FT-VMP: Fault-Tolerant Virtual Machine Placement in Cloud Data Centers

Christopher Gonzalez and Bin Tang
Department of Computer Science, California State University Dominguez Hills, Carson, CA 90747, USA
Email: cgonzalez393@toromail.csudh.edu, btang@csudh.edu

***Abstract*— Virtual machine (VM) replication is an effective technique in cloud data centers to achieve fault-tolerance, load-balance, and quick-responsiveness to user requests. In this paper we study a new fault-tolerant VM placement problem referred to as FT-VMP. Given that different VM has different fault-tolerance requirement (i.e., different VM requires different number of replica copies) and compatibility requirement (i.e., some VMs and their replicas cannot be placed into some physical machines (PMs) due to software or platform incompatibility), FT-VMP studies how to place VM replica copies inside cloud data centers in order to minimize the number of PMs storing VM replicas, under the constraints that i) for fault-tolerant purpose, replica copies of the same VM cannot be placed inside the same PM and ii) each PM has a limited amount of storage capacity. We first prove that FT-VMP is NP-hard. We then design an integer linear programming (ILP)-based algorithm to solve it optimally. As ILP takes time to compute thus is not suitable for large scale cloud data centers, we design a suite of efficient and scalable heuristic fault-tolerant VM placement algorithms. We show that a) ILP-based algorithm outperforms the state-of-the-art VM replica placement in a wide range of network dynamics and b) that all our fault-tolerant VM placement algorithms are able to turn off significant number of PMs to save energy in cloud data centers. In particular, we show that our algorithms can consolidate (i.e., turn off) around 100 PMs in a small data center of 256 PMs and 700 PMs in a large data center of 1028PMs.**

Keywords – Fault-Tolerance, Virtual Machine Placement, Cloud Data Center

I. Introduction

Virtual machine (VM) replication is an effective technique in cloud data centers [14]. By replicating VMs and placing their replica copies into different physical machines (PMs) of the cloud data centers, it not only achieves fault-tolerance of VM applications in the event of PM failures, reduces cloud user access latencies, but also distributes the user requests to VM copies at different PMs thus reducing and balancing server loads. As such, VM replication has been a popular practice in production data centers such as Amazon Simple Storage Service (Amazon S3) [1] and Microsoft Azure [2]. For example, Amazon S3 Replication can automatically replicate S3 objects across different AWS Regions while data in an Azure geo-redundant storage is always replicated three times in the primary region [1, 2].

However, VM replications bring a few challenges to cloud data center operators. First, for the fault-tolerant purpose, it is preferred that multiple replica copies of the same VM application are placed into different PMs (we refer to this as *fault-tolerance constraint* of VMs). This way, in the event of failures of some PMs, it is hoped at least one copy of each VM (either the original one or the replica copy) can still survive and execute to satisfy the user requests. Consequently, such fault-tolerance requirement needs more PMs to be turned on in order for the VM copies to be accessed. This unfortunately exacerbates the power consumption in cloud data centers, which already generates between 1.1% and 1.5% of the total electricity use worldwide and is projected to rise even more [21]. In particular, the energy consumption of PMs contributes a large portion (upto 40-60%) of the total consumption of a data center. Second, as more VM copies need to be created and placed unto cloud data centers, it consumes more cloud resources (i.e., CPUs, storages and memories, and I/Os). However, each PM in cloud data center still has limited amount of such cloud resources. We refer to it as *resource capacity constraint* of PMs. Third, due to software and platform compatibility, not all the VM replica copies can be placed onto all of the PMs. For example, VMware ESXi [6], a type-1 hypervisor for deploying and serving virtual computers, requires the users to use the VM compatibility setting to select the ESXi host when creating a new VM or upgrading an existing VM. The compatibility setting determines the virtual hardware available to the VM including virtual PCI slots, maximum number of CPUs, maximum memory configuration available to the VMs, and other characteristics. We refer to this requirement as *compatibility constraint* of VMs to PMs.

These challenges become more formidable in large scale production data centers, which run hundreds of thousands of VMs on many popular virtualization platforms including Citrix XenServer [3], Microsoft Hyper-V [4], Red Hat KVM [7], and VMware vSphere [8] while accommodating a wide range of service level agreements (SLAs) of fault-tolerance from large number of cloud users. Therefore how to achieve fault-tolerant and energy-efficient VM placement while addressing aforesaid challenges becomes an important research problem.

In this paper, we propose FT-VMP: fault-tolerant virtual machine placement to tackle those challenges. In our model, a set of VM applications submitted by cloud users, referred to as *original VMs*, have already been created and placed inside some PMs of the cloud data center. We assume that the PMs with original VMs cannot be turned off as original VMs are running on these PMs. The fault-tolerance SLA requires that a number of replica copies (referred to as *VM replicas*) to be

made and placed into the cloud data center.[1] The goal of FT-VMP is to place required number of VM replica copies into the cloud data centers in order to minimize the total number of active PMs while satisfying the fault-tolerance constraint of VMs, compatibility constraint of VMs to PMs, and the resource capacity constraint of PMs.

We formally formulate FT-VMP as a graph theoretical problem and prove that it is NP-hard. We then design an integer linear programming (ILP)-based algorithm to solve it optimally. As ILP is not scalable to large scale cloud data centers, we design a suite of efficient and scalable heuristic algorithms for FT-VMP. We compare our algorithms with the state-of-the-art fault-tolerant server consolidation algorithm [18], which considers all above three constraints and with the same goal of minimizing active PMs. We show that a) ILP-based algorithm outperforms the state-of-the-art VM replica placement in a wide range of network dynamics and b) that all our algorithms are not only fault-tolerant, but also energy-efficient, being able to turn off PMs to save energy. In particular, we show that our algorithms can consolidate around 100 PMs in a small data center of 256 PMs and 700 PMs in a large data center of 1028PMs.

Finally, we study another relevant problem – Given an instance of FT-VMP, can all the replica copies be placed inside the cloud data centers without violating the fault-tolerance, compatibility, and resource capacity constraints? We refer to it as *feasibility problem* of FT-VMP. Finding if any given instance of FT-VMP is feasible is critical to achieve efficient operation of a cloud data center. As it enables data center operators to determine whether the required fault-tolerance SLAs from users can be satisfied or not before implementing them, it saves both capital and operational expenditures of cloud data centers. We formulate this feasibility problem formally and solve it by designing a maximum flow-based efficient algorithm. The algorithm is executed on a flow network that is intricately transformed from the data center network while encoding all the constraints presented in FT-VMP. After feasibility is being checked that it is possible to achieve fault-tolerance VM placement with this instance, then we apply different algorithms to solve fault-tolerance VM placement designed in this paper. To the extend of our knowledge, all the existing fault-tolerant VM placement work didnot identify and solve this important problem.

