Reliable Virtual Machine Placement in Distributed Clouds

Song Yang[1], Philipp Wieder[1] and Ramin Yahyapour[1, 2]
[1]Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Göttingen, Germany
[2]Institute of Computer Science, University of Göttingen, Göttingen, Germany
{Song.Yang, Philipp.Wieder, Ramin.Yahyapour}@gwdg.de

***Abstract*—In nowadays cloud computing systems, leveraging the virtualization technology, the customer's requested data computing or storing service is accommodated by a set of mutual-communicated Virtual Machines (VM) in a scalable and elastic manner. These VMs are placed in one or more datacenter nodes according to nodes' capacities, failure probabilities, etc. The VM placement availability refers to the probability that at least one set of the whole customer's requested VMs operates during the entire requested lifetime. The placed VMs should obey the agreed-upon availability, otherwise the cloud provider may face revenue loss.**

In this paper, we study the problem of placing at most H sets of k requested VMs on minimum number of datacenter nodes, such that the VM placement availability requirement is satisfied and each VM pair has a communication delay no greater than the specified. We prove that this problem is NP-hard. We subsequently propose an exact Integer Nonlinear Program (INLP) and an efficient heuristic to solve this problem. Finally, we conduct simulations to compare the proposed algorithms with two existing heuristics in terms of acceptance ratio, average number of used nodes and running time.

***Index Terms*—Virtual Machine Placement; Availability; Cloud Computing**

I. Introduction

Cloud computing [1] is a distributed computing and storing paradigm, which can provide scalable and reliable service over the Internet for on-demanding data-intensive application (e.g., on-line search or video streaming) and data-intensive computing (e.g., analyzing and processing large volume of scientific data). The key features of cloud computing including "pay-as-you-go" and "elastic service" attract more and more service providers and customers to deploy their workload from their own infrastructures or platforms to public or private clouds.

Distributed cloud systems are usually composed of distributed inter-connected datacenters, which leverage virtualization technology to provide computing and storage service for each on-demanding requests. Once a request arrives, several Virtual Machines (VM) are created in one or more datacenter nodes in order to accommodate it. However, the datacenter node failures [2] caused by hardware (e.g., hard disk, memory module) failures and software problems (e.g., software bugs, configuration errors) may result in the loss of the VMs hosted on it and hence the whole service cannot be guaranteed. An efficient way to overcome it is to create and place more VM replicas, but this approach should also take the nodes' availabilities into account. For instance, if all the VMs together with their replicas are placed at nodes with high failure probability, then the proper service cannot be guaranteed. The VM placement availability, a value between 0 and 1, is therefore important and refers to the probability that at least one set of the whole customer's requested VMs are in their operating state during the entire requested lifetime.

In this paper, we study the Reliable VM Placement (RVMP) problem, which is to place at most H sets of k requested VMs on minimum number of network nodes, such that the VM placement availability is no less than δ and each VM pair has a communication delay no greater than the specified. Our key contributions are as follows:

- We propose a mathematical model to formulate VM placement availability, and prove that the Reliable VM Placement (RVMP) problem is NP-hard.
- We propose an Integer Nonlinear Program (INLP) and a heuristic to solve the RVMP problem.
- We compare the proposed algorithms with two existing heuristics in terms of performance via simulations.

The remainder of this paper is organized as follows: Section II presents the related work and Section III formulates the VM placement availability calculation. In Section IV, we define the Reliable VM Placement (RVMP) problem and prove it is NP-hard. We propose an exact Integer Nonlinear Program (INLP) and a heuristic to solve the RVMP problem in Section V. Section VI provides our simulation results, and we conclude in Section VII.

II. Related Work

A high-level comprehensive survey about resource management in clouds including VM placement is shown in [3].

A. Network-Aware VM Placement

Alicherry and Lakshman [4] deal with 3 problems of placing VMs in geo-distributed clouds. They first study how to place requested VMs on distributed datacenter nodes such that the maximized length (e.g., delay) of placed VM pairs is minimized. A 2-approximation algorithm is proposed to solve this problem when a triangle link length is assumed. They subsequently study how to place VMs on physical machines (racks and servers) within a datacenter in order to minimize inter-rack communication. Assuming the topology of the data center is a tree, they devise an exact algorithm to solve this problem. Finally, they propose a heuristic for partitioning VMs into disjoint sets (e.g., rack) such that communication between VMs belonging to different partition is minimized.

Biran *et al.* [5] address the VM placement problem by minimizing the min-cut ratio in the network, which is defined as the used capacity of the cut links consumed by the communication of VMs divided by the total capacity of the cut links. They prove this problem is NP-hard and propose two efficient heuristics to solve it.

