DOI 10.12694/scpe.v20i2.1526
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 20, Number 2, pp. 335–347. http://www.scpe.org

PLACEMENT STRATEGY FOR INTERCOMMUNICATING TASKS OF AN ELASTIC REQUEST IN FOG-CLOUD ENVIRONMENT

R. SRIDHARAN* AND S. DOMNIC †

Abstract. Cloud computing hosts large number of modern day applications using the virtualization concept. However, end-to-end network latencies are detrimental to the performance of IoT (Internet of Things) applications like video surveillance and health monitoring. Although edge/fog computing alleviates the latency concerns to some extent, it is still not suitable for applications having intercommunicating tasks. Further, these applications can be elastic in nature and demand more tasks during their life-time. To address this gap, in this paper a network aware co-allocation strategy for the tasks of an individual applications is proposed. After modelling the problem using bin packing approach with additional constraints, the authors propose a novel heuristic IcAPER,(Intercommunication Aware Placement for Elastic Requests) algorithm. The proposed algorithm uses the network neighborhood machine for placement, once current resource is fully utilized by the application. Using *CloudsimPlus* simulator the performance IcAPER algorithm is compared with First Come First Serve (FCFS), Random and First Fit Decreasing (FFD) algorithms for the parameters (a) resource utilization (b) resource fragmentation and (c) Number of requests having intercommunicating tasks placed on to same PM. Extensive simulation results shows IcAPER maps 34% more tasks on to the same PM and also increase the resource utilization by 13% while decreasing the resource fragmentation by 37.8% when compared to other algorithms in our consideration.

Key words: Fog computing, Cloud computing, Elasticity, VM placement, Intercommunicating task, Network latency, Autoscaling

AMS subject classifications. 68M14, 68M20

1. Introduction.

Due to proliferation of Internet-of-Things (IoT), lot of smart devices utilize information from sensors attached to them and applications of IoT devices have become more personalized, automatized, and intelligent. These devices can adapt to their surroundings, and communicate with other devices to create new forms of smart, intelligent, and autonomous services. Generally smart devices have inadequate computation power, storage and bandwidth. Therefore, new forms of services are backed by cloud resources as cloud provides virtualized elastic resource at low cost using pay-per-use model in their huge computing and storage resources.

However applications, such as video surveillance, real time gaming, real time streaming and augmented reality are too sensitive towards reliability and latency. Since cloud data centers are located near the core network, the information transmitted from/to the end devices of these applications from/to cloud leads to unacceptable latency besides the problems like location-awareness and geo-distribution of IoT applications.

Fog computing paradigm ensures low-latency, high-bandwidth and location-awareness to the IoT applications.This paradigm pushes the elastic resources of cloud namely, computing and storage to the network edges, i.e. near to the data generation point. As and when resources of fog servers are exhausted, naturally computation and storage are pushed back to cloud resource. Combined resources of fog-cloud environment is referred as fog Cloud Datacenter (fogCDC) in this paper. Fog computing comprises of micro-clouds (fog nodes) that are placed at the edge of data originator.These micro-clouds have the mechanism for processing the data before going on to the network. Hence, fog computing structure is useful in big IoT data analytics [1]. High level overview of fog platforms and architecture is presented in [2] along with implementation of Virtual Machines migration in fog computing. Since fog computing cooperates with cloud computing for its resource demand, together with IoT edge devices, a three layered service delivery model as shown in Fig. 1.1, is adopted for fog computing. Both cloudlet [3,4] that was built before the emergence of fog computing, but inherently coincides with fog and ParaDrop [5] that uses gateway (WiFi access point or home set-top box) for fog implementation, essentially highlight the role of virtualization in a fog architecture. It is to be noted that cloudlet is nothing but a task that is mapped on to an auto-destroyable VM, once the task completes its execution. Hence task and VM are interchangeably used in this paper. Group of VMs launched in the distributed fog-cloud servers working together for a specific surveillance task is demonstrated in [6]. Currently majority of the IoT systems and software are built on top of open Web-based technologies [7]. Authors of [8], review the current efforts in realizing

*National Institute of Technology Trichy, Tiruchirapalli- 620 015. Tamil Nadu, India. (sridharter@gmail.com).
†National Institute of Technology Trichy, Tiruchirapalli- 620 015. Tamil Nadu, India (domnic@nitt.edu).