Paper Organization. The rest of the paper is organized as follows. Section II gives an overview of the related literature and the state-of-the-art fault-tolerant VM placement techniques. In Section III, we formulate the FT-VMP problem, prove its NP-hardness, and solve the feasibility problem of FT-VMP. Section IV presents the different algorithms for FT-VMP, including ILP-based optimal algorithm and two efficient heuristic algorithms. In Section V, we compare all the proposed algorithms with the state-of-the-art [18] and discuss the results in details. Section VI concludes the paper with some possible future work.

[1]In literature, original and replica VMs are also referred to as primary and backup VMs respectively [24].

II. **Related Work**

There are different fault-tolerance metrics and mechanisms proposed for VM placement in cloud data centers. Machida et al. [20] proposed a k-fault tolerance VM placement method for consolidated server systems. It ensures that the simultaneous failure of any k PMs would not make the service unavailable. As k fault-tolerance is configured based on the hardware provisions, this metric is from the cloud service provider's perspective. In contrast, as our model addresses fault-tolerance from SLA perspective where different cloud users can request different number of replica copies, it provides more flexible customization of fault-tolerance for cloud users. Recently Zhou et al. [24] et al. further improved k-fault tolerance by tackling failure recovery of VM replica placement. It proposed a redundant VM placement approach to minimize network resource consumption to reassign tasks from a failed primary VM to a backup VM under the k-fault tolerance constraints. However, it only focused on the network resource (e.g., bandwidth and energy) consumption and does not consider energy consumption of PMs.

Zhou et al. [25] observed that k-fault tolerance replication strategies ignore the switch failures and thus cannot achieve the best effect. They designed a (m, n)-fault tolerance VM placement algorithm where m redundant PMs are prepared so as to keep n PMs running at any m PM failures. Luo et al. [19] studied single-point failure tolerance and proposed a multi-objective particle swarm optimization algorithm. Goudarzi and Pedram [14] considered another different fault-tolerance model. They assumed that each client can have multiple VM copies placed and executed simultaneously on different PMs, where each VM copy computing one part of the task requested by the client. They considered a linear PM energy model wherein more load on a PM means more energy consumption. As different PMs have different types with different CPU power, memories, and bandwidths, it needs to compute how many VM copies are needed for each client and where to place them, in order to minimize the energy cost of the active PMs.

All above VM replication work considered either network resource consumption or PM energy consumption, but not both. In our previous work [18], we proposed an energy-efficient VM replication mechanism that considers power consumptions on PMs as well as on edges and switches inside the data center networks. It has two steps. First it proposed a minimum cost flow-based optimal algorithm that minimizes the energy consumption of distributing VM replica copies inside the data center networks. It then proposed a server consolidation algorithm to migrate VM copies around PMs in order to turn off more PMs to save energy. However, the proposed server consolidation algorithm (Algorithm 1, [18]) is a heuristic algorithm that does not have any performance guarantee. In this paper, we formally tackle this problem and propose one ILP-based optimal solution and two efficient heuristic algorithms. We show via extensive experiments that they outperform our previous work most of the time.

A few recent cloud fault-tolerant systems that are not directly related to VM replication and placement are reviewed below. Zheng et al. [23] proposed a component ranking framework, named FTCloud, for building fault-tolerant cloud applications. It employed component invocation structures and invocation frequencies for making significant component ranking. Zhou et al. [26] enhanced network and storage resource usage in a cloud data center used checkpointing, a basic fault-tolerant mechanism that periodically saves the execution state of a VM as an image file. In particular, the identical parts of all VMs that provide the same service are check-pointed as service image s that can be shared by other VMs to reduce the storage resource consumption. Xu et al. [22] proposed that multiple correlated VMs and their backups are grouped together to form a Survivable Virtual Infrastructure (SVI) for a service or a tenant. It determined how to map each SVI to a physical data center network such that operational costs are minimized subject to that each VM's resource requirements and bandwidth demands between VMs can be guaranteed.

The most relevant work to ours is by Gupta et al. [15]. They modelled the server consolidation problem as a variant of the bin packing problem [12] where items to be packed are the servers being consolidated and bins are the target servers. They considered both item-item and bin-item incompatibilities and proposed a two-stage heuristic algorithm. Their item-item and bin-item incompatibilities are essentially the fault-tolerance constraints of VMs and compatibility constraints of VMs to PMs addressed in our paper. However, in our work, as each VM and its replicas are of unit sizes, it is not a variant of the bin packing problem (more discussion in Theorem 1). With this assumption, instead of giving a two-stage heuristic algorithm, we are able to present an optimal ILP based solution as well as proving the NP-hardness of this problem. Finally, it didnot specifically target the problem of fault tolerance VM placement and its related feasibility problem, which are the topics of this paper.

III. Problem Formulation of FT-VMP

In this section, we first present the system models and problem formulation of FT-VMP. We then prove that FT-VMP is NP-hard by reducing from the vertex coloring problem [12]. Finally, we study feasibility problem of FT-VMP, and design an efficient algorithm to solve it.

A. Problem Formulation and NP-hardness.

System Models. We illustrate the problem and solutions using fat tree data center topology [9], shown in Fig. 1. However, as our problem and solutions are applicable to any data center topologies, we model a data center as a general graph $G(V = V_p \cup V_s, E)$. Here $V_p = \{1, ..., |V_p|\}$ is a set of $|V_p|$ PMs and $V_s = \{|V_p|+1, ..., |V|\}$ is a set of $|V|-|V_p|$ switches. E is a set of edges; each edge represents a physical link that exists either between the switches or between switches and PMs.

