LVRM: On the Design of Efficient Link Based Virtual Resource Management Algorithm for Cloud Platforms

Prasan Kumar Sahoo, *Senior Member, IEEE*,
Chinmaya Kumar Dehury, and Bharadwaj Veeravalli, *Senior Member, IEEE*

Abstract—Virtualization technology boosts up traditional computing concept to cloud computing by introducing Virtual Machines (VMs) over the Physical Machines (PMs), which enables the cloud service providers to share the limited computing and network resources among multiple users. Virtual resource mapping can be defined as the process of embedding multiple VMs and their network resource demand onto multiple inter-connected PMs. The existing mechanisms of resource mapping need to be efficient enough to minimize the number of PMs without compromising the deadline of the tasks assigned to the VMs, which is NP-hard. To deal with this problem, a Link based Virtual Resource Management (LVRM) algorithm is designed to map the VMs onto PMs based on the available and required resources of the PMs and VMs, respectively. The designed algorithm exploits the fact that the demanded network bandwidth among VMs should be given higher priority while allocating the physical resources to the inter-connected virtual machines as insufficient network bandwidth may detain the task execution. The proposed algorithm is evaluated by a discrete event simulator and is compared with similar virtual network embedded algorithms. Simulation results show that LVRM can outperform over other network embedded algorithms.

Index Terms—Resource mapping, VM placement, graph theory

1 Introduction

The new paradigm of traditional computing convention of stand-alone personal computer has been extended to cloud computing over past few years. This facilitates the tenants to avail the computing infrastructure as per requirement and enables the pay-as-you-go feature of the cloud computing model. In other words, the available pools of computing resources in a data center are given to the tenants as public utility services [1]. For this, different service models are introduced to suit the tenants' requirement such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) by using Virtualization technology.

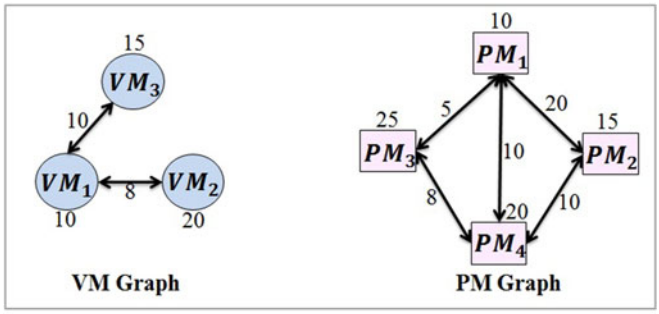
Unlike conventional stand-alone personal computers, where single operating system can run atop single physical machine (PM), with virtualization features users can run multiple instances of different operating systems (OS) inside a virtual machine (VM) at single point of time in cloud computing [2]. Therefore, the incoming user programs always can run in a VM. Users, also known as cloud consumers, may request for different sets of resources, which typically include CPU, memory, storage, and network bandwidth [3].

On the other hand, from the service provider's point of view, a data center consists of hundreds or thousands interconnected physical resources such as servers and switches. For efficient utilization of resources, virtualization is the emerging technology that helps the service provider for dynamic provisioning, configuration, and reconfiguration of physical resources [4]. On the other hand, this virtualization technique also allows the users to change their demands over time by reconfiguring the VMs.

The complexity of the physical resources management and their utilization increases exponentially with the increase in number of interconnected physical servers and number of end-users as the end-users submit their resource demand in terms of multiple interconnected VMs. When the number of end-users increases, multiple VMs are needed to be created on single or multiple PMs to fulfill the user's demand. The responsibility of CSP is also to establish the physical path between corresponding PMs in order to enable the VMs to communicate. The problem of allocating the virtual resource to the available physical computing resource and the networking resource is referred to virtual resource allocation problem, which is our main focus in this paper.

- *P.K. Sahoo is with the Department of Computer Science and Information Engineering, Chang Gung University, Guishan 333, Taiwan and with the Department of Cardiology, Chang Gung Memorial Hospital, Linkou 33305, Taiwan. E-mail: pksahoo@mail.cgu.edu.tw.*
- *C.K. Dehury is with the Department of Computer Science and Information Engineering, Chang Gung University, Guishan 333, Taiwan. E-mail: D0321009@stmail.cgu.edu.tw.*
- *B. Veeravalli is with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077. E-mail: elebv@nus.edu.sg.*

Manuscript received 23 Apr. 2017; revised 26 Sept. 2017; accepted 13 Nov. 2017. Date of publication 7 Dec. 2017; date of current version 9 Mar. 2018. (Corresponding author: Chinmaya Kumar Dehury.)
Recommended for acceptance by X. Li.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2017.2780844



Solution 1: Based on available computing Resources

VM	PM
1	1
2	4
3	3

Solution 2: Based on available network bandwidth

Virtual edge	Physical edge
(1,3)	(4,2)
(1,2)	(4,3)

Fig. 1. An example of different mapping solutions.

In general, the major challenge in the process of allocation of physical resources to the VMs is to minimize the number of PMs for all VMs. In data intensive distributed applications the traffic flow among the VMs is very high, as huge amount of data are transferred among multiple servers. For such network driven applications network resources are given higher priority over the server resources as insufficient network bandwidth may delay the data transfer and therefore the dependent tasks cannot be executed. Moreover, minimizing the number of PMs can lower down the power consumption. On the contrary, this may create problems, such as higher job execution time and SLAs violation due to lack of sufficient resources for VMs.

The incoming jobs from the users normally consist of smaller number of independent or dependent tasks. The dependent or interconnected tasks are represented in the form of graphs and are called as task trees or task graphs. The produced task graph, also known as the task dependency graph, is further analyzed by employing the basic principles to determine the required number of VMs and the communication characteristics among them. After derivation of the VMs, they are configured in such a way that the assignment of tasks onto VMs will not suffer from any delay caused by insufficient virtual resources.

Considering the IaaS model provided by the cloud service provider, available shared interconnected physical servers, and their network topology, we will be focusing on the problem of virtual resource allocation, where a set of interconnected VMs are mapped onto a set of interconnected PMs. Specifically, we are studying an efficient placement of requested virtual resources onto physical resources by minimizing the number of PMs.

1.1 Motivation

The main assets of the cloud computing are the huge amount of physical resources such as memory, storage, and computing capacity. Further, the major challenges for cloud service provider are to manage those huge resources and allocate them to cloud consumers. While allocating the physical resources, the service provider needs to consider different factors such as geographical distance of the PMs, capacity of the PMs etc. Inefficient allocation leads to wastage of resources, more power consumption, less revenue, and low QoS. Based on different research reports discussed in Section 2, computing resources such as CPU, memory, and storage are given higher priority as compared to the network resources in most of the allocation strategies. It is also observed that the resource allocation strategies vary from the applications points of view. For example, as shown in Fig. 1, let us assume that a user sends the request for creating three virtual machines as VM_1, VM_2, and VM_3 with their resource requirement of 10, 20, and 15 units, respectively. Let, 8 units of network bandwidth be required for the information exchange between VM_1 and VM_2 and 10 units of network bandwidth be required between the VM_1 and VM_3. Let, 4 PMs be available with the cloud service provider, PM_1 through PM_4 with limited computing resources and network bandwidth among the PMs.

The available resources in those four PMs can be allocated to three requested virtual machines in different ways such as solution 1 and solution 2 as depicted in Fig. 1. The requested computing resources are given maximum priority and are mapped followed by the mapping of virtual links in solution 1 and therefore the available resources in PM_1, PM_4, and PM_3 are allocated to the virtual machines VM_1, VM_2, and VM_3, respectively. However, the demanded network bandwidth among VMs are given higher priority and are mapped followed by the mapping of the requested computing resource in solution 2 and as a result, virtual machine VM_1, VM_2, and VM_3 are created in physical machines PM_4, PM_3, and PM_2, respectively. By comparing both solutions, the major problem in Solution 1 is that due to the insufficient network bandwidth between PM_1 and PM_3 may affect the performance of the jobs running in virtual machines VM_1 and VM_3 though the demanded computing resources are prioritized and fulfilled by the physical machines. In such scenario, the main problem occurs when the traffic flow within the application is very high. Generally, the VMs require huge amount of data from different data servers in data intensive applications. Transferring massive data among servers require very high network bandwidth. Further, some distributed applications that require huge number of computing nodes also require very high network bandwidth.

For better understanding, let us consider an example of gaming application that runs in the cloud environment. Let each VM be responsible for hosting the single user. In this scenario, multiple VMs hosting multiple users interact with each other with higher network bandwidth demand. While placing the VMs onto multiple physical servers, it is highly essential to ensure that the network bandwidth demand of each VM is fulfilled. The network demand of VMs can be fulfilled by mapping the virtual links in two different ways. First, the virtual link is mapped to the single substrate link and second, the virtual link is mapped to a set of substrate links forming a substrate path, where multiple nodes are involved. In the latter solution, the network latency between two VMs at both ends of the path is high as multiple nodes are involved in a substrate path. Higher network latency affects the execution time even when the requirement on computing resources is fulfilled [5]. In case of an application that demands both network and computing resources, the

execution time will be adversely affected even if one of the resource demands is not met. In contrast, inefficient mapping of the virtual links will affect the execution time even if enough computation resources are provided. This motivates us to think a novel methodology to allocate the physical resources to the virtual machines in such a way that the allocation result is efficient for the both cloud provider and consumer in terms of better utilization of physical resources and QoS, respectively.

1.2 Goals

Based on our aforementioned motivation, the main goals of our work can be summarized as follows.

- Design an efficient link based virtual resource management algorithm to maximize the resource utilization by considering different network and physical resource related parameters.
- Minimize the number of physical machines involved that leads to the minimization of the servers' power consumption and complete processing of the jobs on or before the deadline.
- Minimize the total hopping cost, which indicates that for each virtual edge the corresponding mapped PMs have direct link.