Meng *et al.* [6] address the problem of assigning VMs to slots (CPU/memory on a host) within a datacenter network in order to minimize total network costs. A one to one mapping function is defined as $\pi(x) \rightarrow y$, indicating whether VM x is placed at slot y, where $x, y \in [1, \ldots, k]$ and k denotes the total number of requested VMs and slots. In this context, the network cost is calculated as $\sum D_{ij} C_{\pi(i)\pi(j)}$, where D_{ij} denotes the traffic rate between VMs i and j, and $C_{\pi(i)\pi(j)}$ refers to the communication cost between slot $\pi(i)$ and $\pi(j)$. They prove this problem is NP-hard and propose a heuristic trying to assign VMs with large mutual rate requirement close to each other.

B. Reliable VM Placement

Israel and Raz [7] study the Virtual Machine (VM) Recovery Problem (VMRP). The VMRP is to place the backup VMs for their corresponding servicing VMs on either active or inactive host, which needs to strike a balance between the (active) machine maintenance cost and VM recovery Service Level Agreement (e.g., recovery time). They show that the VMRP is NP-hard, and they propose a bicriteria approximation algorithm and an efficient heuristic to solve it.

Bin *et al.* [8] tackle the VM placement problem by considering k-resiliency constraint to guarantee high availability goals. A VM is marked as k-resilient, if its current host fails and there are up to $k-1$ additional host failures, then it can be guaranteed to relocate to a non-failed host. In this sense, a placement is said to be k-resilient if it satisfies the k-resiliency requirements of all its VMs. They first formulate this problem as a second order optimization statement and then transform it to a generic constraint program in polynomial time.

Zhu *et al.* [9] address the Reliable Resource Allocation (RRA) problem. In this problem, each node has a capacity limit of storing VMs and each link is associated a failure probability value (=1$-$availability). The request is arriving in an on-line fashion, which specifies the requested number of VMs and availability value. The problem is to find a star of a given network, such that the node capacity limit is obeyed and the availability of the star is at least δ. They prove that the RRA problem is NP-hard and propose an exact algorithm as well as a heuristic to solve it. However, the defined problem in [9] does not consider the node's availability and also it restricts to find a star instead of arbitrary subgraph.

Nevertheless, none of above papers quantitatively model the availability of VM placement (and solve the respective reliable VM placement problem), as we do in this paper.

III. VM Placement Availability

The availability of a system is the fraction of time the system is operational during the entire service time. The availability A_j of a network component j (e.g., rack, switch) can be calculated as [10]:

$$A_j = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

where $MTTF$ represents Mean Time To Failure and $MTTR$ denotes Mean Time To Repair. In this paper, we regard the node in the network as a datacenter (similar to [4], [11]). We assume that the datacenter' availability is an overall metric to measure and reflect its availability level, and this fractional value can be calculated based on its physical hardwares' failures[1], typology [13], geographical location, etc. Since calculating the datacenter's availability is out of the scope of this paper, we assume that the datacenter node's availability value is known. It is worthy to mention that the VM placement availability formulation model (together with the later discussed Reliable VM Placement (RVMP) problem) is general and it does not only confine to the scenario discussed in this paper. For instance, it can also be applied to the scenario where we place VMs on a cluster of servers, and each server can be regarded as a node. In this paper, we assume that the node availabilities are uncorrelated/independent. We also assume a more general multiple nodes failure model, which means that at one certain time point, multiple nodes may fail.

Suppose there is a set of k requested VMs V, which are represented by v_1, v_2,..., v_k. Let us use H_i to represent the maximum number of nodes to host VM v_i and denote $H = \max_{i=1}^{k}(H_i)$. Or equivalently, H_i indicates the maximum number of nodes that v_i can be placed on. In the following, we analyze the VM placement availability under two different cases, namely (1) Single Placement: each VM is exactly placed on $H = 1$ node in the network (2) Protected Placement: $\exists v_j \in V$, it can be placed on $H_j > 1$ nodes in the network, so $H > 1$.

In the single placement case, if m nodes with availability A_1, A_2,..., A_m are used for hosting k VMs ($m \leq k$), then the availability (denoted by A_p) of this VM placement is:

$$A_p = A_1 \cdot A_2 \cdots A_m \tag{2}$$

Eq. (2) indicates that since in total k VMs are requested, the availability should take into account the probability that all these k VMs are operational.

In the protected placement case, there exist one or more VMs that can be placed on at most H nodes. Therefore, we regard that a protected placement P is composed of (at most) H single placements. For the ease of clarification, we further define each of H single placements in the protected placement as placement plan p_i, which means the "$i-$th" placing k VMs on m_i nodes, where $1 \leq i \leq H$ and $1 \leq m_i \leq k$. We regard that p_1 as the primary placement plan. We make no difference

[1]Please refer to [2], [12] and the papers therein for more details about physical hardwares' failures characteristics in datacenter networks.

between single placement and placement plan. Since different placement plans may place one or more VMs on the same node, we distinguish the protected placement into two cases, namely (1) *fully protected placement*, for each VM $v \in V$, it is placed by each plan p_i ($1 \leq i \leq H$) at H different nodes, and (2) *partially protected placement*, $\exists v \in V$, it is placed on less than H nodes, i.e., two or more placement plans place v on the same node.