In our model, there are initially a set of l distinct *original VMs* $V_m = \{v_1, v_2, ..., v_l\}$ that are submitted to the data center and they are currently be executed. Here v_j $(1 \le j \le l)$ is stored at its *source PM* $s(j) \in V_p$. A source PM can have multiple original VMs. We denote the set of source PMs as $V_d \subseteq V_p$. For each VM to run, it needs one unit of cloud resources (i.e., CPUs, memories, and disk I/O) for execution. Let rc_i denote the *resource capacity* of PM $i \in V_p$; that is, the total number of original or replica VM copies PM i can store is rc_i. Thus if i is a source PM of some original VMs, its available resource capacity becomes $rc_i - |\{1 \le j \le l | s(j) = i\}|$. Due to diverse fault-tolerance requirement, it requires to place $r_j \ge 0$ copies of v_j into the data center (when $r_j = 0$, it does not need to place any replica copies besides the original VM v_j). Let $R = \max\{r_1, r_2, ..., r_l\}$. Denote the k^{th} replica copy of $v_j \in V_m$, where $1 \le j \le l$ and $0 \le k \le r_j$, as $v_{j,k}$ ($v_{j,0}$ is the original copy v_j). In Fig. 1, there are $l = 4$ original VMs, each has two replicas around it (thus $r_j = 2, 1 \le j \le 4$). Table I shows all the notations.

TABLE I
NOTATION SUMMARY

Notation	Description
V_p	$V_p = \{1, ..., \|V_p\|\}$ is the set of $\|V_p\|$ PMs
V_s	$V_s = \{\|V_p\|+1, ..., \|V\|\}$ is the set of $\|V\| - \|V_p\|$ switches
V_m	$V_m = \{v_1, v_2, ..., v_l\}$ is the set of l original VMs
$s(j)$	$s(j) \in V_p$ is the source PM of original VM v_j
V_d	$V_d \subseteq V_p$ is the set of source PMs
rc_i	Resource capacity of PM i, $1 \le i \le \|V_p\|$
rc_i^e	Effective resource capacity of PM i, $1 \le i \le \|V_p\|$
r_j	$r_j \ge 0$ is the number of replica copies required for VM v_j
R	$R = \max\{r_1, r_2, ..., r_l\}$
$\mathcal{C}(j)$	$\mathcal{C}(j) \subset V_p$ is the *compatibility set* of v_j, i.e., the set of PMs that v_j and its replica can be placed into
$v_{j,k}$	The k^{th} replica copy of VM v_j, $v_{j,0}$ is original VM v_j
$r(j,k)$	Placement function that places $v(j,k)$ at PM $r(j,k)$
y_i	$y_i = 0$ if PM i is turned off, $y_i = 1$ if i is turned on
$x_{i,j,k}$	$x_{i,j,k} = 1$ if $r(j,k) = i$, 0 otherwise

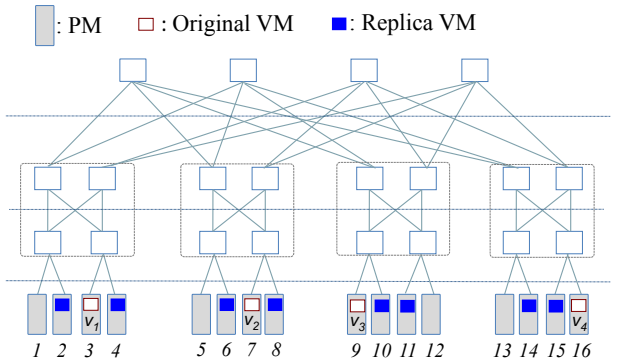

Fig. 1. A $k = 4$ fat tree topology. $l = 4$ original VMs: (v_1, v_2, ..., v_4) are located at PM 3, 7, 9, 16, respectively. $r_j = 2$.

We note that although rc_i is the resource capacity available at PM i, it is possible that not all of its resources can be utilized at i. The fault-tolerance constraint, which stipulates that multiple copies of the same VM be placed at different PMs in order to survive PM failures, incurs two consequences. First, any original VM and its replicas must be placed onto different PMs, thus it must be that $R+1 \le |V_p|$; otherwise, it is infeasible for FT-VMP (we will give detailed analysis of the feasibility condition later). Second, as there are l original VMs, a PM can store at most l VM copies, each from a different

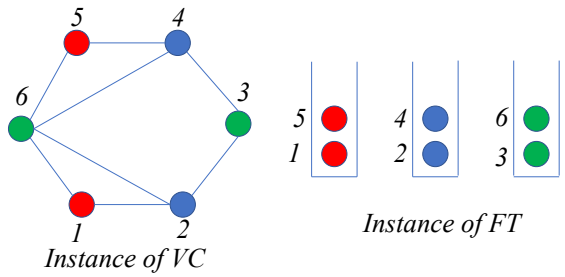

Fig. 2. NP-hardness proof of FT-VMP by reduction from vertex coloring. The graph on the left is 3-colorable if and only if in the corresponding instance of the FT on the right, three PMs can hold all the VM replicas such that the replicas with conflicts (i.e, nodes with different colors) must be put into different PMs.

VM, even though its storage capacity could be larger than l. We thus define *effective resource capacity* of PM i, denoted as rc_i^e, as the maximum resource capacity of i that can be used to store VMs. rc_i^e is the smaller value between the available resource at i and the number of the VMs it can further store (besides its own stored original VMs if it has). That is, $rc_i^e = \min\{rc_i - |\{1 \leq j \leq l | s(j) = i\}|, l - |\{1 \leq j \leq l | s(j) = i\}|\}$.

We define *active PMs* as PMs that have at least one VM copy (either original or replica) after VM placement. Otherwise, it is *inactive* therefore can be turned off. For any source PM $s(j) \in V_d$, $1 \leq j \leq l$, as some original VMs have already been submitted and are currently being executed there, they are considered as active PMs thus cannot be turned off.

Problem Formulation of FT-VMP. Recall that $v_{j,k}$ is the k^{th} replica copy of VM $v_j \in V_m$. We define *placement function* $r : V_m \times \{1, 2, ..., R\} \rightarrow V_p$, indicating that $v_{j,k}$ is placed to *destination PM* $r(j,k) \in V_p$. As the original VM v_j is located at $s(j)$, we have $r(j,0) = s(j)$. Let $y_i = 0$ and 1 indicate that PM i is inactive and active respectively after VM placement r. FT-VMP is to find r that minimizes the total number of active PMs, i.e., $\min \sum_{i=1}^{|V_p|} y_i$, under

1) *fault-tolerance constraint* of VMs: For fault-tolerant purpose, replica copies of the same VM cannot be placed into a) the same PMs and b) the source PM where the original VM is located. That is, $r(j,k) \neq r(j,k^{'}), 1 \leq j \leq l, 0 \leq k \neq k^{'} \leq r_j$, and
2) *resource capacity constraint* of PMs: $|\{1 \leq j \leq l | r(j,k) = i\}| \leq rc_i^e, \forall i \in V_p, 1 \leq k \leq r_j$, and
3) *compatibility constraint* of VMs to PMs: some replica copies cannot be placed into some PMs due to software or platform incompatibility. We define the *compatibility set* of VM v_j as the set of PMs that v_j and its r_j replica copies can be placed upon, and denote it as $\mathcal{C}(j) \subset V_p$.