Rest of the paper is organized as follows. Brief summary of some related works on virtual resource mapping and motivations behind our proposed work are presented in Section 2. Graph representation of tasks and description about the required VMs and available PMs are presented in Section 3. The virtual resource mapping problem is formulated in Section 4. Our proposed link based virtual resource management algorithm is described in Section 5. Performance evaluation of the proposed algorithm is done in Section 6 and concluding remarks are made in Section 7.

2 Related Works

In the last few years, substantial attention is given towards the development of several VM placement algorithms considering the cloud service providers' objectives and different parameters such as available bandwidth, network latency, geographical distribution, cost of energy consumption, requested, and available computing resources [4], [6]. The objectives of the cloud service providers include to provide quality of service, generating more revenue from limited physical resources, satisfying the service level agreements etc. Few researchers have considered the VM placement problem as a Virtual Network Embedding (VNE) problem and have formulated the VNE problem as Mixed Integer Programming (MIP) problem [4], [7]. Authors in [4] have considered the hard and soft QoS while allocating the requested computation resources. Their objectives are to minimize the total amount of different types of allocated physical resources and to minimize the overall number of hops for a virtual link.

Authors in [8], [9], address the VNE problem as multi-objective linear programming problem in virtualized cloud data centers with the goal to maximize the revenue and minimize the embedding cost. However, mapping of node followed by the virtual links may also lead to the physical resource fragmentation and may not be able to minimize the embedding cost. Considering the Fat-Tree data center topology, authors in [10] propose multiple virtual network (VN) placement algorithms, which can guarantee the demand of the incoming tasks. However, it is not clear how the virtual network consisting of virtual machines and virtual switches is derived. Furthermore, mapping of VMs before the virtual links may suffer from the fact that the required network bandwidth may not be guaranteed by the proposed mechanisms. Similarly, in the proposed VNE algorithm in [11], it is not clear as to which resource type should be mapped first and therefore it may yield inefficient result.

Considering inter-VM communication, authors in [12] have proposed the VM placement techniques in modular data center environment. Though multiple VMs are mapped onto single PM to reduce the communication cost, the proposed model may turn into time-consuming mapping algorithm due to dynamic workload of the VMs. Authors in [13], [14] propose a novel mechanism to embed the requested virtual network onto the physical network. The virtual network requests are selected periodically through an auction mechanism. Such model may not be able to maintain the fairness among the users. The model in [13] does not monitor the round trip time of physical links as a result of which higher round trip time may lead to low data transfer rate among the corresponding tasks.

With the goal to maximize the acceptance ratio, authors in [5] have considered the federated cloud environment for mapping multiple VM clusters. The geographical distance among the cloud providers increases the data transfer time among the mapped VMs and therefore the execution time increases. Authors in [15] have extended the VNE problem by assigning priority class to each virtual link in a request. However, it cannot minimize the total amount of required network resource as it does not map multiple VMs from one request onto single physical machine. Authors in [16] propose a new path algebraic strategy to map both virtual nodes and virtual links at the same time with the goal to optimize the revenue and acceptance ratio. However, the proposed mechanism consumes more time due to the link mapping strategy. The time slot based resource allocation mechanism presented in [7] may not produce efficient mapping solution, where a large number of VMs are mapped to a single PM.

Different VM scheduling algorithms that are proposed in [17], [18] minimizes the server energy cost by minimizing the number of active physical servers and network elements. However, the algorithm in [17] avoids the opportunity to embed multiple virtual machines those belong to single virtual network requests onto a single physical machine. Authors in [18] address VM assignment problem with the objective to minimize the power consumption without considering the power consumption of physical edge. Similarly, the resource allocation model proposed in [19] avoids the effect of inter-VM communication, which increases the data transfer delay.

Considering cloud gaming application, the model in [6] gives more priority to the network latency of the servers resulting imbalance workload among physical servers. Similarly, considering multi-tenant IaaS clouds, the resource sharing mechanism presented in [20] provides a set of VMs to a group of buyer based on the priority. This does not consider available network bandwidth as an important parameter to execute the dependent tasks. Authors in [21] propose

bandwidth sharing and pricing policies that can provide incentives to users and revenue to the CSP by assigning different pricing values to the congested and non-congested links. Authors in [22] propose heuristic virtual resource mapping algorithm to evaluate and detect VM threats and vulnerability. However, the proposed mechanisms do not exploit the opportunity to allocate or map multiple VMs onto single PM in order to produce efficient result in terms of server utilization and acceptance rate.

A number of approaches related to virtual resource management have been proposed considering multiple factors such as power consumption, revenue optimization, and application specific resource allocation. Though the studies in the current literature achieve their respective goals, to the best of our knowledge none of the studies propose a resource management mechanism, which can minimize the required amount of physical resources and minimize the total hopping cost by taking the advantages of mapping multiple VMs onto single PM.

3 Graphical Representation

In this section, we adopt the graph theory to represent the physical resources as logical graphs. We have considered the scenario where jobs arrive from the cloud consumer and are analyzed by a third party analyzer to produce the corresponding task graphs and eventually the corresponding VM-graphs. While generating the VM-graph, the number of VMs should be minimized. It is assumed that the VMs are heterogeneous and the available physical servers are configured with non-uniform amount of resources, i.e., heterogeneous servers. It is also assumed that the common oversubscription of the cloud resource is not allowed as this may lead to dynamic PM overload, which requires a dedicated mechanism to handle. The available physical servers and communication between them are represented as PM-graphs. The detailed description of the task graph, corresponding VM-graph, and the available PM-graph are given below.

3.1 Task Graph

It is assumed that the incoming jobs consist of a large number of small size tasks. Based on the communication dependencies among those tasks and their resource requirements, the incoming jobs are analyzed to produce the task graph. Complex scientific and multimedia applications are represented by directed acyclic graph, where a vertex or node represents a single task and an edge between two vertices represent the communication between them [23]. An edge from vertex i to vertex j represents the dependency of vertex j on vertex i. In other words, the task represented by the vertex j can start only after the execution of the task represented by the vertex i [24]. Hence, a task graph is also called the directed acyclic graph (DAG) and henceforth the term DAG is used instead of task graph. It is to be noted that the tasks are of block types, which indicate that a task should receive all inputs in order to start its execution. The stream type tasks are beyond the scope of this paper.

3.2 VM-Graph

Virtual Machine graph (VM-graph) is derived from the task graph or DAG. The VM-graph indicates the number of virtual machines that are required to finish the job before the deadline. The number of virtual machines needed is decided by the degree of parallelism in DAG. Along with this, VM-graph also reveals the communication between tasks/vertices in DAG. A vertex in VM-graph represents a VM and an edge between two vertices represents the data flow between two virtual machines. The derived VM-graph acts as a bridge between the actually submitted jobs and physical servers.

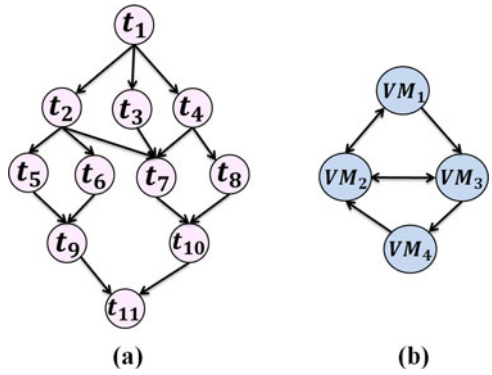


Fig. 2. An example of (a) task graph and (b) required VMs and their communication.

3.3 Derivation of VM-Graph from Task Graph

The task graph is associated with multiple parameters such as the computation and network resource requirements. Each task may also come with specific hard or soft deadline. Hence, we propose here how to derive the VM-graph from the Task-graph. In order to illustrate the derivation, an example of the task graph consisting of eleven tasks is presented in Fig. 2a. The task graph is presented in the form of directed acyclic graph. As discussed in Section 3.1, the tasks are assumed to be block types, which infers that a task can start its execution only after receiving all the required inputs from other tasks. In other words, the child tasks cannot start their execution before completion of the execution of the parent tasks. For simplicity, it is assumed that VMs are configured with enough amount of resources to fulfill the demand of the tasks and can execute exactly one task at a time. Therefore, eleven VMs are required to execute eleven tasks given in Fig. 2.

Furthermore, the number of VMs can be minimized by taking the degree of parallelism of the tasks into consideration. Considering the block type tasks and degree of the parallelism, the number of VMs can be calculated as the maximum number of tasks running concurrently at any particular instant of time. As shown in Fig. 2a, maximum four tasks, i.e., tasks t_5, t_6, t_7, and t_8 run concurrently at any particular instant of time. Accordingly, four VMs are required to execute all eleven tasks as shown in Fig. 2b.