In the fully protected placement case, the availability can be calculated as:

$$A_{PD}^{F} = 1 - \prod_{i=1}^{H}(1 - A_{p_i}) = \sum_{i=1}^{H} A_{p_i} - \sum_{0<i<j\leq H} A_{p_i} \cdot A_{p_j} + \sum_{0<i<j<u\leq H} A_{p_i} \cdot A_{p_j} \cdot A_{p_u} + \cdots + (-1)^{H-1} \prod_{i=1}^{H} A_{p_i} \tag{3}$$

where $A_{p_i} = \prod_{n \in m_i} A_n$ denotes the availability of single VM placement according to Eq. (2). Eq. (3) implies that the availability of H placement plans is equivalent to the probability that at least one single placement (a set of k VMs) is operational in the service-life time.

In the partially protected placement case, if there exists one VM which is placed on less than H nodes, we could regard that it is "sharely" placed by more than one placement plan. For example in Fig. 1, each node is associated with its own availability value and we need to place three VMs (v_1, v_2 and v_3) on it. We set $H_1 = H_2 = 2$ and $H_3 = 1$ for simplicity. We assume that placement plan p_1 to place v_1 and v_2 on node A, and placement plan p_2 to place replica v_1' and replica v_2' on node B, where $v_1' = v_1, v_2' = v_2$. On the other hand, v_3 is only placed on one node. Therefore, we can regard that p_1 and p_2 "sharely" place v_3 on node C.

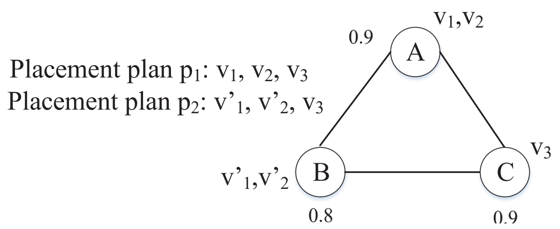

Placement plan p1: v1, v2, v3
Placement plan p2: v'1, v'2, v3
0.9
A
v1,v2
B
v'1,v'2
0.8
C
v3
0.9

Fig. 1: An example of partially VM placement.

However, we cannot directly apply Eq. (3) to calculate its availability, since the availabilities of nodes which hold "shared" VMs will be counted more than once. To amend this, we use a new operator $\circ$[2]. Suppose m different nodes $n_1, n_2, \ldots, n_m$ with availabilities $A_1, A_2, \ldots, A_m$. For a node n_x with availability A_x, $\circ$ can be defined as follows:

$$A_1 \cdot A_2 \cdots A_m \circ A_x = \begin{cases} \prod_{i=1}^{m} A_i & \text{if } \exists n_i = n_x \\ \prod_{i=1}^{m} A_i \cdot A_x & \text{otherwise} \end{cases} \tag{4}$$

[2]Similarly as we did in [14] to calculate the connection availability of partially link-disjoint paths.

Let $\coprod$ denote the $\circ$ operations of different sets, then the availability (represented by A_{PD}^{H}) of H partially placement plans can now be represented as:

$$\begin{aligned} A_{PD}^{H} &= 1 - \coprod_{i=1}^{H}(1 - A_{p_i}) \\ &= 1 - (1 - A_{p_1}) \circ (1 - A_{p_2}) \circ \circ \circ (1 - A_{p_H}) \\ &= \sum_{i=1}^{H} A_{p_i} - \sum_{0<i<j\leq H} A_{p_i} \circ A_{p_j} + \\ &\sum_{0<i<j<u\leq H} A_{p_i} \circ A_{p_j} \circ A_{p_u} + \cdots + (-1)^{H-1} \coprod_{i=1}^{H} A_{p_i} \end{aligned} \tag{5}$$

where A_{p_i} denotes the availability of placement plan p_i and can be calculated from Eq. (2).

To better illustrate it, we take Fig. 2 for example. In Fig. 2, it is assumed that each node can host at most one VM for simplicity and its availability value is labeled on itself. Suppose three VMs v_1, v_2, and v_3 are placed on node s, a and b, respectively, then according to Eq. (2), the availability of this VM placement is $0.95 \cdot 0.9 \cdot 0.9 = 0.7695$. If we set $\delta = 0.85$, this VM placement cannot satisfy customer's requirement. We regard the above VM placement plan as p_1. And we add one more plan p_2, where we place one more replica of v_2 (represented by v_2') on node c, one more replica of v_3 (represented by v_3') on d, and regard p_2 and p_1 together sharely place v_1 on node s. Hence, according to Eq. (5), the overall availability of p_1 and p_2 is $1-(1-0.95\cdot0.9\cdot0.9)\circ(1-0.95\cdot0.8\cdot0.6) = 0.95\cdot 0.9\cdot0.9+0.95\cdot0.8\cdot0.6-0.95\cdot0.9\cdot0.9\cdot0.8\cdot0.6 = 0.85614 > \delta$.