Theorem 1: FT-VMP is NP-hard.

Proof: We reduce vertex coloring (VC) problem [12], which is NP-hard, to a special case of FT-VMP. In this special case, referred to as FT, we only consider the fault-tolerance constraint and assume no original VMs are placed in the PMs. Given a graph $G = (V, E)$, VC finds minimum number of colors to color all the vertices in V such that any two nodes u and v, $(u, v) \in E$, have different colors. Its decision version is given an integer $k \leq |V|$, VC finds if G is k-colorable; that is, if there exists a partition of V into k subsets $V_1, \ldots, V_k$, such that any two nodes u and v with $(u, v) \in E$ do not belong to the same subset V_j, $1 \leq j \leq k$. The minimum value of k is called the *chromatic number* of G, denoted as $\chi(G)$.

Given any instance of $G(V, E)$ in VC, we construct an instance of cloud data center in FT as follows. We let all the vertices in V be all the VM replica copies to be placed, and let different colors assigned to vertices in VC represent different PMs in FT. Further, two replicas are with conflict, meaning they cannot be placed into the same PM, if their corresponding vertices in V are connected by an edge. This construction is demonstrated by the graph in Fig. 2. It shows that the graph is 3-colorable if and only if three PMs are needed to hold all the replicas, such that the nodes (i.e., replicas) with different colors are put into different PMs. We claim that a $\chi(G) = k$ in VC if and only if that all the replica copies can be placed into k PMs in FT.

First, if $\chi(G) = k$, then k colors can be assigned to nodes in V such that two nodes u and v with $(u, v) \in E$ have two different colors. As different colors represent different PMs and u and v are colored differently, the two corresponding VM replica copies (which have conflict) cannot be placed into the same PM. Also, all the VM replica corresponding to vertices with the same color can then be placed in to the same PM denoted by that color. Therefore if $\chi(G) = k$ in VC, then all the replica copies can be placed into k PMs in FT while no two replicas with conflict are placed into the same PM.

On the other hand, if all the replica copies can be placed into k PMs in FT, it must be that any two replicas with conflict are not placed into the same PM. This shows that their corresponding vertices in VC, which are connected by an edge, are colored with different colors. As k PMs are able to store all the replica copies in FT, it shows that G is k-colorable in VC. ∎

Note that FT-VMP resembles well-known bin packing packing [12] in that it packs VM replica copies into PMs while minimizing number of PMs. In bin packing, items of different sizes are packed into bins with the goal of minimizing the number of bins used. We stress that the "varying item sizes" in bin packing is the key premise that makes it NP-hard. In FT-VMP, however, as we assume that all the VMs and all their replicas have the same unit size, we cannot prove the NP-hardness of FT-VMP by reducing from bin packing. The computational intractability of FT-VMP comes from its fault-tolerance constraint of VMs and compatibility constraint of VMs to PMs. If we consider different VM sizes, FT-VMP becomes a new variant of bin packing then techniques working for bin packing might become relevant.

B. Feasibility Problem of FT-VMP.

Feasibility of FT-VM asks the following question – Given any instance of FT-VMP, that is, the fault-tolerance requirement of cloud users for VM replication and the resource capacity and compatibility constraints, is it possible to place all the replica copies into the cloud data center to satisfy the fault-tolerance requirement without violating the other

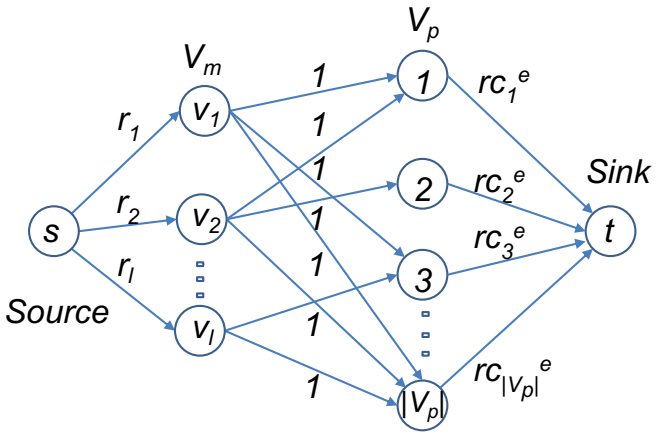

Fig. 3. Flow network $G'(V', E')$ to check the feasibility of FT-VMP. The value on each edge is its capacity. Note there is no edge between $v_j \in V_m$ and $i \in V_p - \mathcal{C}(j)$ if PM i is not in v_j's compatibility set $\mathcal{C}(j)$.

two constraints? We answer this question by designing a maximum-flow based technique on a flow network that is transformed from the data center network. Below we show how to transform the data center topology $G(V, E)$ of Fig. 1 into a flow network $G'(V', E')$ of Fig. 3.

i). $V' = \{s\} \cup \{t\} \cup V_m \cup V_p$, where s is a source node, t is a sink node, V_m is the set of original VMs, and V_p is the set of PMs.

ii). $E' = \{(s, v_j) : v_j \in V_m\} \cup \{(i, t) : i \in V_p\} \cup \{(v_j, i)), \forall v_j \in V_m, i \in \mathcal{C}(j)\}$. We note that between nodes V_m and V_p, there only exists edges between $v_j \in V_m$ and PM in v_j's compatibility set $\mathcal{C}(j)$.

iii). Set the capacity of edge (s, v_j) as r_j, number of replica copies for VM v_j. Set the capacity of edge (i, t) as m_i^e, the effective storage capacity of PM i. Set the capacity of edge (v_j, i), $v_j \in V_m$ and $i \in V_p$, as 1.

We have $|V'| = l + |V_p| + 2$ and $|E'| = l + |V_p| + l \cdot |V_p| - \sum_j^l |\mathcal{C}(j)|$. Next we claim that if the maximum amount of flow computed in above flow network (Fig. 3) is the total number of replica copies to be placed in the data center, then FT-VMP is feasible.

Theorem 2: Given any instance of FT-VMP, if the maximum flow obtained from the corresponding flow network is $\sum_{j=1}^{l} r_j$, then this instance is feasible.