In order to derive the VM graph from the Task graph, tasks in each level of the DAG are assigned to the VMs randomly. For example, tasks t_1 at level 1 can be assigned to VM_1. Tasks t_2, t_3, t_4 at level 2 can be assigned to VM_1, VM_2, and VM_3, respectively. Tasks t_5, t_6, t_7, and t_8 at level 3 can be assigned to VM_1, VM_2, VM_3, and VM_4, respectively. Similarly, tasks t_9 and t_{10} at level 4 can be assigned to VM_1 and VM_3, respectively and the task t_{11} at level 5 can be assigned to VM_1. Furthermore, for each directed edge in the task graph, there must exist a directed edge between the corresponding VMs of the VM-graph. For example, for the

TABLE 1
List of Notations

Notation	Description
$G^P(N^P, E^P)$	Graph of Physical Machines; PM-graph
$G^V(N^V, E^V)$	Graph of Virtual Machines; VM-graph
$E(UV)$	Physical link $UV \in E^P$
$E'(uv)$	Virtual link $uv \in E^V$
$W_t(UV)$	Weight of physical link UV at time t
$L_t(UV)$	Load on physical link $E(UV)$ at time t
$BW_{remain}(UV)$	Remaining network bandwidth on link $E(UV)$
$BW_{req}(uv)$	Required network bandwidth by link $E'(uv)$
$\sigma_t(UV)$	Trust value of link $E(UV)$ at time t
$\beta(UV)$	Current round trip time over PM U and V
$\omega_t(UV)$	Energy consumption by link $E(UV)$
Π_{uv}^{UV}	Path from PM U to V for virtual edge $E'(uv)$
$HC(\Pi_{uv}^{UV})$	Hopping cost of the path Π_{uv}^{UV}
$\alpha_x^P(U)$	Resource of type x available at PM U
$\alpha_x^V(u)$	Resource of type x demanded by VM u
x	Resource type, such as memory, CPU capacity, Storage capacity etc

directed edge from task t_4 to t_8, there exists a corresponding directed edge from VM_3 to VM_4 as shown in Fig. 2b. For the directed edge from task t_1 to t_3, a directed edge from VM_1 to VM_2 is drawn in the VM graph. A corresponding directed edge is also drawn from VM_2 to VM_1 as there exists a directed edge from task t_6 to t_9. In the similar fashion, rest of the directed edges of the task graph can be transformed to the corresponding directed edges of VM pairs in the VM graph. In this example, we assume that one VM can execute only one task at any instant of time.

3.4 PM-Graph

Physical Machine graph (PM-graph) is the representation of all physical machines and their communications, where a vertex represents any physical machine and an edge represents the communication between any two PMs. In addition to this, other different characteristics are associated with the edges and the nodes in PM-graph such as weight of an edge and nodes. The calculation of weight of nodes and edges is described in Section 4. Both VM-graph and PM-graphs are input to the proposed algorithm and the output is a set of PMs to host the VM(s).

4 Problem Formulation

As mentioned in Section 3, the available physical machines are represented as undirected weighted graph $G^P = \left(N^P, E^P\right)$, where N^P represents the set of available physical machines and E^P represents the set of physical communication links. Likewise, $G^V = \left(N^V, E^V\right)$ be the directed weighted graph representing the requested virtual machines or the VM-graph. N^V represents the set of virtual machines and E^V represents the set of virtual communication links among virtual machines. $\alpha_x^P(U)$ be the available resources of type x associated with physical machine U, $U \in N_P$. $\alpha_x^V(u)$ be the amount of resources of type x demanded by the virtual machine u, $u \in N_V$. The resource types typically include memory, CPU, and storage capacity. $BW_{remain}(UV)$ and $BW_{max}(UV)$ be the remaining bandwidth and maximum available bandwidth on a physical link (UV), respectively. Similarly, in VM-graph, $BW_{req}(uv)$ is the amount of required bandwidth for a virtual link (uv). List of notations with their description are presented in Table 1.

4.1 Weight of Physical Links

For better placement of virtual links over physical links, we have emphasized the calculation of weight of the physical links. To calculate the weight of a physical link, remaining and maximum bandwidth of that link, average load over a period of time, RTT, geographical distance of physical machines, energy consumption and number of hops are taken into consideration. Though, the current RTT value depends on the current load, we have also considered the average load of the physical link over a period of time. Weight of a physical link increases when the load, current RTT value, and energy consumption decreases. We believe that for better performance and to meet the deadline of users' multi-task jobs, higher preference should be given to the network resources over computing resources. In executing multi-task jobs, where multiple virtual machines are involved, network resource has direct impact on computing performance as any delay due to data transfer in between two virtual machines results in increasing the computation delay. Hence, prioritizing the network resources over computing resources is recommended.

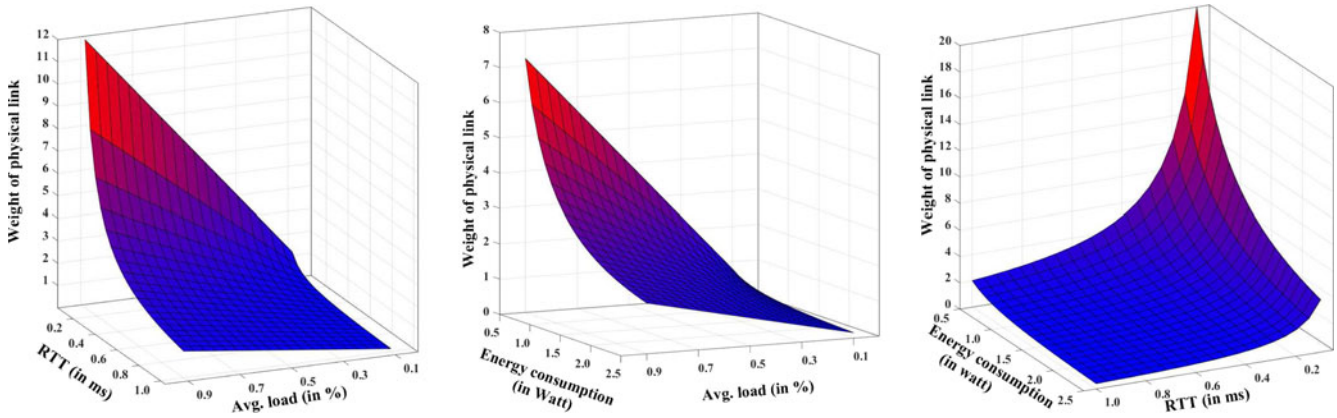
In the proposed resource allocation mechanism, maximum bandwidth and remaining bandwidth are used to calculate the load on the physical link. The value of load of a physical link ranges from 0 to 1 and can be calculated as follows:

$$L_T(UV) = 1 - \frac{BW_{remain}(UV)}{BW_{max}(UV)}, \tag{1}$$

where, $L_T(UV)$ is the load on the physical link UV at current time T. It is assumed that maximum bandwidth BW_{max} and remaining bandwidth BW_{remain} available on each physical link is provided by the cloud service provider (CSP). Similarly, from load $L_T(UV)$, mean load $\overline{L}_T(UV)$ on a link is calculated by taking the average of load from beginning to the current time T in a specified time interval. The time interval is defined as the time duration between any two consecutive time instances for calculating the load of a physical link. For example, let the load of a physical link (UV) at time $t = 0$ be 0.75, at time $t = 1$ be 0.25 and at current time $t = T = 2$ be 0.65. The average load can be calculated as $\overline{L}_T(UV) = \frac{0.75+0.25+0.65}{2+1} = 0.55$. In general, considering the loads that are calculated at $T + 1$ number of time instances, the average load can be calculated as

$$\overline{L}_T(UV) = \sum_{t=0}^{T} \frac{L_t(UV)}{T+1}. \tag{2}$$

To find a better choice of physical link for a virtual link, the round trip time or the latency plays an important role. Round Trip Time (RTT) represents the time taken to travel a data packet from one physical machine to another. Hence, the physical link with lesser RTT can be considered as the most suitable physical link in the process of mapping. Different factors such as geographical distance, number of network elements affect the RTT of the physical link. Let $\beta(UV)$ be the current RTT in between the physical machine U and V. The current RTT and load of the physical link are used to



(a) Load and RTT. (b) Load and energy consumption (c) RTT and energy consumption.

Fig. 3. Effect of load, round trip time (RTT), and energy consumption on weight of physical link.

calculate the Trust Value (σ) of the physical link at time t, which can be represented as

$$\sigma_t(UV) = \frac{\overline{L}_t(UV)}{\beta(UV)} \times \frac{1}{\Delta} \quad ; \qquad 0 < \Delta < 1, \tag{3}$$

where, Δ is a tuning parameter to reflect the effect of minor change in current latency on trust value of the corresponding physical edge. With lower value of Δ, slight change in $\overline{L}_t(UV)$ and $\beta(UV)$ will have greater impact on the trust value $\sigma_t(UV)$. Such greater impact on the trust value $\sigma_t(UV)$ of a physical link cannot be realized by removing the constant Δ from Equation (3), which may affect further in calculating the weight of a physical link. Hence, the aforementioned calculated trust value and energy consumption are combined and are calculated as the weight of the physical link ($W_t(UV)$), which can be represented as follows:

$$W_t(UV) = \frac{\sigma_t(UV)}{\omega_t(UV)}, \tag{4}$$

where, $\omega_t(UV)$ indicates the energy consumption by the physical link UV at time t. Equations (2), (3), and (4) are formulated to establish the relationship among multiple parameters such as current RTT, load on the physical links and energy consumed by the link. Physical link with *large* trust value and *short* geographical distance is *more* preferable for a virtual link. This is helpful when multiple physical links are eligible to host a virtual link based on bandwidth. Physical link with maximum weight is selected to host the virtual link. Fig. 3 illustrates the relationship among aforementioned parameters. Here, the load can be defined as the percentage of the network resource allocated to different users. The average load ranges between 10 to 90 percent, whereas the current RTT ranges between 0.1 to 1 ms. For demonstration purpose, the value of power consumption is kept between 0.1 to 2.5 watt as given in [25]. In Fig. 3a, it can be observed that the physical links with higher average workload and less RTT have maximum weight. Similarly, the physical links with maximum average workload and least power consumption have maximum weight value with constant RTT, as shown in Fig. 3b. Hence, weight of the physical link is inversely proportional to the RTT and energy consumption as shown in Fig. 3c. The value of Δ is kept constant at 0.5 for calculating the above weights. Since, weight of a physical link is inversely proportional to the RTT and energy consumption and is directly proportional to the bandwidth, we give priority to the links with higher weights. As a result, embedding the virtual links onto the physical links with smaller RTT value and higher bandwidth ensures better performance as compared to the physical link with larger RTT value and lower bandwidth.