FIG. 1.1. *Three layered architecture of Fog Computing*

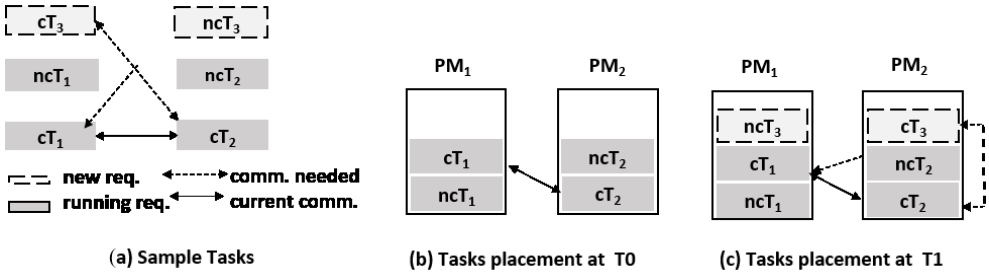

FIG. 1.2. *Placement of sample Tasks using Load balancing FCFS*

big IoT data analytics on fog networking. Therefore, we can easily conclude that majority of application types can use VM based Fog-cloud environment for their processing.

Some of the VMs of these application need inter-communication. For example VMs hosting database task and hosting application task need inter communication in a three-tier web applications. Similarly VMs that hosts object tracker task in a video surveillance system that uses multiple cameras covering multiple areas, need inter-task communication. As these applications request sporadic resources, they depend heavily on elastic capability of the fog-cloud paradigm. Elasticity can be classified as two types: horizontal and vertical. In horizontal elasticity, dynamically the number of VMs are adjusted, whereas the resources (CPU, storage and Memory) of a VM are dynamically adjusted in vertical elasticity [9]. In such a scenario mapping of tasks to the suitable Physical Machine (PM)[1], called Virtual Machine Placement (VMP), is a challenging cloud problem.

To summarize, a cloud request can be classified as those having inter-communicating VMs and those not having inter-communicating VMs, which can be further classified as needing elasticity and not. Authors, confines this work to find suitable placement for application tasks that need inter-communication and horizontal elasticity as well, hereafter called as simply elastic request. Other requests are referred as non-elastic requests, which may or may not have inter-communicating VMs. Applications having inter-communicating VMs, gives better performance, when they are placed together on to the same PM [10] as there will not be any network latency between these VMs. But general VM placement schemes fail to recognize this and place the VMs on to PMs in an arbitrary manner.

Consider an example of five requests that are currently(time t_0) executing in a datacenter. Let the first

[1]PM denotes fog server as well as the servers of cloud Datacenter

(ncT_1*)* and third *(*ncT_2 *)* requests, having one tasks each, does not require inter-communication. Let the second request having two task *(*cT_1 *and* cT_2 *)*needs inter-communication. Assume that the fourth *(* ncT_3 and fifth *(*cT_3*)* are newly (time t_1) arrived requests. While the forth request is non-intercommunicating type, the fifth request is an additional task of second request that needs inter communication with *(*cT_1 *and* cT_2 *).* Further, these tasks follow the communication pattern represented by arrows, as shown in Fig. 1.2(a). Assuming that we have two PMs to place these tasks. Using the widely used VM placement algorithm First Come First Serve (FCFS) along with load balancing, placements of tasks at t_0 resembles to that of Fig. 1.2(b). Placement of tasks at t_1 is shown Fig. 1.2(c). If all the tasks are placed onto single PM itself, the other PM could be switched off to reduce the overall power consumption. Else placing all the inter-communicating tasks *(*cT_1*,* cT_2 *and* cT_3*)* on to the same physical machine reduces the power consumption used for communication, as network latency among these tasks is nil. Since fog computing uses the cloud resources once their resources are fully utilized, placements for inter-communication task of a single request can happen both at fog resources and cloud resources as well. Particularly, when elastic requirements of this request have to be addressed, placements using both fog and cloud resources in unavoidable.

Motivated by this observation, in this paper, after formulating Inter-communication Aware Task Placement of Elastic Request (IcAPER) problem, we describe in detail, the proposed IcAPER algorithm that improves the resource management of fogCDC. Performance of individual fog applications is also bound to improve using this algorithm, as it implements co-allocation for tasks of these applications. Co-allocation is also adhered in cloud resources once fog server is fully utilized. This is achieved by maintaining the identity of tasks with reference to its application across fogCDC. In summary, contributions of this paper are as follows:

- A mathematical modeling of Inter-communication Aware Task Placement of an Elastic Request (IcAPER) problem. This model considers requests of individual IoT application having one or more tasks along with its communication pattern and elastic needs during their lifetime for placement instead of individual tasks, a widely followed practice is presented.
- Guided by IcAPER model, IcAPER algorithm that minimize the network latency and improves the datacenter efficiency is proposed.
- Effectiveness of the proposed algorithm is demonstrated in a simulation environment via (a) reduced VM migrations, (b) resource fragmentation, (c) improved resource utilization and (d) Number of requests having intercommunicating tasks placed on to same PM.