Proof: We show that if the maximum flow computed in flow network Fig. 3 is $\sum_{j=1}^{l} r_j$, it must be that r_j copies of VM v_j, $1 \le j \le l$, are placed inside the cloud data center Fig. 1 while satisfying all the three constraints viz. fault-tolerance constraint, compatibility constraint, and resource capacity constraint.

First, when the amount of flow out of s is $\sum_{j=1}^{l} r_j$, as the edge capacity of (s, v_j) is r_j, it must be that there are r_1 amount of flow on edge (s, v_1), r_2 amount of flow on (s, v_2), ..., and r_l amount of flow on (s, v_l). This indicates that r_j copies of VM v_j are placed inside the cloud data center.

Second, since the capacities of all the edges (v_j, i), $v_j \in V_m$ and $i \in V_p$, is 1, it guarantees that different replica copies of the same VM are placed on different PMs, satisfying the fault-tolerance constraint. Meanwhile, as there does not exist an edge between VM node v_j and PM node $i \in V_p - \mathcal{C}(j)$, replica copies of VM v_j can only be placed to PMs in its compatibility set $\mathcal{C}(j)$, satisfying compatibility constraint of VMs.

Finally, as the edge capacity of edge (i, t) is m_j^e, it only allows at most m_j^e amount of flow come in and out of node $i \in V_p$. This stipulates that PM node i can not store more VM copies than what its effective resource capacity m_j^e allows. This satisfies the resource capacity constraint of PMs. ∎

Maximum Flow Algorithms. There are two kinds of maximum flow algorithms viz. augmenting path based [10, 11, 16] and push-relabel based [13, 17]. While both algorithms can yield strongly polynomial running time, push-relabel in general is more flexible and efficient than augmenting path. We thus adopt push-relabel method for maximum flow in our implementation. As the time complexity of push–relabel maximum flow algorithm is $O(|V'|^2 \cdot |E'|)$ for a flow network $G(V', E')$, and $|V'| = O(l + |V_p|)$ and $|E'| = O(l \cdot |V_p|)$, the running time for feasibility problem of FT-VMP is $O((l + |V_p|)^2 \cdot l \cdot |V_p|)$.

Given an instance of FT-VMP, after finding out that it is feasible using above maximum flow-based approach, we then use different algorithms designed in next section to accomplish various kinds of fault-tolerant VM placement.

IV. Algorithmic Solutions of FT-VMP

A. Linear Programming Solution for FT-VMP.

We present our linear programming formulation as follows. Besides y_i, which indicates if PM i is on or not, we introduce one more decision variable $x_{i,j,k}$. $x_{i,j,k} = 1$ if replica copy $v_{j,k}$ is placed at PM i; that is, $r(j,k) = i$. Otherwise, $x_{i,j,k} = 0$.

$$\min \sum_{i=1}^{|V_p|} y_i \tag{1}$$

s.t.

$$y_i = 0, 1 \quad \forall i \in V_p \tag{2}$$

$$x_{i,j,k} = 0, 1 \quad \forall i \in V_p, 1 \le j \le l, 1 \le k \le r_j \tag{3}$$

$$y_i = 1, \quad \forall i \in V_d \tag{4}$$

$$\sum_{i \in V_p} x_{i,j,k} = 1, \quad \forall 1 \le j \le l, 1 \le k \le r_j \tag{5}$$

$$y_i \ge x_{i,j,k}, \quad \forall i \in V_p - V_d, 1 \le j \le l, 1 \le k \le r_j \tag{6}$$

$$y_i * m_i^e \ge \sum_{j=1}^{l} \sum_{k=1}^{r_j} x_{i,j,k}, \quad \forall i \in V_p \tag{7}$$

$$\sum_{k=1}^{r_j} x_{i,j,k} \le 1, \quad \forall i \in V_p, 1 \le j \le l \tag{8}$$

$$x_{i,j,k} = 0, \quad \forall 1 \le j \le l, 1 \le k \le r_j, i \in V_p - \mathcal{C}(j) \tag{9}$$

The objective function 1 is to minimize the number of active PMs. Equations 2 and 3 are the integer constraints of

y_i and $x_{i,j,k}$, respectively. Equation 4 indicates that source PMs (PMs with at least one original VMs) must be turned on. Equation 5 guarantees that each VM has all its required number of replica copies placed. Inequality 6 indicates for any PM that is not a source PM, it must be turned on if at least one VM replica copy is placed into it. Inequality 7 enforces the resource capacity constraint of PMs. Inequality 8 enforces fault-tolerance constraint that different replica copies of the same VM cannot be placed onto the same PM. And finally, Equation 9 enforces the compatibility constraint for VMs, that is, v_j and all its replicas can not be placed into PMs that are in $V_p - \mathcal{C}(j)$.

B. Heuristic Algorithm for FT-VMP

In addition to the ILP-based algorithm, we propose two time-efficient heuristics. One is a greedy algorithm that always finds the first available PM to place replicas (Algo. 1). The other is a VM replica migration algorithm (Algo. 2) that improves an existing server consolidation algorithm (Algo. 1, [18]).

Greedy VM Repica Placement Algorithm. Algo. 1 works as follows. As all the source PMs must be turned on, they are initially the set of active PMs. We sort all the replicas in the non-descending order of the cardinalities of their compatibility sets, and place the ones with smallest compatibility first. Replicas with small compatibility sets have less number of compatible PMs to place upon, thus should be placed first before running out of options. We try to place each replica into the first available PM in the active PM set while satisfying fault-tolerance, compatibility, and resource capacity constraints. If not successful, we turn on another PM in this replica's compatibility set that is not an active PM, place this replica in it, and add it in the active PM set. It stops until all the replicas are placed in the data center. Sorting takes $l \cdot \log l$, placing replicas takes $l \cdot R$, $R = \max\{r_1, r_2, ..., r_l\}$. Thus the time complexity of this algorithm is $O(l \cdot (\log l + R))$.

Algorithm 1: Greedy VM Replica Placement Algorithm.
Input: An FT-VMP instance.
Output: The set of active PMs.