4.2 Objective Function

In the proposed resource mapping mechanism, it is assumed that multiple virtual machines from a single VM request can be hosted by the single physical machine. To achieve this, a logical edge is introduced, where the physical machines at both the ends are same. For example, the edge from the physical machine U to U is considered to be a logical edge. The weight of the logical edge is more than any of the physical edge. For this, we assign ∞ to the weight and BW_{remain} of a logical edge. All physical machines that are part of any logical edge are also included in the PM-graph.

Let, η_v^p and κ be the binary and decision variables, respectively and are expressed as follows:

$$\eta_v^p = \begin{cases} 1 & \text{if VM } v \text{ is assigned to PM } p \\ 0 & \text{Otherwise} \end{cases} \tag{5}$$

$$\kappa(U) = \begin{cases} 1 & \text{if PM } U \text{ is hosting at least one VM from set } N^V \\ 0 & \text{Otherwise.} \end{cases} \tag{6}$$

Π_{uv}^{UV} indicates the path from PM U to PM V for virtual edge (uv). All data communication between VM u and v must follow this path. Π_{uv}^{UV} is used to calculate the hopping cost $HC(\Pi_{uv}^{UV})$ for a virtual edge. Hopping cost is defined as the number of network elements present in a physical path including the sink physical machine. The minimum value of HC for the virtual edges is 1, when both end VMs of a virtual edge are mapped onto different PMs. However, when both end VMs of a virtual edge are mapped onto single PM, the hopping cost is calculated as 0. Our goal is to minimize the number of active physical servers and total hopping cost. With the notations defined above, the virtual resource mapping problem can be formulated as follows:

Objective

$$\min \sum_{\forall p \in N^P} \kappa(p) + \sum_{\forall (uv) \in E^V, \forall (UV) \in E^P} HC(\Pi_{uv}^{UV}).$$

Constraints

$$\eta_{v_i}^{p_j} \in \{0,1\}, \forall v_i \in N^V, \forall p_j \in N^P \tag{7}$$

$$\sum_{j=1}^{|N^P|} \eta_{v_i}^{p_j} = 1, 1 \leq i \leq |N^V| \tag{8}$$

$$p_j \in N^p, \nexists v_i \in N^V, \alpha_x^{p_j} \geq \alpha_x^{v_i}, \forall x \tag{9}$$

$$\sum_{\forall v \in N^V} \alpha_x^v \leq \sum_{\forall p \in N^P} \alpha_x^p, \quad \forall x \tag{10}$$

$$\sum_{\forall (uv) \in E^V} BW_{req}(uv) \leq \sum_{\forall (UV) \in E^P} BW_{remain}(UV) \tag{11}$$

$$\max_{\forall v \in N^V} \alpha_x^v \leq \max_{\forall p \in N^P} \alpha_x^p, \quad \forall x \tag{12}$$

$$\max_{\forall (uv) \in E^V} BW_{req}(uv) \leq \max_{\forall (UV) \in E^P} BW_{remain}(UV) \tag{13}$$

$$N^P \neq \emptyset, E^P \neq \emptyset, N^V \neq \emptyset, E^V \neq \emptyset. \tag{14}$$

1) The objective function is two fold:
   a) *To minimize the number of physical machines involved*. $\kappa(p)$ represents the binary value indicating if the physical machine p, $p \in N^P$ is hosting any VM v from the set N^V. The first term in the objective function indicates the number of physical machines required to allocate the physical resources to the virtual machines.
   b) *To minimize total hopping cost*. Total hopping cost is the sum of hopping cost for each virtual edge. Multiple virtual machines can be assigned to single physical machine. In such situation, the hopping cost for the corresponding virtual edge, if exists, is calculated as 0.
2) Constraint (8) ensures that a virtual machine can be assigned to utmost one physical machine.
3) Constraint (9) ensures that there does not exist any VM whose resource requirement is less than the resource available at a particular PM. Such PM should be considered as not eligible to host any VM.
4) Constraints (10) and (11) ensure that the total amount of resources and network bandwidth demanded by the virtual request does not exceed the total amount of available physical resource and network bandwidth, respectively.
5) Constraints (12) and (13) ensure that the maximum amount of resources demanded by any virtual machine does not exceed currently available maximum physical resource at any single physical machine. The constraint also holds in case of network bandwidth.

5 The Virtual Resource Management Algorithm

Following the previous section, where we have formulated the problem of mapping VM-graph onto available PM-graph with the goal to minimize the number of PMs and network elements involved, we propose here the Link based Virtual Resource Management (LVRM) algorithm to map the VMs onto PMs in an efficient way. Further, the proposed resource allocation method is independent of any cloud pricing model. Hence, we do not focus on the optimization of pricing and monetary cost of the user. However, we believe that it may help the CSP to reduce the total monetary cost of the physical resource for the user by minimizing the required amount of physical resources.

The proposed algorithm takes VM-graph G^V and PM-graph G^P as inputs and gives the mapping function

$$H = \{u \rightarrow U : \forall u \in N^V, \exists U \in N^P\} \tag{15}$$

$$PATH = \{\Pi_{uv}^{UV} : (uv) \in E^V, U \in N^P, V \in N^P\}, \tag{16}$$

as output. For all VM u in G^V, the set H represents the PM U in G^P. The set $PATH$ represents the set of all paths after mapping the virtual edges onto the physical edges. Path Π_{uv}^{UV} is the path from PM U to PM V for virtual edge (uv). During the mapping process, the values of different parameters related to PM-graph are updated after mapping each virtual edge. In this way, multiple virtual machines can be mapped onto the single physical machine.

Before the actual mapping starts, the proposed algorithm will copy G^P to $G'^P(N'^P, E'^P)$ by eliminating those PMs that cannot satisfy the resource Constraint (9) for the VMs in G^V. Furthermore, the respective adjacent edges are also eliminated from G'^P. This filtration process fastens the execution of mapping algorithm. Hence, the filtered PM-graph G'^P will be considered in the mapping process. As discussed earlier, we are giving maximum priority to the network bandwidth in the proposed solution and therefore the mapping process will start from the placement of virtual edge with maximum bandwidth demand.

The mapping process will start with sorting the virtual edges in E^V in *descending* order based on their bandwidth demand $BW_{req}(uv)$. Considering the virtual edge with maximum bandwidth demand, a set of candidate edges will be selected in such a way that all the candidate edges satisfy the Constraint (12). Based on the weight, the set of candidate edges will be sorted in descending order. The candidate edge with max weight will be given highest priority. Following this, the edge with highest priority is selected for the feasibility test. Feasibility test ensures that both the associated physical machines can host the VMs associated with the current virtual link. Hence, the PMs associated with the current physical link can fulfill the demand of the VMs. In case of failure of the current candidate edge in feasibility test, the candidate edge with next highest priority will be taken into account. This process will be continued until the virtual edge is mapped to one of the physical edges from the set of candidate edges. It is to be noted that we formulate the resource mapping problem as an optimization problem with an objective to minimize the hopping cost, which can be achieved by reducing the number of PMs and selecting the PMs with minimum number of intermediate

network devices. To meet this objective, suitable physical links must be selected for each virtual edge, which is achieved by taking weight of the physical links into consideration. Detail procedure of this link based virtual resource management is given in Algorithm 1.

Algorithm 1. Link Based Virtual Resource Management Algorithm

Data: VM-graph: $G^V(N^V, E^V)$,
PM-graph: $G^P(N^P, E^P)$
Result: $H = \{u \rightarrow U : \forall u \in N^V, \exists U \in N^P\}$
$PATH = \{\Pi_{uv}^{UV} : (uv) \in E^V, (UV) \in E^P\}$
1 $H = \emptyset$;
2 $PATH = (uv) \rightarrow \emptyset : (uv) \in E^V$;
3 $G'^P(N'^P, E'^P) \leftarrow G^P -$ {all vertex U and their adjacent edges: U cannot host at least one VM};
4 $Q \leftarrow Sort\_on\_BWdemand\_Descend(E^V)$;
5 **while** $q \leftarrow Extract\_Max(Q) \neq NULL$ **do**
6 $v_1 \leftarrow Vertex1(q)$;
7 $v_2 \leftarrow Vertex2(q)$;
8 $PM_1 \leftarrow PM(v_1)$;
9 $PM_2 \leftarrow PM(v_2)$;
10 **if** $PM_1 \neq NULL$ & $PM_2 \neq NULL$ **then**
11 /* Both the virtual machines are already assigned to some PMs */
12 **if** E$(PM_1PM_2) \in E'^P$ **then**
13 **if** $BW_{remain}(PM_1PM_2) \leq BW_{req}(v_1v_2)$ **then**
14 Add $E(PM_1PM_2)$ to $PATH$;
15 **end**
16 **else**
17 $path = DIJKSTRA\_ShortestPath(PM_1, PM_2)$;
18 Add $path$ to $PATH$ for virtual link q ;
19 **end**
20 Update G'^P ;
21 update BW_{remain} of q;
22 update weight of the physical link ;
23 update α_x^P for PM_1 and PM_2 ;
24 Remove q from Q and Goto Line 5 ;
25 **else**
26 Find the set of candidate edges C from E'^P such that it satisfy Constraint (9) and must attached to either PM_1 or PM_2 or both ;
27 Assign the priorities to the candidate edges based on their weight ;
28 **while** $p \leftarrow Extract\_Max(C) \neq NULL$ **do**
29 Let P_1 and P_2 be two PMs connected to p ;
30 /*Check the feasibility test of p */
31 **if** *p can host q* **then**
32 Update H and Map $p \leftrightarrow q$;
33 Add $E(P_1P_2)$ to $PATH$;
34 Update G'^P;
35 update BW_{remain} of q;
36 update weight of the physical link ;
37 update α_x^P for P_1 and P_2;
38 Remove q from Q and Goto Line 5 ;
39 **else**
40 Remove p from C and Goto Line 5 ;
41 **end**
42 **end**
43 **end**
44 **end**

5.1 LVRM Algorithm

The proposed LVRM algorithm starts with filtering the graph G^P in Line 3 by removing the PMs those cannot satisfy the Constraint (9) and the edges attached to those PMs. This removal process helps in improving the running time of the algorithm as the resultant preprocessed graph $G'^P(N'^P, E'^P)$ becomes smaller as compared to the original graph $G^P(N^P, E^P)$. Line 4 sorts the set of virtual edges E^V and assigns a new set of virtual edges Q. As $HEAPSORT$ algorithm can give better performance in terms of time and space complexities, it is used to sort the set of virtual edges. However, any sorting algorithm such as Merge sort and Quick sort could be used too to sort the virtual edges. The reason behind sorting the set of virtual edges is to map the virtual edges with higher bandwidth demand before the edges with lesser bandwidth demand. Hence, the virtual edges are sorted based on their bandwidth demand.