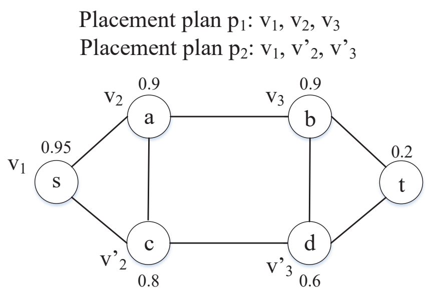

Placement plan p1: v1, v2, v3
Placement plan p2: v1, v'2, v'3
0.9
v2
a
0.9
v3
b
0.95
v1
s
0.2
t
c
v'2
0.8
d
v'3
0.6

Fig. 2: An example of partially VM placement availability calculation.

IV. Problem Definition and Complexity Analysis

A. Problem Definition

Formally, the Reliable VM Placement (RVMP) problem is defined as follows:

Definition 1: Given a complete graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ denotes a set of N nodes and $\mathcal{L}$ represents a set of $L = \frac{N(N-1)}{2}$ links. Each node $N_i \in \mathcal{N}$ has storage upper bound of s_i, and each link $l \in \mathcal{L}$ is associated a delay value d_l. A request is represented by $r(k, c, V, T, \delta)$, where k indicates the number of requested VMs V with demanding capacity

c_v ($v \in V$), T is a $k \times k$ matrix, which specifies the delay constraint between any two VMs, and δ denotes requested availability. The Reliable VM Placement (RVMP) problem is to place at most H sets of k VMs by using minimum number of nodes such that:

1) Each node does not exceed its storage limit.
2) Any two VMs i_1 and i_2 under the same placement plan have a communication delay no greater than $T(i_1, i_2)$.
3) Its VM placement availability is no less than δ.

In the RVMP problem, we assume each VM can be placed at up to H different nodes. Moreover, we only consider the nodes that have the storage capabilities (e.g., datacenter nodes) and ignore some other nodes in the network (e.g., router nodes). In fact, the link between each node pair in the RVMP problem actually may imply a (set of) path(s) which may traverse some other intermediate nodes. Finding reliable and delay-sensitive paths is out of scope of this paper, we therefore assume that the path(s) (represented by link in the RVMP problem) between each node pair is precalculated and highly reliable.

B. Complexity Analysis

Theorem 1: The RVMP problem is NP-hard.

Proof: Let's first introduce the Bin-Packing problem: Given n items with sizes e_1, e_2, …, e_n, and a set of m bins with capacity c_1, c_2, …,c_m, the Bin-Packing problem is to pack all the items into minimized number of bins without violating the bin capacity size. If we assume that the delay of links in the network is 0 and all the nodes have availability 1, then the RVMP problem for $H = 1$ turns into the bin-packing problem, which is NP-hard [15]. ■

Next, we further analyze the complexity of the RVMP problem when the objective and link delay constraint are not taken into account. We consider this problem with and without node storage limits.

- Each node has unlimited storage: In this case, each set of k VMs can be placed on one node and we need to find H nodes in the network to store each set of k VMs. This can be solved in $\binom{N}{H}$ searching when $N > H$ or using N nodes to host N sets of k VMs when $N \leq H$, which is polynomial time solvable.
- Each node has limited storage: In this case, we assume $\prod_{n \in \mathcal{N}} A_n \geq \delta$ and $H = 1$ for simplicity. Therefore, the problem is equivalent to placing k VMs on N nodes in the network such that the node storage constraint is obeyed. This problem is therefore equivalent to the decision version of the Bin-Packing problem, which is NP-hard.

V. Exact Solution and Heuristic

A. Exact Solution

In this subsection, we propose an exact Integer Nonlinear Program (INLP) to solve the RVMP problem. We start by explaining the necessary notations and variables:

INLP notations:

$r(k, c, V, T, \delta)$: a request r which specifies a set of k VMs V. For each $v \in V$, its demanding capacity c_v. T indicates the delay constraint among different VMs, and δ implies the requested VM placement availability.

$\mathcal{N}, \mathcal{L}$: set of N nodes and set of L links, respectively.

H: The maximum number of times for each VM to be placed in the network.

INLP variable:

P_{vn}^h: whether VM v is placed on node n by placement plan h, where $v \in V$, $n \in \mathcal{N}$ and $1 \leq h \leq H$.