0. **Notations**:
   A: set of active PMs;
1. Sort $\mathcal{C}(j)$, $1 \le j \le l$, in the non-descending order of their cardinalities $|\mathcal{C}(j)|$;
2. WLOG, let $|\mathcal{C}(1)| \le |\mathcal{C}(2)| \le ... \le |\mathcal{C}(l)|$;
3. $A = \{s(j)\}$, $1 \le j \le l$;
4. **for** ($j = 1$ to l)
5. **for** ($k = 1$ to r_j)
6. Let $\mathcal{C}(j) \cap A = B$;
7. **if** ($B == \phi$) // B is an empty set
8. Let x be the first element in $\mathcal{C}(j)$;
9. $A = A \cup \{x\}$;
10. **else**
11. Let x be the first element in B;
12. **end if;**
13. Place $r_{j,k}$ at x;
14. **end for;**
15. **end for;**
16. **RETURN** A. /*Return the set active PMs */

VM Replica Migration Algorithm. In [18], we proposed an energy-efficient VM replication mechanism that considers power consumptions on PMs as well as on edges and switches inside the data center networks. To accomplish that, it first minimizes the network energy consumption of distributing the replica copies using minimum cost flow algorithm, and then designs a server consolidation algorithm (Algo. 1, [18]) to moves the VM replicas around to minimize active PMs. Our VM replica migration algorithm is based upon and improves this server consolidation algorithm. Below we first introduced two definitions in [18] and present their server consolidation algorithm. We then present in detail our VM replica migration algorithm.

Definition 1: (**Target Physical Machine (TPM) of a Replica VM $v_{j,k}$.**) A TPM of $v_{j,k}$ is a PM that replica VM $v_{j,k}$ can be possibly moved to while respective the minimum flow cost resulted from VM replica placement. For replica VM $v_{j,k}$, recall its source PM is $s(j)$ and its destination PM from minimum cost flow is $r(j,k)$. A PM i is a TPM of $v_{j,k}$ if a) $c_{S(j),r(j,k)} = c_{S(j),i}$, that is, it has the same replication cost to $s(j)$ as $r(j,k)$ does, b) it has enough storage to store $v_{j,k}$, and c) it does not store original VM v_j or any replica VM of v_j, that is, for any VM $v_{j',k'}$ PM i stores, $j' \neq j$. □

Definition 2: (**Consolidating Physical Machine (CPM).**) A physical machine is CPM if it is active and it is not a source PM. That is, CPMs are PMs that can be potentially turned off and made inactive. □

The server consolidation algorithm (Algo. 1, [18]) starts with the CPMs with only one replica, and checks if it can find a TPM. If so, it moves this replica to this TPM and turns this CPM off. Then it works on CPMs with two replicas and tries to move their replicas by finding TPMs. If both of them can be moved, it turns this TPM off. This continues with CPMs with more replicas until all the CPMs are being checked.

Below we make two observations about this algorithm and propose two critical improvements correspondingly. First, some replicas in some CPMs are moved regardless the rest of replicas cannot be moved or not from that CPMs. This not only does not turn off the considered CPM but also put some replicas on other PMs, making it more difficult to turn those PMs off in following steps. To address this, our improved server consolidation algorithm moves replicas in a CPM only when *all* the replicas can be moved out of this CPM; otherwise none of the replicas are moved. Second, when there are multiple TPMs that a replica can be moved to, current server consolidation algorithm randomly chooses any of them. Our improved algorithm instead will check if any of the TPMs is a source PM, and if so, move the replica to it. This is because the source PM cannot be turned off anyway, while moving replica to a non-source PM makes it more difficult to "empty" this PM. We referred to this improved algorithm VM Replica Migration Algorithm (Algo. 2), which is detailed

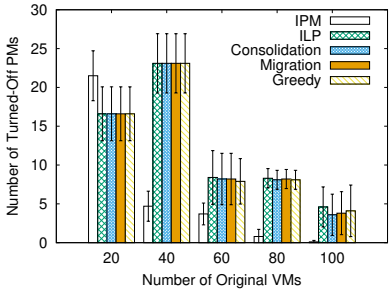

Fig. 4. Performance comparison by varying l, number of original VMs. Here, $k = 8$, $r_j = 10$, and $rc_i = 10$.

below.

Algorithm 2: VM Replica Migration Algorithm.
Input: VM replica placement from minimum cost flow.
Output: The set of active PMs.

0. **Notations**:
   A: set of active PMs;
   turn_off: true if a CPM can be turned off by having all its replicas moved to other PMs;
1. Set A as the set of CPMs after minimum cost flow VM replication in [18];
2. Let m be the largest number of replicas a PM in A has;
3. **for** ($i = 1$ to m)
4. Denote the set of CPMs with i replicas as $\{cpm_1, cpm_2, ..., cpm_{n_i}\}$, where $n_i \geq 0$;
5. **for** ($1 \leq j \leq n_i$) // Try to turn off cpm_j
6. turn_off = true;
7. Denote the replicas in cpm_j as $\{R_1, R_2, ...R_x\}$;
8. **for** ($1 \leq k \leq x$) //replica R_k
9. Find all of R_k's TPMs $\mathcal{T}_k$;
10. **if** ($\mathcal{T}_k == \phi$) //T_k is an empty set
11. turn_off=false;
12. break;
13. **end if;**
14. **end for;**
15. **if** (turn_off == true) // cmp_j can be turned off
16. **for** ($1 \leq k \leq x$) //Move R_k out of cmp_j
17. **if** (There is a source PM in $\mathcal{T}_k$)
18. Move R_k to this source PM;
19. **else** Move R_k to any PM in $\mathcal{T}_k$;
20. **end for;**
21. $A = A - \{cpm_j\}$; //cpm_j is now turned off
22. **end if;**
23. **end for;**
24. **end for;**
25. **RETURN** A. // Return the set of active PMs

Time Complexity. There are three *for* loops in Algo. 2. As $m = O(l)$, where l is the total number of VMs, $n_i = O(|V|^3/4)$, where $|V|^3/4$ is the total number of PMs, and $x = O(l)$, the total time complexity of Algo. 2 is $O(l^2 \cdot |V|^3)$.

V. Performance Evaluation

Simulation Setting. In this section we compare our designed algorithms with the existing work. We refer to our ILP-based algorithm as **ILP**, the greedy VM replica placement algorithm as **Greedy**, the VM replica migration algorithm as **Migration**. We also refer to the server consolidation algorithm in [18] as **Consolidation**. We implemented ILP using LP solver *lpsolve* [5] and wrote our own Java simulator for other algorithms on a Windows10 computer with Intel Core i7-6500U 2.50 GHz CPU and 32 GB RAM. We compare all the algorithms in terms of how many PMs they can turn off. We consider two different initial VM replica placements in the simulations. First, we compare them using the VM replica placement output from the minimum cost flow (MCF) VM replication in [18], which minimizes the energy consumption in the VM replication process. Second, we compare them using random placement of VM replicas. We also plot the inactive PMs (referred to as **IPMs**) resulted from each case. Note that all such IPMs are turned off before we run the four algorithms, as they do not have any replicas in them. We consider a small $k = 8$ data center with 128 PMs and a large $k = 16$ data center with 1024 PMs. Each data point in the plots is an average over ten runs. In all plots, the error bars indicate 95% confidence interval.