Rest of this paper is organized as follows. *Section 2* reviews the available literature related to the problem. While Section 3 describes the problem formulation and its modeling, Section 4 describes details the proposed IcAPER algorithm along with illustration of IcAPER placement for the sample task described in Fig. 1.2(a). Section 5 presents the implementation and analysis of IcAPER and Section 6 concludes this work.

2. Related Works. Fog computing, relatively a new distributed computing paradigm, extends the cloud service to the edge of the network [11]. Fog device, any element in the network capable of hosting application modules, include resource-rich servers and resource-poor devices such as gateways, mobile, smart vehicles, smart cameras etc. As both fog resource and cloud resources can host VMs, this work reviews the general VM placement, a multi-parameter optimization problem in this section. Simulation environments [12,13,14] have been used in evaluating the solutions for this problem .

Multi-dimensional bin-packing problem is used to model the VMP, where PMs are considered as bins and VMs are considered as items, with the goal of finding minimum number of bins to hold all the items. Since bin-packing is a NP-hard problem, heuristic based approximation algorithms are proposed for VMP. First Fit (FF) algorithm [15], a greedy method is the most popular algorithm used for solving bin-packing problem. FF keeps filling the current bin until the PM could not hold the new VM. Even though this approach is time-effective, it usually occupies more PM, than that of an optimal solution. Modified version of FF is proposed by Panigrahy et al. [16] where VMs are sorted in decreasing order before applying FF on the sorted items. This is called First Fit Decreasing (FFD) algorithm which ensures that large items are placed first.

Authors of [17] proposed a dynamic mapping between new VMs and the PMs using probability theory, and demonstrate that the energy consumption of DC is minimal. While authors of [18] proposed the VMP solution for a balanced PMs interms of load in a DC, others [19, 20] tried to minimize the power consumption of DC by employing consolidation techniques to ensure that active PMs are always minimal by maximizing their

utilization. Mahfuzur et al., [21], proposed static initial VM placement where, VMs are co-placed based on their resource requirement compatibility. In this approach, PMs and VMs are differentiated as resource critical and non-critical based on their historical usage. But they have not considered the elastic requirements of an application hosted by these VMs. To minimize the power consumption, a decentralized decision based VMP is proposed by Feller et al., [22], in which, a peer-peer (P2P) communication model is proposed among PMs. Deploying this communication model, periodic consolidations of VMs are accomplished by using cost-aware Ant Colony Optimization (ACO) techniques. Cui et al., [23] brought out the importance of policy and network awareness before considering VM migrations. The authors demonstrated an efficient VM management scheme which reduces the communication cost while considering PMs to migrate the VMs.

Further, some contributions have explored two-stage approach as well for VMP problem. Beloglazove et al., [24] proposed two stages for the VMP: (i) initial placement of VMs and (ii) optimizing the current placement. In this, using CPU utilization of the VMs, a Modified Best Fit Decreasing (MBFD) algorithm is used for initial placement and the second phase is triggered whenever an overloaded or an under-loaded PM is detected as a result of a well-defined CPU threshold. Without considering elasticity, Zheng et al., [25] subdivided the VMP as two sub-problems. A Best Fit (BF) algorithm, used for incremental placements acts as first step followed by another step in which, periodical trigger of VM consolidation focusing on reducing power consumption and minimization of resource wastage.

An interesting two-phase online VMP is proposed by Shi et al., [26] where PMs and VM requests are represented as vectors. During the first phase, based on cosine similarity of the vectors, PM types are assigned to VM requests and based on resource request, VMs are categorized for each PM type. Using utilization thresholds of PMs, reconfiguration of VMs is triggered in the second phase. Like [24] here also authors have not taken into account elasticity at all. Another two phase approached for VMP proposed by Fabio et al., [27] considers complex IaaS environment including elasticity but did not consider the intercommunication patterns while placing the additional VM of an elastic requests. Mishra et al., [28] proposed algorithm to map the task onto VM and the VMs to PM. This algorithm minimizes the makespan and task rejection rate while reducing the energy consumption by reducing the active number PMs. The reviewed works, except [21,27,10,23] solve VMP problem as simple placement strategy using the current PM resource utilization for new VM placement, consolidation and energy consumption. Authors [21,27] include the elasticity in their works but not intercommunication whereas, authors of [10, 23] include network awareness but not elasticity.