Objective:

$$\min \sum_{n \in \mathcal{N}} \left(\max_{1 \leq h \leq H, v \in V} P_{vn}^h \right) \tag{6}$$

Placement constraint:

$$\sum_{n \in \mathcal{N}, 1 \leq h \leq H} P_{vn}^h \geq 1 \quad \forall v \in V \tag{7}$$

Storage constraint:

$$\sum_{v \in V} \left(\max_{h=1}^{H} P_{vn}^h \right) \cdot c_v \leq s_n \qquad \forall n \in \mathcal{N} \tag{8}$$

Delay constraint:

$$d_{(x,y)} \cdot P_{ax}^h \cdot P_{by}^h \leq T(a, b) \quad \forall 1 \leq h \leq H, (x, y) \in \mathcal{L}, \\ a, b \in V : a \neq b \tag{9}$$

VM placement availability constraint:

$$\sum_{h=1}^{H} \prod_{n \in \mathcal{N}} \min_{v \in V} \left(1 - P_{vn}^h + P_{vn}^h A_n \right) - \\ \sum_{1 \leq h < u \leq H} \prod_{n \in \mathcal{N}} \min \left(\min_{v \in V} (1 - P_{vn}^h + P_{vn}^h A_n), \min_{v \in V} (1 - P_{vn}^h + P_{vn}^h A_n) \right) \\ + \cdots + (-1)^{H-1} \left(\prod_{n \in \mathcal{N}} \min_{1 \leq h \leq H} (\min_{v \in V} (1 - P_{vn}^h + P_{vn}^h A_n)) \right) \geq \delta \tag{10}$$

Eq. (6) minimizes the number of total used nodes. For instance, we first calculate the maximum value of P_{vn}^h for node $n \in \mathcal{N}$, and as long as $P_{vn}^h = 1$ for some $1 \leq h \leq H$ and $v \in V$, it means that node n in use to host VM. After that, we take the sum of $\left(\max_{1 \leq h \leq H, v \in V} P_{vn}^h \right)$ for all the nodes in $\mathcal{N}$ and try to minimize this value. Eq. (7) ensures that each one of k requested VMs must be placed in the network. Eq. (8) ensures that each node does not exceed its storage limit when VMs are place on it. Eq. (9) makes sure that any two VMs under the same placement plan should have a delay less than the specified time constraint. Eq. (10) ensures that the availability constraint is obeyed. We also note that Eq. (10) can simultaneously calculate the availability of the fully protected placement, partially protected placement, and single placement. For instance when $H = 2$, Eq. (10) becomes:

$$\prod_{n\in\mathcal{N}}\min_{v\in V}(1-P^1_{vn}+P^1_{vn}A_n)+\prod_{n\in\mathcal{N}}\min_{v\in V}(1-P^2_{vn}+P^2_{vn}A_n)$$
$$-\prod_{n\in\mathcal{N}}\min(\min_{v\in V}(1-P^1_{vn}+P^1_{vn}A_n),\min_{v\in V}(1-P^2_{vn}+P^2_{vn}A_n))$$
$$\geq\delta \tag{11}$$

When $P^1_{vn}=P^2_{vn}$ for all $n\in\mathcal{N}$, Eq. (11) becomes

$$\prod_{n\in\mathcal{N}}\min_{v\in V}(1-P^1_{vn}+P^1_{vn}A_n)\geq\delta$$

which is the VM placement availability constraint for the single placement.

B. Heuristic Algorithm

Algorithm 1: DSG($\mathcal{G}(\mathcal{N},\mathcal{L}), r(k,c,V,T,\delta), H$)

```
1 $VP[h][v][n] \leftarrow 0\ \forall 1 \leq h \leq H, |v| = k, |n| = N$
2 for $h \leftarrow 1$ to $H$ do
3     $VP[h] \leftarrow$ DSGPlace($\mathcal{G}(\mathcal{N},\mathcal{L}), r(k,c,V,T,\delta), H$)
4     $\mathcal{N} \leftarrow \mathcal{N}\backslash\mathcal{N}_x$, where $\mathcal{N}_x$ denotes a subset of the used
        nodes for already found placement plans.
5     if $1-(1-A_{VP[1]})\cdot(1-A_{VP[2]})\dots(1-A_{VP[H]})\geq\delta$ then
6         Call PartiallyDSGPlace($\mathcal{G}, VP[h][k][N], r, H$)
7 Return null
```