The mapping process starts with extracting the edge from set Q with higher bandwidth demand and continues until an edge can have minimum bandwidth demand (Line 5-44). For each virtual edge $q \in Q$, first the attached VMs are checked if they are hosted by any physical machines. If both VMs u and v are hosted by PMs PM_1 and PM_2, respectively and if edge $E'^P(PM_1PM_2) \neq \emptyset$, the virtual edge $E'(uv)$ is assigned to the physical edge $E'^P(PM_1PM_2)$ based on the remaining bandwidth (BW_{remain}) of $E'^P(PM_1PM_2)$ (Line 13). The Dijkstra's shortest path algorithm is used when $E'^P(PM_1PM_2) = \emptyset$. The Dijkstra algorithm returns the shortest path between PM_1 and PM_2, which is further added to $PATH$ for the virtual edge q. On the other hand, if the virtual node u or v is not yet hosted by any of the physical machines, a set of candidate edges C is selected from E'^P (Line 26) based on the Constraint (9). If any one of the VMs is already assigned to a PM, all candidate edges must be connected to that PM. Based on the weight, priorities are assigned to each candidate edge $p \in C$ (Line 27). As discussed earlier, each candidate edge p must go through the feasibility test. In the feasibility test, the server resource requirement by virtual machines must be satisfied by the physical machines present at both ends of the candidate edge. If the candidate edge is a logical one, in other words, if $P_1 = P_2$, in Line 29, the feasibility test would be as follows:

$$\alpha_x^P(P_1) \geq \alpha_x^V(V_1) + \alpha_x^V(V_2). \tag{17}$$

After the feasibility test is successful, H and $PATH$ are updated and the current virtual edge q is mapped to the current candidate edge p. The preprocessed PM-graph G'^P is updated upon any updating in the resultant set H and $PATH$ (Lines 20-23 and Lines 34-37).

Lemma 1. *For any VM-graph $G^V(N^V, E^V)$ and PM-graph $G^P(N^P, E^P)$, the number of physical machines required to allocate the server resource is less than or equal to the number of virtual machines in the VM-graph. Mathematically*

$$1 \leq \chi(M) \leq |N^V|, \tag{18}$$

where, $\chi(M)$ is the total number of physical machines allocated in the solution M for the set of virtual machines N^V.

Proof. According to Constraint (8), one virtual machine cannot be assigned to multiple physical machines. This implies that the value of $\chi(M)$ cannot exceed $|N^V|$.

Furthermore, as discussed in Section 4.1, one physical machine can host multiple virtual machines. As given in Algorithm 1, Line 29, there is no restriction that the physical machines at both end of the candidate edge p must be different, the algorithm searches for a physical machine that can host the virtual machines at both ends of a virtual edge. This implies that the number of physical machines can be reduced to one for two virtual machines. Hence, the value of $\chi(M)$ can be less than the number of virtual machines in the VM-graph. □

Theorem 1. *The total amount of network bandwidth allocated can be less than the total amount of required network bandwidth. Mathematically*

$$0 \leq R_n \leq r_n, r_n > 0, \tag{19}$$

where, R_n and r_n are the total amount of allocated and required network bandwidth, respectively.

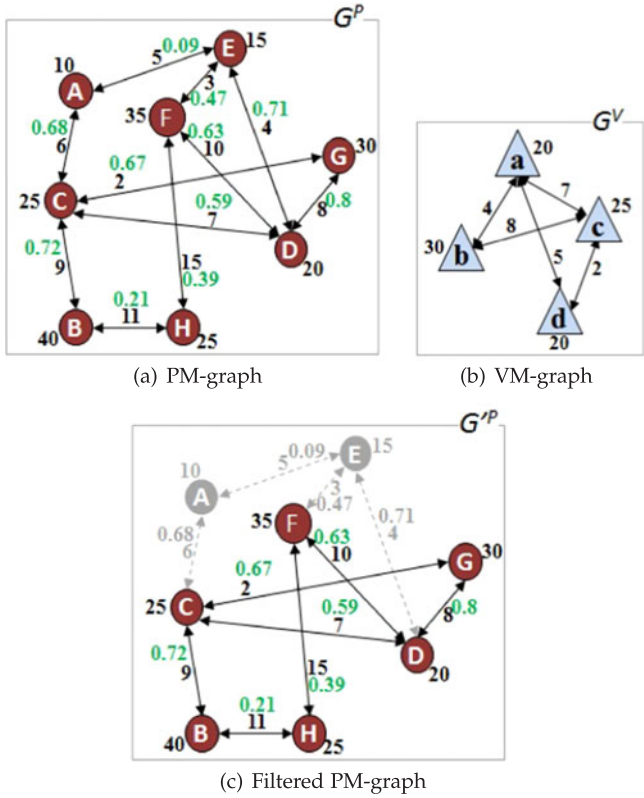
Proof. According to the Constraint (14), the VM-graph must contain at least one virtual edge. Hence, the VM-graph must contain at least two virtual machines. Furthermore, it is proved in Lemma 1 that the minimum number of required physical machines is one for each VM-graph. This implies that one physical machine can be used to serve a virtual request of two VMs and no network bandwidth is required in order to exchange the data between those two virtual machines. □

Theorem 2. *The time complexity of the proposed algorithm is $\mathcal{O}(mE \log n)$, where n and m are the number of PMs and VMs, respectively and E is the number of physical edges.*

Proof. The algorithm starts with filtering the original PM-graph G^P (Line 3) with the time complexity of $\mathcal{O}(n)$. The time taken to sort the virtual edges in Line 4 by $Heapsort$ algorithm is $\mathcal{O}(e\log e)$, where e represents the number of virtual edges. As the value of e is very less for each virtual request, the running time of the sorting algorithm possess very less impact. Mapping process from line 5 to line 44 is iterated for m number of times. For each virtual edge, $\mathcal{O}(n)$ running time is required to find the candidate edges. Selection of one candidate edge will take $\mathcal{O}(n)$ time, if all the physical edges are eligible to host the current virtual edge. Following the selection process, mapping of the virtual edge onto the physical edge and the update of the PM-graph will take constant time $\mathcal{O}(1)$. In the worst case scenario, for every virtual edge the algorithm will invoke Dijkstra shortest path algorithm whose running time is $\mathcal{O}((n+E)\log n)$. Hence, the running time of proposed LVRM algorithm can be calculated as

$$\begin{aligned}&\mathcal{O}(m) \times [\mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}((n+E)\log n)]\\ &= \mathcal{O}(mn) + \mathcal{O}(m) \times [\mathcal{O}(n \log n) + \mathcal{O}(E \log n)]\\ &= \mathcal{O}(mn \log n) + \mathcal{O}(mE \log n).\end{aligned}$$

As the number of physical edges E is more than the number of physical machines n, we can conclude that the worst case time complexity of the proposed algorithm would be $\mathcal{O}(mE \log n)$. □



(a) PM-graph (b) VM-graph

(c) Filtered PM-graph

Fig. 4. An example of LVRM algorithm.

Theorem 3. *The space complexity of the proposed algorithm is $\mathcal{O}(2 * p^2 + v^2)$, where p and v are the numbers of PMs and VMs present in PM and VM graph, respectively.*

Proof. As given in Line 3 of Algorithm 1, the original PM graph G^P is preprocessed by removing all vertices and adjacent edges that cannot host at least one virtual machine. The preprocessed graph is copied in another graph variable G'^P for further execution. In the worst case scenario, the amount of memory space required to store G^P and G'^P is same. Further, considering the list representation of the graph that includes the node and edge representation, the upper bound of the space complexity for the graph G^P is $\mathcal{O}(p^2)$, where p is the number of PMs present in the PM graph, $p = |N^P|$. The total amount of space required is $\mathcal{O}(2 * p^2)$ as in the worst case, the PM graph G^P and G'^P are same. Considering the VM graph, the upper bound of the space complexity for the VM is $\mathcal{O}(v^2)$, where v is the number of VMs present in the VM graph. Combining the space requirement of both PM graph and VM graph, the space complexity for the proposed algorithm can be deduced as $\mathcal{O}(2 * p^2 + v^2)$. □

5.2 Example of LVRM

For better understanding of the proposed LVRM algorithm, we have taken an example as depicted in Fig. 4. Fig. 4b represents the VM-graph consisting of four VMs and PM-graph consists of eight PMs as depicted in Fig. 4a. The integer value in each PM represents the available units of the computing resource. The fractional value in each edge of PM-graph represents the weight and the whole number represents the available units of bandwidth on that communication link. In VM-graph, the numeric value on VMs and

edges represents the required unit of computing resource and network bandwidth, respectively. In these figures, all numeric values are taken randomly and both VM-graph and PM-graph are input to the LVRM algorithm.

