDCs. Line 3-8 takes $O(V|A_{TH}||N^i||\bar{N}|n_{max}N_{rack})$ time to complete by running Algorithm 3 up to V times. Thus the total running time of the algorithm is $O(|I|\log(V)+|\bar{N}|^2\log|\bar{N}|+(|\bar{N}|+V)|A_{TH}||N^i||\bar{N}|n_{max}N_{rack})$.

VII. Performance Evaluations

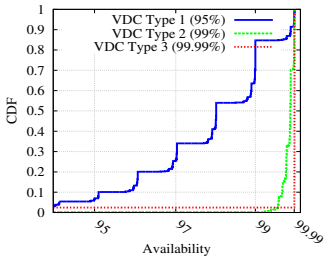
We have implemented Venice and evaluated its performance against VDC Planner [18], which is a VDC embedding framework that leverages VM migration to achieve high revenue. However, VDC Planner does not use availability information for VDC embedding. In our experiments, we have simulated a VL2 topology [10] with 120 physical machines organized in 4 racks that are connected through 4 top-of-rack switches, 4 aggregation switches and 4 core switches. Each physical machine has 4 CPU cores, 8GB of memory, 100GB of disk space, and contains a 1 Gbps network adapter. The availability of each equipment (either a server, a switch or a link) is randomly chosen within $\{99.95\%, 99.99\%, 99.995\%, 99.999\%\}$. The arrival of VDC requests follows a Poisson process with the average rate of 0.010 requests/s during non-busy period (12 hours) and 0.020 requests/s during busy periods (12 hours). This reflects the demand fluctuation in data centers (e.g., time-of-the-day effect). We use 3 types of topologies in our experiments: (1) Multi-Tiered, which represents multi-tiered applications, (2) Partition-Aggregate, which represents query-processing applications, (3) MapReduce, which represents batch applications, as shown in Figure 5. The CPU, memory and disk capacity of each VM is generated randomly between $0-4$ cores, $0-2$GB of RAM and $0-10$GB of disk space, respectively. The number of VMs per group is randomly chosen between 1 and 10. The bandwidth requirement of each virtual link is set randomly between 0 and 100 Mbps. The lifetime of VDCs is exponentially distributed with an average of 3 hours. If a VDC cannot be embedded (e.g., due to a lack of resources), it waits in the queue for a duration of 1 hour before it is withdrawn. For each VDC, we randomly select availability requirements among $\{95\%, 99\%, 99.99\%\}$, similar to the ones used by Google App [2]. For convenience, we set $L = 20$, $\gamma_n = 1$, $\lambda_{th} = 0.015$ and $k = 2$.

We first evaluated our heuristics for computing VDC availability with $k = 2$. As shown in Figure 6, For a three-tier application where each tier consists of 5 servers, setting $k = 2$ can already estimate the availability with error less than 0.006%. Furthermore, the importance sampling heuristic in equation (4) achieve better accuracy than the naïve sampling heuristic in equation (3). We found the running time of both heuristics are similar as shown in Figure 7, suggesting they are practical for real applications.

We then evaluated the performance of VDC Planner and Venice without using VM migration. Figure 8 shows the Cumulative Distribution Function (CDF) of VDC availability. It is evident from Figure 8a that the distributions of VDC availability for VDC Planner are nearly identical for all 3 types of VDCs, which agrees with the fact that VDC Planner is availability-oblivious. Compared to VDC Planner, Venice improves the number of type 2 and 3 VDCs satisfying availability requirements by 35%. Similarly, we also evaluated the VDC availability of both algorithms when VM migration is used. The results are shown in Figure 9. It can be seen that Venice again achieves higher availability for VDCs of type 2 and 3. However, the average VDC availability is lower than the case where VM migration is not used. To understand the reason, Figure 10 shows the number of VDCs accepted by each algorithm. It is clear that when VM migration is used, Venice is able to accept a lot more type 2 and 3 VDC requests at the cost of lowering the average VDC availability, as doing