Algorithm 2: DSGPlace($\mathcal{G}(\mathcal{N},\mathcal{L}), r(k,c,V,T,\delta), H,$)

```
1 foreach $v_m$ in $V$ $(1 \leq m \leq k)$ do
2     $v_x \leftarrow v_m$, $Q \leftarrow \emptyset$, $\mathcal{G}^m \leftarrow \mathcal{G}$, $P^m[V][N] \leftarrow 0$
3     while $Q.Count < k$ do
4         Sort the nodes in $\mathcal{G}^m$ by their availabilities in the
            decreasing order $n_1$, $n_2$,...,$n_N$
5         Find one node $n_a$ with maximum availability to
            host $v_x$ without violating the delay constraint
            with already placed VMs, such that $s_{n_a} \geq c_{v_x}$
6         if Step 5 succeeds then
7             $P^m[v_x][n_a] \leftarrow 1$, $s_{n_a} \leftarrow s_{n_a} - c_{v_x}$, $A_{n_a} \leftarrow 1$,
                $Q$.Add($v_x$)
8         else
9             Break;
10        $D \leftarrow +\infty$;
11        foreach $v_i$ in $V\backslash Q$ do
12            foreach $v_j$ in $Q$ do
13                if $D > T(v_i,v_j)$ then
14                    $D \leftarrow T(v_i,v_j)$, $v_x \leftarrow v_i$
15 Return $P^m$ with the maximum availability.
```

Our proposed heuristic, called the Delay-Sensitive Greedy (DSG) placement algorithm, is shown in Algorithm 1. Instead of placing VMs on nodes in a conventional way, the logic of DSG is to assign nodes to VMs until all the VMs are covered by the nodes without violating delay constraint. Since we want to use least number of nodes to host VMs to satisfy the availability, we gradually increase the amount of finding placement plans. In what follows, we explain each step of the heuristic algorithm.

Algorithm 3: PartiallyDSGPlace($\mathcal{G}, VP[H][k][N], r, H$)

```
1 Sort the nodes that host VMs in increasing order by
    nodes' availabilities. Denote this set as $\mathcal{N}_y$.
2 foreach $n \in \mathcal{N}_y$ do
3     $VB \leftarrow VP$
4     $VB[h][v][n] \leftarrow 0$ for $1 \leq h \leq H$, $v \in V$ and $n \in \mathcal{N}$
5     foreach placement plan $h = 1...H$ do
6         Try to use its other used nodes to host the VMs
            that are originally placed by it on $n$ if possible.
7     if $1-(1-A_{VB[1]})\cdot(1-A_{VB[2]})\dots(1-A_{VB[H]})\geq\delta$ then
8         $VP \leftarrow VB$, $VB \leftarrow \emptyset$.
9 Return $VP$
```

In Step 1 of Algorithm 1, we first initialize a binary variable $VP[h][v][n]$ representing whether VM $v \in V$ is hosted by node $n \in \mathcal{N}$ under the plan h. After that, for placement plan h, we call Algorithm 2 to place VMs on nodes in Step 3. The purpose of Step 4 is to avoid different plans to have the same placement result. But this will only happen when single node's free capacity is far greater than the VM demanding capacity. That is, all the VMs can be placed on the same node and its remaining free capacity is still large enough so that another set of k VMs can be placed on it. In Step 5, we calculate the availability of $VP[h]$ (containing h placement plans). If availability value of these h placement plans is no less than δ, we call Algorithm 3 trying to return a partially VM placement solution in order to further reduce the number of used nodes. In Algorithm 3, for each node $n \in \mathcal{N}$, we first clear all the VMs resided on it. For each placement plan p, it tries to use its other used nodes to host the VMs that are originally placed by it on n. For simplicity, we apply a greedy approach: for each one of used nodes by placement plan h, it tries to host the VMs originally placed by it from n as many as possible. After that, we calculate whether the whole availability still satisfies δ. If so, we assign this partially placement solution to VP, otherwise we return the original fully placement VP. Next, we will explain the details of Algorithm 2, which is to find single placement.

In Step 1 of Algorithm 2 we start with each $v_m \in V$, assign it to v_x in Step 2. We use a queue Q to store already placed VMs, and initially it is set empty. Besides, we also define variable $P^m[v][n]$ to indicate whether VM $v \in V$ is hosted by node $n \in \mathcal{N}$ corresponding to the placement plan starting with VM v_m. As long as Q's count is less than k, Step 4-Step 9 are going to assign nodes to host unassigned VMs. Step 5 tries to find a node n_a with maximum availability whose capacity should be at least $c(v_x)$. Moreover, if v_x is placed on n_a, it should not violate the delay constraint with already hosted VMs. If it succeeds, in Step 7, the capacity of n_a is reduced by c_{v_x}, the availability of n_a is changed to 1, and v_x is added to Q. The reason to change node's availability is that if some nodes have been used to host the existing VM(s), then the availability for these nodes to host other (unassigned) VMs is 1. So we need to change its "availability" after each iteration of covering VM(s). If such node cannot be found in Step 5, it indicates that not all the VMs are covered and we regard this placement plan should "sharely" place uncovered VMs with one of $H-1$ placement plans found in Algorithm 1. The algorithm then breaks in Step 9. Following that, Step 10-Step 14 search for an unsigned VM, which has a least delay constraint to the already placed VMs, and assigns it to v_x. When Q's count is equal to k, it indicates that all the VMs have been hosted, which means we get a "complete" placement plan. Finally, in Step 15, the algorithm returns a placement plan with the biggest availability from k already determined single placements.