Filtered PM-graph G'^P is obtained by removing those PMs that cannot host any of the VMs in VM-graph as depicted in Fig. 4c. The newly processed PM-graph G'^P is derived from G^P by removing PM A and E as they cannot host any of the VMs from the VM-graph. Since, the virtual edge (bc) has maximum demand, it is processed first. As both VMs b and c are not yet assigned to any of the PMs, the set of candidate edges from G^P will be (DG), (BC), and (DF) with weight 0.8, 0.72, and 0.63, respectively. Though three candidate edges are considered in this example, in real implementation, all candidate edges can be taken into consideration. Hence, edges (BH) and (HF) can also be considered as the candidate edges. After calculating the set of candidate edges, the priority value is assigned to each edge based on its weight. Hence, edge (DG) and (BC) get the highest priority and second highest priority, respectively and so on. Each edge needs to go through the feasibility test starting from edge with highest priority to the edge with lowest priority. For edge (DG), physical machine D can host neither VM b nor c. Hence, the edge with next highest priority, i.e., edge (BC) must be taken into consideration to host the virtual edge (bc). For physical edge (BC), both physical machines B and C can host the virtual machines b and c, respectively. Hence, VM c can be assigned to PM C and VM b can be assigned to PM B. In case of both PMs of an edge are eligible to host both VMs of the virtual edge, the VMs are assigned to PMs by following the BESTFIT policy. Likewise, the mapping process can be carried out for other virtual edges such as ac, ad, ab, and cd.

6 Performance Evaluation

In order to evaluate the performance of proposed LVRM algorithm, a discrete event java-based simulator CVI-Sim [4] is used. This allows us to generate the substrate network as well as a huge number of virtual requests using different probabilistic approaches such as the arrival of requests follows the Poisson distribution. This also allows us to distribute the resources to the virtual nodes and links. We compare our algorithm against two popular traditional virtual network embedding algorithms: G-SP [26] and G-MCF [27]. G-SP is the greedy node mapping with shortest path algorithm. The problem of virtual network embedding is addressed by taking virtual network reconfiguration into account. The virtual network is mapped by a basic virtual network assignment algorithm followed by adaptive optimization strategies. On the other hand, G-MCF is a greedy node mapping multi-commodity flow problem. The approach of solving the embedding problem is a two-stage solution. First, virtual links are split over multiple substrate paths. Second, path migration algorithm is executed for better utilization of the substrate network.

6.1 Simulation Setup

The proposed LVRM algorithm is evaluated and is compared with G-SP and G-MCF in our simulation. We have considered one service provider equipped with 100 numbers of physical servers and randomly generated links. The links are generated randomly with a probability value 0.5. This also indicates the connectivity probability of two physical servers. The resources are assigned to the physical servers by following random distribution. Available number of CPUs capacities are randomly assigned to each server ranging from 10 through 20 CPUs. Likewise, the storage and memory capacity of the servers are randomly distributed in between 1000 GB through 2000 GB and 20000 MB through 50000 MB, respectively. Available bandwidth between the physical servers is randomly distributed between 1000 Bps and 10000 Bps. The unit of bandwidth Bps in a link is referred to as Bytes per second.

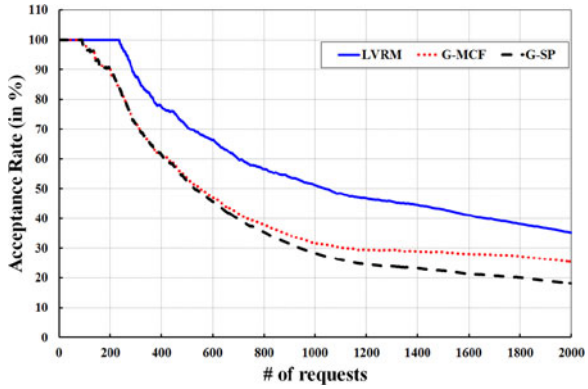


Fig. 5. Acceptance rate in %.

For generating the virtual requests, the number of virtual machines ranges from 2 through 10 for each request. It is assumed that the arrival of requests follows the Poisson distribution with the mean of 5 requests per 100 time units and lifetime of each request follows an exponential distribution with an average of 500 time units of lifetime. In some results, unexpired lifetime of requests is considered. The maximum number of virtual links for each request is 100. The number of links is decided by the link probability 0.5. This indicates the connectivity probability of two virtual machines. The resource demand of each request is randomly distributed. The number of CPUs demand and bandwidth demand for each virtual machine and each virtual link range from 0 through 4 CPUs and 100 Bps through 500 Bps, respectively. Similarly, required storage and memory for each virtual machine range from 500 GB through 1000 GB and 8000 MB through 10000 MB, respectively. Taking above-mentioned performance matrix, the following simulation results are derived.

6.2 Simulation Results

The proposed algorithm is evaluated by comparing its performance with G-SP and G-MCF algorithms. With the increasing number of incoming requests, the behavior of proposed algorithm has been studied in terms of percentage of requests accepted, which is termed as *acceptance rate*, and average number of hop count per virtual edge as depicted in Figs. 5 and 6, respectively. In Fig. 5, the lifetime of the requests is considered to be non-expired, which leads to a gradual decrease in the acceptance rate. In case of LVRM algorithm, the acceptance rate of incoming requests remains 100 percent for more than 200 requests. When the number of requests increases to 2,000, the acceptance rate decreases

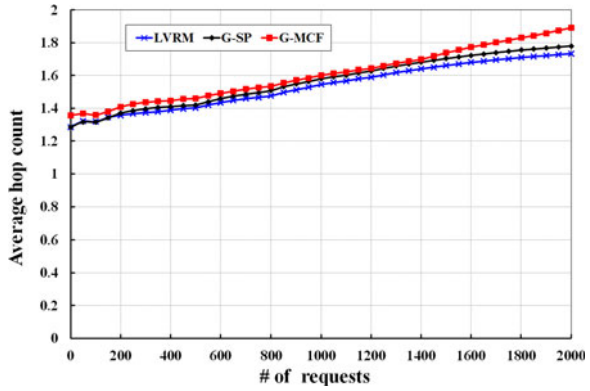


Fig. 6. Average number of hop counts.

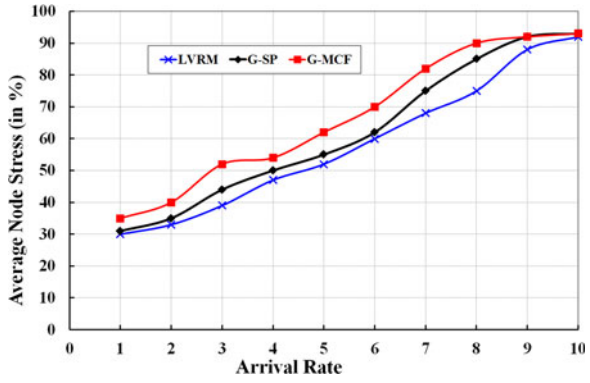


Fig. 7. Average node stress with no expiration lifetime of requests.

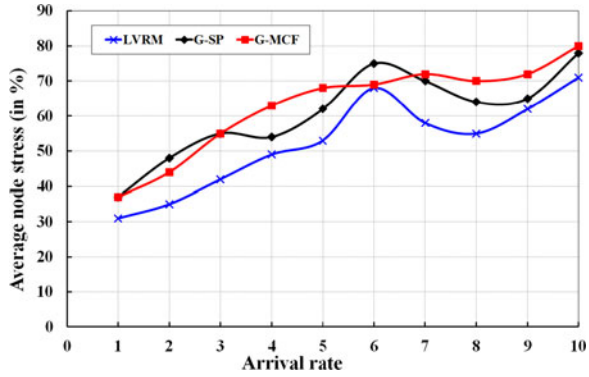


Fig. 8. Average node stress with exponentially distributed lifetime of requests.

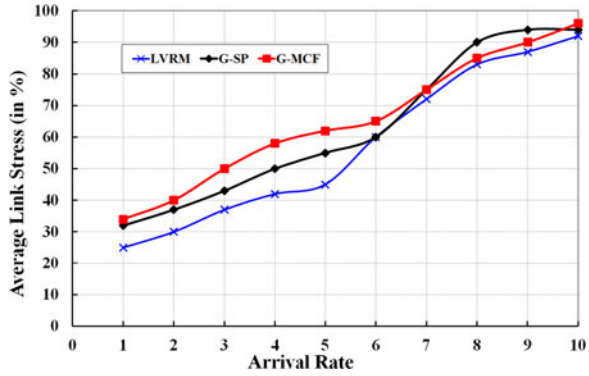


Fig. 9. Average link stress with no expiration lifetime of requests.

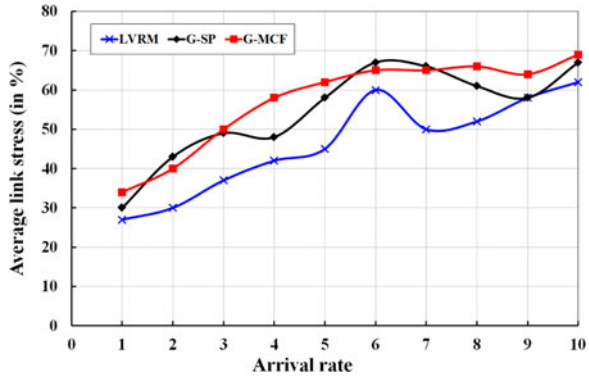


Fig. 10. Average link stress with exponentially distributed lifetime of requests.

to 38 percent, which is 30 and 18 percent in case of G-MCF and G-SP algorithm, respectively. Fig. 6 represents the average hop count per virtual link. Though the acceptance rate is 100 percent for the first 200 requests, the average hop count falls in between 1.2 to 1.4. This indicates that more than one physical node is involved in the communication between two virtual machines.