As explained in Section 1, tasks placement accomplished after considering both the network awareness and intercommunicating needs of the tasks, together with the knowledge of prior placement of tasks from this application is bound to increase the application performance. Hence, this work propose to use prior tasks placement knowledge of an intercommunicating elastic application, for the placement of its additional tasks, while ensuring reduced overhead of fogCDC and also increased application performance.

3. Problem Statement and Proposed Model. This section explain the problem that is addressed through this work and its mathematical programming model followed tasks placements using IcAPER for the sample tasks described in Section 1.

3.1. Problem statement. The focus of this work is to improve task placement of an elastic IoT application requests for overall betterment of fogCDC management. By and large, and also by design, only volume of data generated by an IoT application can change over period of time and hence need more processing power to arrive timely decisions, which can be addressed by elasticity of fogCDC. These tasks requests can be classified into three types, namely (a) new VM requests, (b) migrating VMs requests and (c) incremental VM requests. Requests wanting elasticity service are expected to give the minimum number of VMs ($VMmin$) that needs to be running permanently during the lifetime of those applications. They are also expected to give maximum number of VMs ($VMmax$) that can be allocated at any given point of time to cater to the application fleets. In effect, the number of VMs dedicated to these applications at any time shall be between $VMmin$ and $VMmax$. This requirement is in consistent with popular cloud service providers like Google and Amazon. In addition users are also expected specify need of intercommunication to their tasks. Since, application logic remains same for the life-time of an IoT applications, using the historical data, the service providers can also easily identify inter-communication needs of these tasks.

Therefore, we can assign application driven unique identifier for these tasks. Using this identifier we can

easily influence the task scheduler for a desired resource mapping for these tasks. Efficiently placed *VMmin* and incremental VMs of these requests can give various benefits to both user and fog-cloud service provider. These incremental VM requests, when solved as a fixed placement problem result in cascading benefits. Firstly, inter VM communication latency is reduced as network is not involved, which is bound to increase the application performance. Secondly, placing *VMmin* VMs together reduces the potential migrations during cloud server consolidation. This reduced migration and network latency leads to minimal power consumption in fogCDC. Thirdly, as individual PMs are balanced with the placement of VMs, serving both communication depend applications and applications made up of non-communicating tasks, SLA violations are bound to be minimal with respect to inter-communicating tasks.

3.2. Proposed Model. The proposed work consider the communication pattern of the tasks of a request and also the elastic needs of requests as additional constraints for the general VMP problem (GVMP), to formulate the IcAPER problem. IcAPER can be viewed as a bin packing problem where PMs are treated as bins and VMs are treated as items. This is an optimization problem in which number of PMs (bins) used to place the VMs (items) are to be minimized. This work segregate the requests into three types: (a) requests for placing group of tasks having inter-communication along with elastic needs (r^e_{comu}), (b)without elasticity but needs inter-communication (r_{comu}) and (c)all other requests (r).

To formalize the problem we make the following assumptions:

- Let r_1, r_2, r_αbe the requests defined by a four-tuple r_k= $\{u, VM_{type}, VM_{min}, VM_{max}\}, 1 \leq k \leq \alpha$ where VM_{min} and VM_{max} are number of VMs, $VM_{type} = \{micro, small, medium, large, xlarge\}$ and u= $\{r^e_{comu}, r_{comu}, r\}$.
- $p = \{p_1, p_2, p_n\}$ and $v = \{v_1, v_2, v_m\}$ as the identifier of PMs and VMs respectively
- Both PMs and VMs are constrained with D-dimensional resource vectors R (number of CPUs, memory size, storage space, bandwidth speed etc.).
- n represents total number of PMs, m represents total number of requests and N represents VM_{min}, Then from the above assumptions we have,

$$r_k = \sum_{i=1}^{VM_{min}} v_i \tag{3.1}$$

Hence,

$$m = \sum_{k=1}^{\alpha} r_k \tag{3.2}$$

Let us introduce a function ϕ to indicate the placement schema $\phi(r_k) = p$ if VMs of r_k th request are placed on the pth PM. The objective of GVMP is, given a request sequence S, and PMs $p, 1 \leq p \leq n$ with capacity C, find a placement function ϕ such that

$$minimize \sum_{i=1}^{n} p_i \tag{3.3}$$

subject to the following two constraints.