VM replica placement from MCF VM replication. In this part, we use the VM replica placement that is output by the MCF-based VM replication algorithm in [18]. Fig. 4 shows the comparison by increasing the number of original VMs in the cloud data center l from 20, 40, ..., to 100, while fixing the number of replicas of each VM r_j, $1 \leq j \leq l$, as 10, and the resource capacity of each PM rc_i, $1 \leq i \leq |V_p|$, as 10. As l increases, it clearly shows that number of IPMs decreases, as more VM replicas are placed inside the cloud data center thus less number of PMs can be turned off. We observe that when l is small (20 and 40), the performance of all the four algorithms (ILP, Consolidation, Migration, and Greedy) are similar, and each of them can further turn off 15-25 PMs. However, when increasing l, ILP always outperforms the other three by turning off at least one more PM, as it is an optimal solution. Also it is observed that with the increase of l, it gets more difficult to turn off PMs thus the number of turned-off PMs decreases for all the four algorithms. However, we do observe that when l increases from 20 to 40, the number of turned-off PMs increases instead. This is due to the fact that in this transition, the number of IPMs has a dramatical decrease, from 22 at $l = 20$ to 5 at $l = 40$. This leaves many PMs with sparse placement of VM replicas, making the task of VM placement easier at $l = 40$. When l gets to 100, it shows that all our designed algorithms viz. ILP, Migration, and Greedy outperform the Consolidation, demonstrating the effectiveness of our designed algorithms.

Fig. 5 compare all the algorithms by varying r_j, number of number of replicas of each VM, from 5, 10, 15, to 20. Although all the four algorithms perform very similarly, ILP still performs slightly better by being able to turning off one

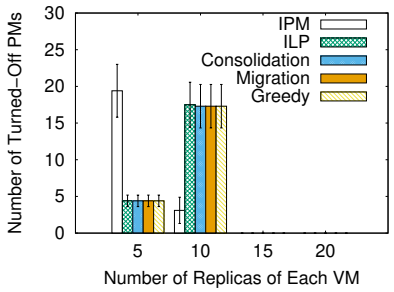

Fig. 5. Performance comparison by varying r_j, number of replica copies of each VM. Here, $k = 8$, $l = 50$, and $rc_i = 10$.

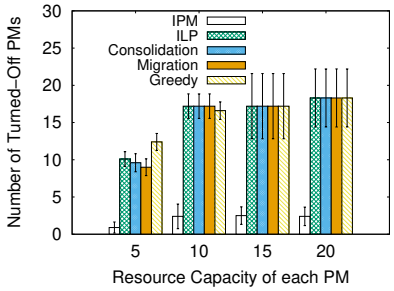

Fig. 6. Performance comparison by varying rc_i, storage capacity of each PM. Here, $k = 8$, $l = 50$, and $r_j = 10$.

or two more PMs, thus saving more energy than others. It also shows that when increasing r_j from 5 to 10, the number of PMs that can be turned off by all the algorithms increases from around 5 to around 17. This is rather counter-intuitive as it shows that the more VM replicas in the network, the more PMs being turned off. However, it can be explained as follows. When $r_j = 5$ there are around 10 IPMs, while when $r_j = 10$ there are around 3 IPMs. Although there are more replicas for $r_j = 10$, however, as only 3 IPMs are initially turned off, the replica placement on the rest active PMs becomes more sparser than that of $r_j = 5$. As the result, more PMs can be further turned off at $r_j = 10$. However, when we further increase r_j to 15 and 20, not only is there no IPMs that can be turned off, none of the four algorithms can further turn off any PMs due to the dense placement of VM replicas.

Fig. 6 compares all the algorithms by varying the resource capacity of PMs rc_i from 5, 10, 15, to 20. It shows that the number of IPMs from the MCF replica placement is less than five, increasing from 1 at $rc_i = 5$ to 3 at $rc_i = 10, 15$, and 20. As each PM increases its recourse capacity thus can hold more VM replicas, placement of VM replicas gets more sparse thus more PMs (i.e., IPMs) can be turned off. We observe that with the increase of the resource capacity of each PM, the number of turned-off PMs increases for all algorithms. They can further turn off between 8 and 18 more PMs, which is much larger than the number of IPMs initially turned off. This demonstrates that all the four algorithms are effective in moving VM replicas around to consolidate PMs in order to turn off more PMs. Finally, we observe that when resource capacity is 5, Greedy outperforms ILP by turning off more PMs. This is obviously not possible. A further investigation of the trace data shows that in this case, Greedy cannot place all the VM replicas while satisfying all the constraints. As such, the VM replica placement is more sparse in Greedy than that of ILP to achieve a better consolidation.

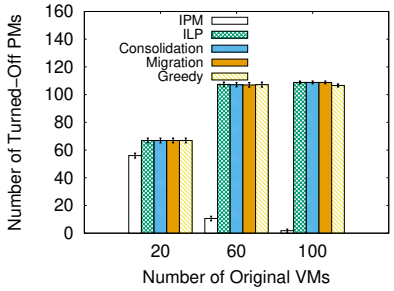

Fig. 7. Performance comparison in random placement by varying l, number of original VMs. Here, $k = 8$, $r_j = 10$, and $rc_i = 30$.

Random VM replica placement. Finally, we create a random initial VM replica placement that satisfies the fault-tolerance, compatibility, and resource capacity constraints, and execute these four algorithms on this placement. For compatibility constraint, we assume that each original VM and its replica copies cannot be placed to a randomly chosen specific PM due to software or platform incompatibility. Fig. 7 shows the performance comparison in a $k = 8$ data center. It shows that while increasing number of original VMs l the number of turned off PMs increases. As l increases from 20 to 60 and to 100, the number of IPMs of the random placement decreases from 60 to 10 to 2, thus there are more PMs that are initially on to be turned off by the algorithms. Compared to placement from MCF-based VM replication (Fig. 4), which can turn off less than 25 PMs, in random VM replica placement, our algorithms are able to turn off around 100 PMs. This demonstrates that that our algorithms work particularly effective for random VM replica placement.