The time complexity of Algorithm 2 can be calculated like this: There are k VMs in total in Step 1, and Step 3 has also k iterations. Sorting algorithm for instance like insertion sort in Step 4 takes $O(N\log(N))$ time, and Step 5 has a complexity of $O(N)$. Step 9-Step 13 consume at most $O(k^2)$ time. Therefore, the whole complexity of Algorithm 2 is $O(k^2(N\log(N)+k^2))$. In Algorithm 3, Step 1 consumes $O(N\log(N))$ time via insertion sort and Step 2-8 consume $O(N^2H)$ time, leading to a whole complexity of $O(N(\log N+NH))$. Consequently, the whole time complexity of Algorithm 1 is $O(k^2H(N\log(N)+k^2))$, since it calls at most H times of Algorithm 2.

VI. Simulation

A. Simulation Setup

We conduct simulations on a 16-node complete network[3]. If we set c (VM demanding capacity) relatively too small, then by placing as many VMs as possible on one node, we may obtain a solution. If we set c relatively too big, then the solution may not exist. Therefore, we let the node's capacity at most three times of the requested capacity of one single VM, by which we want to challenge the algorithm to find the solution. Consequently, the simulation parameters are set like this: the node capacities are randomly distributed between 100 and 200 units, and the node availabilities are randomly distributed among the set of $\{0.99, 0.999, 0.9995, 0.9999\}$. The link delay $d_l \in [10, 20]$. For each request $r(k, c, V, T, \delta)$, $k \in [3, 5]$, $c \in [60, 130]$, each element in the delay matrix T is between 15 and 25, and δ is in the set of $\{0.999, 0.9999, 0.99999, 0.999999\}$. We randomly generate 100 requests for $k = 3, 4, 5$. We set $H = 2$ and 3.

We compare our exact INLP and heuristic DSG with two heuristics, namely (1) Greedy Placement (GP) and (2) Random Placement (RP). These 2 algorithms follow the similar routine with Algorithm 1, except: (1) In Step 6, they directly return the placement result if its availability is satisfied, instead of checking partially placement solution, and (2) They call different heuristics in Step 3 (different from Algorithm 2), which we specify as follows:

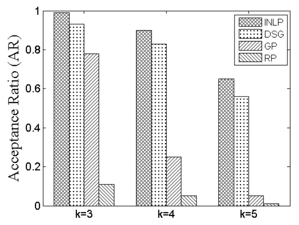
- GP: It first selects a node with the greatest availability and places as many VMs as possible on it under its storage limit. It then selects the second largest availability node and places remaining VMs as many as possible, which should also satisfy the delay constraint with already placed VMs. This procedure continues until all the VMs are placed or all the nodes have been iterated.
- RP: It first randomly selects a node, and places as many VMs as possible on it without exceeding its capacity. Then it randomly selects the second node, and places remaining VMs as many as possible without violating node's capacity constraint and the delay constraint with already placed VMs. This procedure continues until all the VMs are placed or all the nodes have been iterated.

The simulations are run on a desktop PC with 2.7 GHz and 8 GB memory. We use IBM ILOG CPLEX 12.6 to implement the proposed INLP and C# to implement the heuristics.

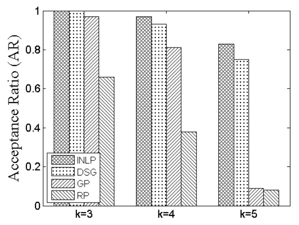
[3]During simulations we found that the INLP for $N \geq 20$ keeps on running for at least one day and does not terminate with a feasible solution. Therefore, we choose $N = 16$ for evaluation of both the INLP and heuristics.

B. Simulation Results

We first evaluate the performance of the algorithms in terms of Acceptance Ratio (AR), which is defined as the number of accepted requests over all the requests. Fig. 3 shows that the exact INLP always achieves the highest AR. DSG has a close performance with INLP, and it outperforms all the other two heuristics. Besides, we notice that for the same algorithm, it achieves higher AR value when H increases, since more VM replicas are allowed to be placed for a higher H.



(a) $H = 2$



(b) $H = 3$

Fig. 3: Acceptance Ratio over 100 requests: (a) $H = 2$ (b) $H = 3$.