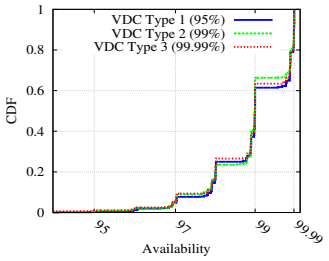


(a) VDC Planner

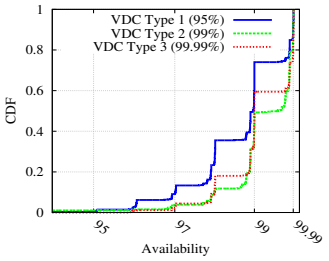


(b) Venice

Fig. 8: VDC Availability not using migration and consolidation



(a) VDC Planner



(b) Venice

Fig. 9: VDC Availability using migration and consolidation

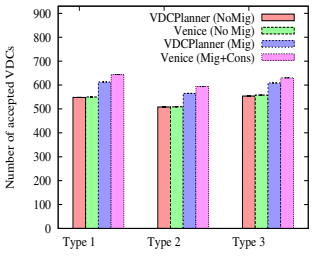


Fig. 10: Acceptance Rate

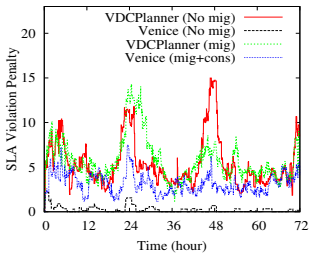


Fig. 11: SLA violation Penalty

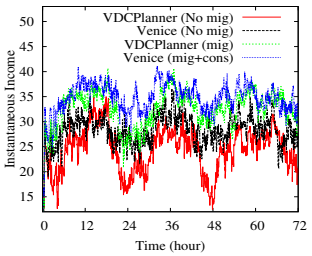


Fig. 12: Instantaneous income

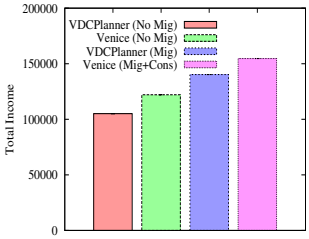


Fig. 13: Total income

so can improve the total revenue gain. Lastly, Figure 11 shows Venice is able to achieve lower service penalty due to failures compared to VDC Planner. Finally, Figure 12 and 13 show the revenue gain of each method. We found Venice can improve the total net income by $10-15\%$ compared to VDC planner.

VIII. Conclusion

As Cloud data centers gain popularity for delivering business critical services, ensuring high availability of cloud services has become a critical concern for cloud providers. However, despite the recent studies on this problem, none of the existing work has considered heterogenous failure characteristics and dependencies among application components within a data center. In this paper, we first developed a practical algorithm for computing VDC availability, and then designed Venice as a framework for achieving high availability of the embedded applications. Through simulations, we show that, compared to availability-oblivious solutions, Venice can increase the number of VDCs satisfying availability requirements by up to 35% and thereby maximize the net income by up to 15%.

IX. Acknowledgement

This work was supported by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network.

References

[1] The Avoidable Cost of Downtime. http://m.softchoice.com/files/pdf/brands/ca/ACOD_REPORT.pdf.

[2] Google Apps Service Level Agreement. http://www.google.com/apps/intl/en/terms/sla.html.

[3] S.K. Au and J.L. Beck. A new adaptive importance sampling scheme for reliability calculations. *Structural Safety*, 21(2):135–158, 1999.

[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM*, 2011.

[5] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani. Data center network virtualization: A survey. *IEEE Communications Surveys Tutorials*, 2013.

[6] A. Birolini. *Reliability Engineering: Theory and Practice*. Springer Berlin Heidelberg, 2010.

[7] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica. Surviving failures in bandwidth-constrained datacenters. In *ACM SIGCOMM*, 2012.

[8] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 1996.

[9] P. Gill and N. Jain. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM*, 2011.

[10] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *ACM SIGCOMM*, August 2009.

[11] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, and P. Sun. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *ACM CoNEXT*, 2010.

[12] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[13] K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In *ACM SoCC*, 2010.

[14] X. Wu, D. Turner, C. Chen, D. Maltz, X. Yang, L. Yuan, and M. Zhang. Netpilot: automating datacenter network failure mitigation. In *ACM SIGCOMM*, 2012.

[15] J. Xu, J. Tang, K. Kwiat, and G. Xue. Survivable virtual infrastructure mapping in virtualized data centers. In *IEEE CLOUD*, 2012.

[16] W. Yeow, C. Westphal, and U. Kozat. Designing and embedding reliable virtual infrastructures. *SIGCOMM CCR*, 2011.

[17] Q. Zhang, M. F. Zhani, R. Boutaba, and J. Hellerstein. Harmony: Dynamic heterogeneity-aware resource provisioning in clouds. In *IEEE ICDCS*, 2013.

[18] M. F. Zhani, Q. Zhang, G. Simon, and R. Boutaba. VDC planner: Dynamic migration-aware virtual data center embedding for clouds. In *IFIP/IEEE IM*, 2013.