The stress on node and link has been studied extensively under no-expiration lifetime and exponentially distributed lifetime of each request. Here, stress refers to as the load on physical node and physical link. Figs. 7 and 8 show the average node stress under different arrival rate. With exponentially distributed lifetime of the virtual machines, the average node stress ranges between 50 through 75 percent. On the other hand, the average node stress remains constant after certain number of requests are mapped to the physical network. It can be observed that the node stress gradually increases up to 93 percent and the stress on nodes become stable under no-expiration lifetime of the requests. In practical environment, most of the requests have certain lifetime and after the execution of job the resources are released and leased to other requests as shown in Fig. 8. Here, we have considered that the mean lifetime of the requests is 500 time units. Since the requests have limited lifetime, the average node stress of the virtual machines ranges between 55 through 80 percent. Numbers of incoming requests are directly proportional to the stress on nodes as shown in Fig. 8.

Similar to the node stress, we have also studied the link stress under no-expiration lifetime and exponentially distributed lifetime of the incoming requests, as depicted in Figs. 9 and 10, respectively. In both cases, the arrival rate varies between 1 through 10 number of requests per unit time. The arrival rate refers to as the number of requests that CSP received per unit time. The stress of the links is observed in every 10,000 time units and the average value is plotted on the graph. Overall stress on the network represents the load on the network. More stress indicates more load on the network. Our goal is to minimize the stress on the network for certain number of requests by minimizing the stress on both node and link. The result presented in Fig. 10 shows that the stress on link is balanced and less as compared to other G-SP and G-MCF algorithms. The stress on the link fluctuates between 50 through 70 percent, whereas it fluctuates between 60 through 70 percent in case of G-MCF and G-SP algorithm. However, it is observed that virtual link probability and lifetime of the requests cause non-linear variation in average node stress as shown in

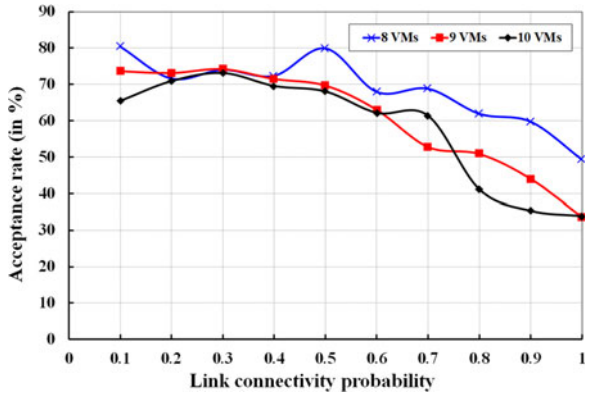


Fig. 11. Link connectivity probability versus acceptance rate.

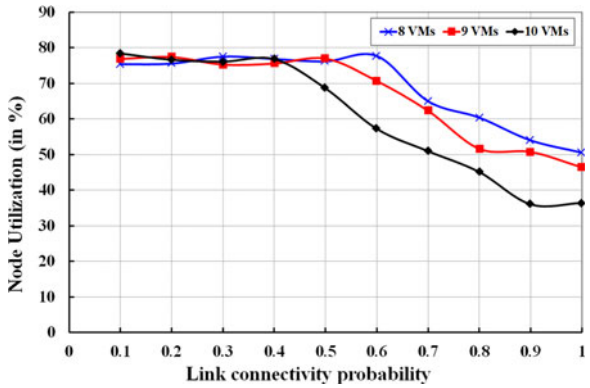


Fig. 12. Link connectivity probability versus node utilization.

Fig. 8 and in average link stress as shown in Fig. 10, when arrival rate is 6.

We have also studied the effect of connectivity on acceptance rate and utilization of both nodes and links. The link connectivity of the virtual requests represents the probability that two virtual machines are connected. The increase in link connectivity value will increase the number of virtual link in a request. In our study, the connectivity value is increased to 1 from 0.1. The connectivity value has direct impact on the acceptance rate as shown in Fig. 11. A request is considered to be rejected, if the exact amount of total required computing and network bandwidth resource is not fulfilled by the CSP. For each request, single attempt is made to map the VM graph onto PM graph. We have considered the requests with only 8, 9, and 10 virtual machines. For the fixed number of virtual machines in a request, the results are observed in every 10,000 time units. We have fixed the number of requests and number of virtual machines to 2,000 and 8, respectively. With this performance matrix, the connectivity value is increased by 0.1 in each simulation. It is observed that the acceptance rate decreases from 80 to 50 percent for fixed number of request and virtual machines. When the number of virtual machines increases, the initial acceptance rate is dropped from 80 to 73 percent. The variable lifetime of the requests at different time instances causes the non-linear variation in the acceptance rate as observed in Fig. 11 when the link connectivity probability is 0.5.

Figs. 12 and 13 show the effect of connectivity on utilization of the physical servers and the physical links. With more number of virtual links in a request, the utilization of the physical links increases, whereas the utilization of nodes

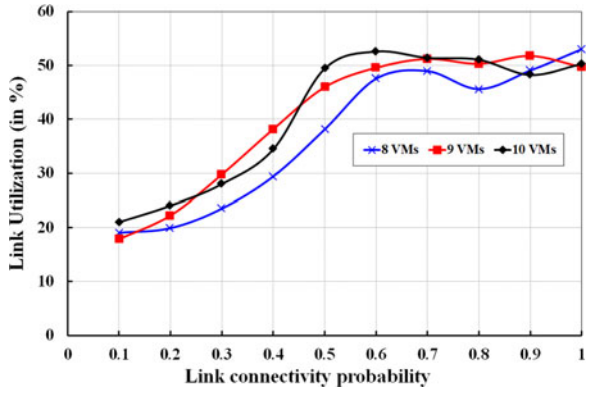


Fig. 13. Link connectivity probability versus link utilization.

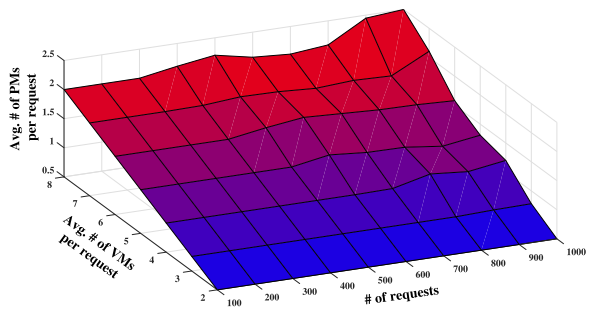


Fig. 14. Avg. number of allocated PMs.

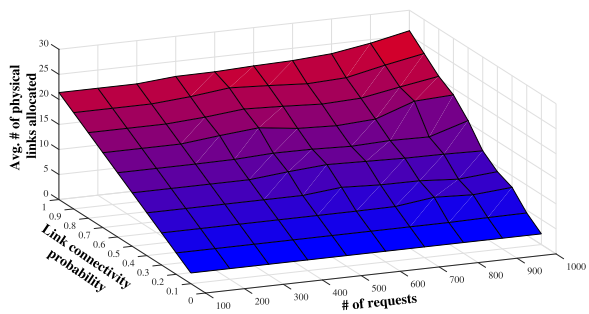


Fig. 15. Avg. number of allocated physical links.

decreases. In our simulation, node utilization involves utilization of CPU, memory, and hard-disk. From Figs. 12 and 13, we can conclude that the connectivity probability also has direct impact on the average hop count per virtual link. Further, a variation in the link and node utilization is observed during the simulation, which is due to the variable lifetime of the requests.

The efficiency of the proposed algorithm in terms of average number of physical machines and links allocated to the requests is shown in Figs. 14 and 15, respectively. In Fig. 14, the number of requests ranges between 100 through 1,000 and the average number of VMs per request ranges between 2 through 8. It is observed that the average number of PMs allocated to each request ranges between 0.5 through 2.5, whereas the link connectivity probability in virtual requests is varied from 0.1 to 1 and the number of VMs in each request is 8 as presented in Fig. 15. Under such demand, it is observed that the average number of physical links required to map the virtual links ranges between 3 to 25. Multiple VMs from the single request are mapped to one PM, which helps minimize the average number of PMs and physical links required for each request.

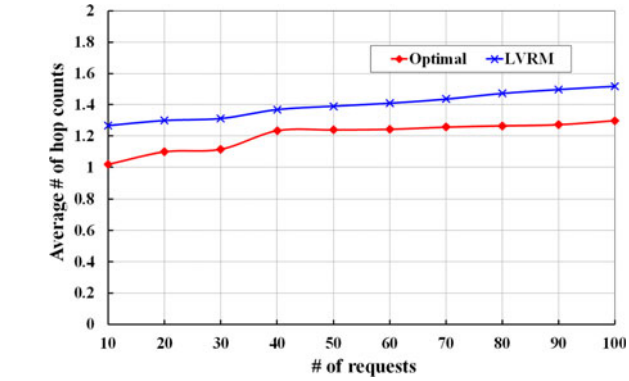


Fig. 16. Comparison of LVRM with optimal allocation in terms of hop counts.