Next, we compare the algorithms in terms of Average Number of Used Nodes (ANUN). The ANUN is defined as the total number of nodes consumed by all the accepted requests divided by the number of accepted requests. To have a fair comparison, we only consider the requests accepted by all four algorithms. Considering that GP and RP have a low acceptance ratio when $H = 2$, which means that the common accepted requests are fewer, so we only present the ANUN result when

$H = 3$. We also omit the ANUN value for GP and RP when $k = 5$, since the number of their common accepted requests when $k = 5$ are very small. From Fig. 4, we see that the INLP achieves the minimum value of ANUN, and our proposed DSG obtains the second lowest ANUN value. RP obtains a lower ANUN value than GP when $k = 4$, since it is regarded to place more shared VMs. From above, we observe that even under the constrained simulation setup, the exact INLP can always accept most requests and consume least amount of nodes as well, which validates its correctness.

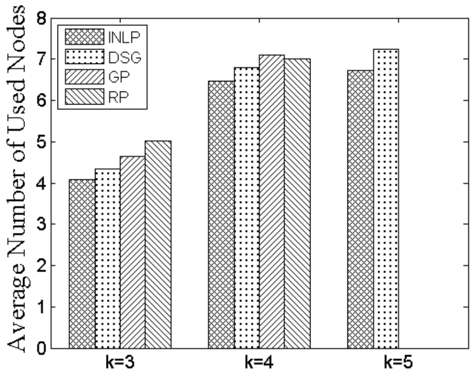
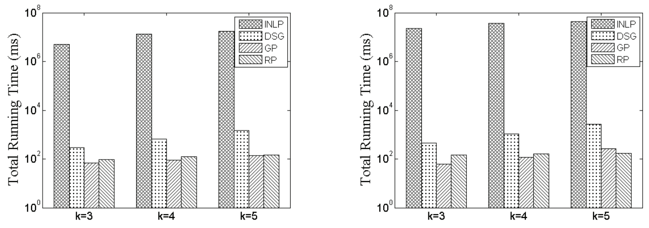


Fig. 4: Average number of used nodes when $H = 3$, where the value of GP and RP are omitted for $k = 5$.

Finally, Fig. 5 presents the total running time for 100 requests (in log scale). The INLP is significantly more time consuming than all the other 3 heuristics. The DSG, on the other hand, has a slightly higher running time than the other two heuristics, but it pays off by having a higher AR as shown in Figs. 3a and 3b, and lower ANUN shown in Fig. 4. In all, we conclude that the exact INLP can be used as an optimal solution when the computation speed is not a big concern. However, as the problem size increases, its running time will increase exponentially. On the contrary, our proposed DSG is a good compromise between performance and running time, and it is the preferred choice for when VM placement request needs to be computed on-the-fly.



(a) $H = 2$

(b) $H = 3$

Fig. 5: Total running time over 100 requests: (a) $H = 2$ (b) $H = 3$.

VII. Conclusion

In this paper, we have studied the RVMP problem, which is to place at most H sets of k requested VMs on minimum number of nodes, such that the VM placement availability is no less than δ and each VM pair has a communication delay no greater than the specified. We have proved that the RVMP problem is NP-hard. To solve it, we have proposed an exact INLP as well as an efficient heuristic. The simulation results reveal that, our proposed heuristic can always achieve a better performance in terms of acceptance ratio and average number of used nodes than the other two heuristics, and it only requires a slightly higher running time. On the other hand, the exact INLP can always achieve the best performance, but its running time is significantly larger than all the heuristics.

Acknowledgement

This research was funded by the joint EU FP7 Marie Curie Actions Cleansky Project, Contract No. 607584.

References

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. of IEEE Grid Computing Environments Workshop*, 2008, pp. 1–10.
[2] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, 2011, pp. 350–361.
[3] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, pp. 1–53, 2014.
[4] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. of IEEE INFOCOM*, 2012, pp. 963–971.
[5] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware vm placement for cloud systems," in *Proc. of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2012, pp. 498–506.
[6] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.
[7] A. Israel and D. Raz, "Cost aware fault recovery in clouds," in *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2013, pp. 9–17.
[8] 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 31st IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011, pp. 700–709.
[9] Y. Zhu, Y. Liang, Q. Zhang, X. Wang, P. Palacharla, and M. Sekiya, "Reliable resource allocation for optically interconnected distributed clouds," in *Proc. of IEEE International Conference on Communications (ICC)*, 2014, pp. 3301–3306.
[10] W. Zou, M. Janić, R. Kooij, and F. Kuipers, "On the availability of networks," *Proc. of BroadBand Europe*, 2007.
[11] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 762–772, 2013.
[12] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *Proc. of IEEE INFOCOM*, 2014, pp. 289–297.
[13] Y. Liu, J. K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi, *Data center networks: Topologies, architectures and fault-tolerance characteristics*. Springer Science & Business Media, 2013.
[14] S. Yang, S. Trajanovski, and F. A. Kuipers, "Availability-based path selection and network vulnerability assessment," *Networks*, vol. 66, no. 4, pp. 306–319, 2015.
[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman & Co., 1979.