Scalability studies. Lastly, we study the scalability of our algorithms for random initial placement using a large scale $k = 16$ data center of 1024 PMs. As ILP takes long time to compute, we only compare Greedy, Consolidation, and Migration. In Fig. 8 we set number of original VMs as 500, resource capacity of a PM as 10, and change the number of replicas of each VM r_j from 5, 10, 15, to 20. We observe that the number of IPMs and the number of PMs that can be turned off by all the three algorithms decrease with the increase of r_j. As more copies of VM replicas are placed, it gets more difficult to find an IPM or turn off any existing PMs by moving out their VM replicas to others. Nonetheless, all algorithms are able to turn off close to 700 PMs. Finally, Fig. 9 shows that when increasing the resource capacity of each PM, all three algorithms are able to turn off around 700 PMs, again, due to the effectiveness of our algorithms on random VM placement. These demonstrate the effectiveness of our algorithms in turning off PMs thus saving energy in

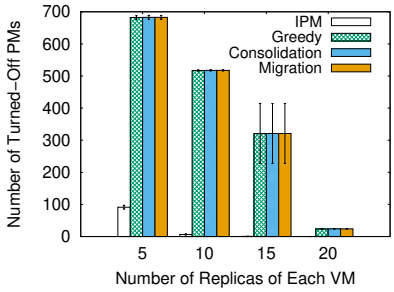

Fig. 8. Scalability study by varying r_j, number of replica copies of each VM. Here, $k = 16$, $l = 500$, and $rc_i = 10$.

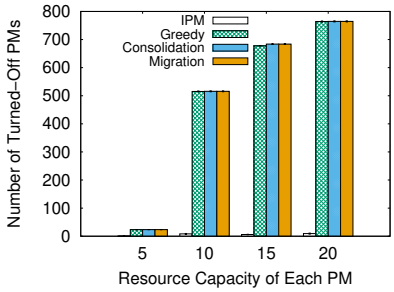

Fig. 9. Scalability study by varying rc_i, storage capacity of each PM. Here, $k = 16$, $l = 500$, and $r_j = 10$.

large scale data centers.

VI. Conclusion and Future Work

We propose a new fault-tolerant VM placement problem called FT-VMP. Given that different VMs have different fault-tolerance requirement and compatibility requirement, FT-VMP studies how to place VM replica copies inside cloud data centers in order to minimize the number of PMs that stores VM replicas under the constraint that PMs have limited storage capacity. We give this problem a thorough formal treatment by first proving that it is NP-hard, and then designing an ILP-based algorithm to solve it optimally. We also design a suite of time-efficient and scalable heuristic algorithms. Via extensive simulations, we show that ILP-based algorithm outperforms the state-of-the-art VM replica placement in a wide range of network dynamics. Our algorithms are not only fault-tolerant but also energy-efficient, as they manage to turn off PMs to save energy while placing multiple replica copies of the same VM. As a future and ongoing work, we will consider the energy consumption inside network and take into account the energy cost of migrating VMs from one PM to another. We will design an integral approach that minimizes the sum of energy consumptions in both network and PMs via fault-tolerant VM placement and migration. Our initial finding is that as minimizing network energy consumption often is in conflict with minimizing PM energy consumption, there is a need for multi-objective optimization techniques with the solution form of Pareto optimal front.

VII. Acknowledgement

This work was supported in part by NSF Grant CNS-1911191.

References

[1] Amazon s3 replication. https://aws.amazon.com/s3/features/replication/.
[2] Azure storage redundancy. https://docs.microsoft.com/en-us/azure/storage/common/storage-redundancy.
[3] Citrix hypervisor - server virtualization and consolidation. https://www.citrix.com/products/citrix-hypervisor/.
[4] Introduction to hyper-v on windows 10. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/.
[5] Linear programming solver lp_solve. https://sourceforge.net/projects/lpsolve/.
[6] Vmware esxi: The purpose-built bare metal hypervisor. https://www.vmware.com/products/esxi-and-esx.html.
[7] Vmware esxi: The purpose-built bare metal hypervisor. Kernel-based Virtual Machine (KVM).
[8] vsphere: A server virtualization platform from vmware. https://www.vmware.com/products/vsphere.html.
[9] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, 2008.
[10] E.A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 1(1):1277–1280, 1970.
[11] J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM*, 19:248–264, 1972.
[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman Co., USA, 1979.
[13] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum flow problem. In *In Proc. of the 18th ACM Symposium on the Theory of Computing*, 1986.
[14] H. Goudarzi and M. Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. In *Proc. of IEEE Cloud*, 2012.
[15] R. Gupta, S. K. Bose, S. Sundarrajan, M. Chebiyam, and A. Chakrabarti. A two stage heuristic algorithm for solving the server consolidation problem with item-item and bin-item incompatibility constraints. In *2008 IEEE International Conference on Services Computing*, volume 2, pages 39–46, 2008.
[16] L.R. Ford Jr. and D.R. Fulkerson. Maximum flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
[17] A.V. Karzanov. Determining the maximal flow in a network by the method of preflows. *Soviet Mathematics Dokladly*, 15:434–437, 1974.
[18] P. Khani, B. Tang, J. Han, and M. Beheshti. Power-efficient virtual machine replication in data centers. In *Proc. of the ICC 2016*.
[19] J. Luo, W. Song, and L. Yin. Reliable virtual machine placement based on multi-objective optimization with traffic-aware algorithm in industrial cloud. *IEEE Access*, 6:23043–23052, 2018.
[20] F. Machida, M. Kawato, and Y. Maeno. Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *Proc. of the 12th IEEE/IFIP NOMS, mini conference*, 2010.
[21] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J. M. Pierson, and A. V. Vasilakos. Cloud computing: Survey on energy efficiency. *ACM Comput. Surv.*, 47(2), 2014.
[22] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue. Survivable virtual infrastructure mapping in virtualized data centers. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 196–203, 2012.
[23] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King. Component ranking for fault-tolerant cloud applications. *IEEE Transactions on Services Computing*, 5(4):540–550, 2012.
[24] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, and R. Buyya. Cloud service reliability enhancement via virtual machine placement optimization. *IEEE Transactions on Services Computing*, 10(6):902–913, 2017.
[25] A. Zhou, S. Wang, C. Hsu, M. H. Kim, and K.S. Wong. Virtual machine placement with (m, n)-fault tolerance in cloud data center. *Cluster Computing*, 22:11619–11631, 2019.
[26] A. Zhou, S. Wang, Z. Zheng, C. Hsu, M. R. Lyu, and F. Yang. On cloud service reliability enhancement with optimal resource usage. *IEEE Transactions on Cloud Computing*, 4(4):452–466, 2016.