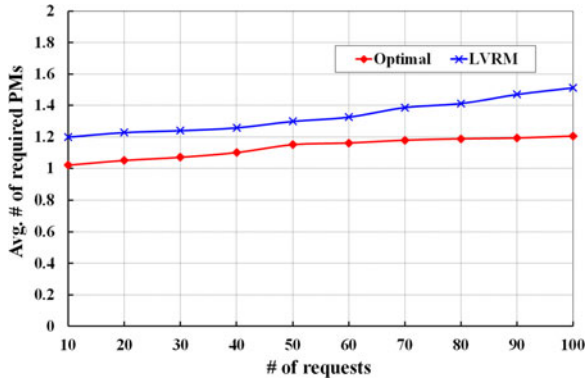


Fig. 17. Comparison of LVRM with optimal allocation in terms of number of required PMs.

In order to demonstrate how close is our proposed algorithm to the optimal solution, we have compared our results obtained from the LVRM algorithm with the optimal one. The optimal result in terms of average number of hop counts and required numbers of PMs is obtained by comparing all possible mapping solutions. The experimental environment consists of 30 physical machines with link connectivity probability of 0.8. The number of virtual requests ranges from 10 through 100. Each virtual request consists of 2 to 4 numbers of VMs with a minimum link connectivity probability of 0.9.

Fig. 16 shows the comparison of our proposed LVRM algorithm with optimal solution in terms of number of hops. The average number of hop counts in the proposed LVRM ranges between 1.267 and 1.518 while maps from 10 to 100 numbers of virtual requests. The number of hop counts ranges from 1.019 through 1.298 for mapping the same number of virtual requests. In Fig. 17, comparison of proposed LVRM algorithm with optimal solution is done in terms of average number of required PMs. An average of approximately 1.2 and 1.02 PMs is required to map 10 virtual requests in case of LVRM and optimal solution, respectively. However, the average number of PMs increases to 1.5 to map 100 virtual requests in case of LVRM algorithm, which is approximately 25 percent more than that of the optimal solution.

7 Conclusions

In this paper, we deal with virtual resource mapping problems and have proposed an efficient link based virtual resource management algorithm to minimize the number of physical machines. In the proposed approach, virtual links are mapped followed by the virtual nodes as the inefficient mapping of virtual links have direct impact on the execution time of the corresponding tasks even if the computation resource demand is fulfilled. The virtual links with highest network bandwidth demand are given maximum priority. Graph theory is used as a tool to represent the incoming tasks, required VMs and available PMs. Dijkstra algorithm is applied to find the substrate path between two physical machines. Furthermore, the proposed algorithm takes advantage of assigning multiple VMs to single PM to ignore the network demand of the corresponding VMs. This is one of the reasons for higher acceptance rate of the users' requests. Besides, our proposed algorithms are simulated extensively to compare our result with similar algorithms along this direction and the superiority of our proposed approach over others is clearly demonstrated. However, in order to verify and improve the results, we strive to implement the proposed virtual resource mapping mechanism in real cloud environment, which will be part of our future work.

Acknowledgments

This work is partly supported by Ministry of Science and Technology (MOST), Taiwan under the grant number 106-2221-E-182-014.

References

[1] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Trans. Serv. Comput.*, vol. 5, no. 2, pp. 164–177, Apr.–Jun. 2012.
[2] Y. Dong, X. Zhang, J. Dai, and H. Guan, "HYVI: A hybrid virtualization solution balancing performance and manageability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2332–2341, Sep. 2014.
[3] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM*, 2012, pp. 2876–2880.
[4] C. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, C. Cervello-Pastor, and A. Monje, "On the optimal allocation of virtual resources in cloud computing networks," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1060–1071, Jun. 2013.
[5] A. Aral and T. Ovatman, "Network-aware embedding of virtual machine clusters onto federated cloud infrastructure," *J. Syst. Softw.*, vol. 120, pp. 89–104, 2016.
[6] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "Placing virtual machines to optimize cloud gaming experience," *IEEE Trans. Cloud Comput.*, vol. 3, no. 1, pp. 42–53, Jan. 2015.
[7] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein, "Virtual network embedding with opportunistic resource sharing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 816–827, Mar. 2014.
[8] T. Wang and M. Hamdi, "Presto: Towards efficient online virtual network embedding in virtualized cloud data centers," *Comput. Netw.*, vol. 106, pp. 196–208, 2016.
[9] S. Haeri and L. Trajkovic, "Virtual network embedding via Monte Carlo tree search," *IEEE Trans. Cybern.*, vol. PP, no. 99, pp. 1–12, 2017, doi: 10.1109/TCYB.2016.2645123.
[10] J. Duan and Y. Yang, "Placement and performance analysis of virtual multicast networks in fat-tree data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 3013–3028, Oct. 2016.
[11] Z. Yang and Y. Guo, "An exact virtual network embedding algorithm based on integer linear programming for virtual network request with location constraint," *China Commun.*, vol. 13, no. 8, pp. 177–183, Aug. 2016.
[12] L. Zhang, X. Yin, Z. Li, and C. Wu, "Hierarchical virtual machine placement in modular data centers," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, 2015, pp. 171–178.

[13] A. Jarray and A. Karmouch, "Decomposition approaches for virtual network embedding with one-shot node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 1012–1025, Jun. 2015.
[14] L. Mashayekhy, M. M. Nejad, and D. Grosu, "A PTAS mechanism for provisioning and allocation of heterogeneous cloud resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 9, pp. 2386–2399, Sep. 2015.
[15] N. Ogino, T. Kitahara, S. Arakawa, and M. Murata, "Virtual network embedding with multiple priority classes sharing substrate resources," *Comput. Netw.*, vol. 112, pp. 52–66, Jan. 2017.
[16] X. Hesselbach, J. R. Amazonas, S. Villanueva, and J. F. Botero, "Coordinated node and link mapping {VNE} using a new paths algebra strategy," *J. Netw. Comput. Appl.*, vol. 69, pp. 14–26, 2016.
[17] G. Sun, V. Anand, D. Liao, C. Lu, X. Zhang, and N.-H. Bao, "Power-efficient provisioning for online virtual network requests in cloud-based data centers," *IEEE Syst. J.*, vol. 9, no. 2, pp. 427–441, Jun. 2015.
[18] J. A. Aroca, A. F. Anta, M. A. Mosteiro, C. Thraves, and L. Wang, "Power-efficient assignment of virtual machines to physical machines," *Future Generation Comput. Syst.*, vol. 54, pp. 82–94, 2016.
[19] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Trans. Comput.*, vol. 63, no. 11, pp. 2647–2660, Nov. 2014.
[20] H. Liu and B. He, "F2C: Enabling fair and fine-grained resource sharing in multi-tenant IaaS clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2589–2602, Sep. 2016.
[21] H. Shen and Z. Li, "New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2682–2697, Sep. 2016.
[22] W. Hou, Z. Ning, L. Guo, Z. Chen, and M. S. Obaidat, "Novel framework of risk-aware virtual network embedding in optical data center networks," *IEEE Syst. J.*, vol. PP, no. 99, pp. 1–10, 2017, doi: 10.1109/JSYST.2017.2673828.
[23] P. Choudhury, P. Chakrabarti, and R. Kumar, "Online scheduling of dynamic task graphs with communication and contention for multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 1, pp. 126–133, Jan. 2012.
[24] K. Kanoun, N. Mastronarde, D. Atienza, and M. van der Schaar, "Online energy-efficient task-graph scheduling for multicore platforms," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 33, no. 8, pp. 1194–1207, Aug. 2014.
[25] C. Gunaratne, K. Christensen, B. Nordman, and S. Suen, "Reducing the energy consumption of ethernet with adaptive link rate (ALR)," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 448–461, Apr. 2008.
[26] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.
[27] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, 2008.

Prasan Kumar Sahoo received the BSc (with Honors) degree in physics and the MSc degree in mathematics from Utkal University, India, in 1987 and 1994, respectively. He received the MTech degree in computer science from the Indian Institute of Technology (IIT), Kharagpur, India, in 2000 and the PhD degree in mathematics from Utkal University, India, in 2002, and the 2nd PhD degree in computer science and information engineering from National Central University, Taiwan, in 2009. He is currently a professor in the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. He is an associate researcher in the Department of Cardiology, Chang Gung Memorial Hospital, Linkou since 2016. He has worked as an associate professor in the Department of Information Management, Vanung University, Taiwan from 2007 to 2011. He was director of International Affairs Center, Chang Gung University from Feb, 2013 to Jan, 2017. He was a visiting associate professor in the Department of Computer Science, Universite Claude Bernard Lyon 1, France. His current research interests include big data analytic, cloud computing, and IoT. He is an Editorial Board member of the *International Journal of Vehicle Information and Communication Systems* (IJVIC) and has served as the Program Committee member of several IEEE and ACM conferences. He was Program chair of ICCT, 2010 and is a senior member of the IEEE.

Chinmaya Kumar Dehury received the BCA degree from Sambalpur University, India, in June 2009 and the MCA degree from Biju Pattnaik University, India, in June 2013. Currently he is working toward the PhD degree in the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. His research interests include scheduling, resource management, and fault tolerance problems of cloud computing.

Bharadwaj Veeravalli received the BSc degree in physics from Madurai-Kamaraj University, India, in 1987, the master's degree in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1991, and the PhD degree from the Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India, in 1994. He received gold medals for his bachelor degree overall performance and for an outstanding PhD thesis (IISc, Bangalore India) in the years 1987 and 1994, respectively. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering (CIE) Division, National University of Singapore, Singapore, as a tenured associate professor. His main stream research interests include cloud/grid/cluster computing, scheduling in parallel and distributed systems, and multimedia computing. He is one of the earliest researchers in the field of Divisible Load Theory (DLT). He is currently serving the editorial board of the *IEEE Transactions on Cloud Computing* as an associate editor. He is a senior member of the IEEE and the IEEE-CS.