- **Resource constraints:** During the placement of VMs belonging to r_k requests, the PMs identified to host these VMs are subject to primary resource constraints:

$$\forall 1 \leq p \leq n, 1 \leq d \leq D \quad \sum_{r_k=1}^{m} \sum_{v=1}^{r_k} X_{vp} * R^d_v \leq C \tag{3.4}$$

where $R^d_v (1 \leq d \leq D)$ is the resource demand of v^{th} VM and

$$X_{vp} = \begin{cases} 1, \\ 0, \quad \text{otherwise} \end{cases}$$

- **Placement constraints:** Each VM of user request r has to be placed to run on a single PM

$$\forall 1 \leq p \leq n, \quad \sum_{r_k=1}^{m} \sum_{v=1}^{r_k} X_{vp} = 1 \tag{3.5}$$

Each item has resource vector $R_i = \{R_i^1, R_i^2, .R_i^d\}$ and each dimension $l \in [1, .., d]$, denotes resource demand size R_i^l which is normalized 0..1 and capacity of each PM is also normalized to 1, then we have

$$X_{vp} \in \{0, 1\} \quad \text{and} \quad n \in \{0, 1\} \tag{3.6}$$

Next IcAPER is formulated by adding three more constraints, as described below.

- ***Same-PM* constraint:** In any request having inter-communication tasks (r^e_{comu}), let VMs v_1 and v_2 have co-allocation requirement and are required to be co-allocated onto same PM $p \in [1n]$ then we can combine VM v_1 and v_2 to one group and form a new VM v_{new}. For each resource dimension d, the size of the group VM is denoted by sum of VM v_1 and v_2.

$$i.e. R^d_{v_{new}} = \quad R^d_{v_1} + R^d_{v_2} \tag{3.7}$$

Hence the following constraint is added to GVMP:

$$\begin{aligned} \forall p \in [1n], \{\forall v_1, v_2 \in [1..VM_{min}]\} \in r_k \quad \& \\ r_k \in \{r^e_{comu}, r_{comu}\}, \quad X_{v_1 p} = X_{v_2 p} \end{aligned} \tag{3.8}$$

- ***Fixed-PM* constraint:** Let us define matrix A_{ij}, where $i, j \in \{1m\}$, to denote the request relation among individual tasks. As explained in previous section, we also set the individual element of this matrix with r_k, the application driven request identifier, to which the current tasks belongs.This identifier is uniquely identifiable across fogCDC. Assume that $r_k^{'}$ denotes the elastic requirements of request r_k, then for each tasks of an inter-communication needed elastic task group, we add the following fixed-PM constraint:

$$\forall r_k \in r^e_{comu}, \forall 1 \leq k \leq \alpha, \quad x_{vp} = r_k^{'} = r_k \tag{3.9}$$

Comment: All the above explained constraints are suitable for cloud resources only. This is because, in normal circumstances, there will be only a single resource rich fog server will be available per area to which all IoT applications send the data. Fog architecture depends solely on cloud resource for computation-heavy tasks. Hence the objective Eq. 3.3 with constraints of *cf. Eqs. 3.4, 3.5, and 3.8* are to be solved for an effective initial VMP of an individual request while the objective *cf. Eq. 3.3* with constraints of Eqs. 3.4 and 3.5 and Eqs. 3.4 and 3.9 to be satisfied during the additional VM requests of an intercommunicating elastic request.

Claim: IcAPER is NP-Complete.

Proof: As per constraint Eq. 3.8, resource demand vectors of all the tasks specified by VMmin r^e_{comu} and r_{comu} are combined to form a new VM having higher resource demand vector. Characteristics of VM, an equivalent to item in a bin-packing problem are not altered by adhering to this constraint. As stated in Eq. 3.6 we can equate the normalized IcAPER problem to that of normalized GVMP by dividing VM sizes with the corresponding resource capacities of PMs. Hence similar to GVMP, IcAPER is also NP-Complete.

Since minimization problem of IcAPER is NP-Complete, we cannot get the optimal placement solutions in polynomial time unless P=NP [29]. Hence we propose a algorithmic based solution in this paper.

4. Proposed IcAPER Algorithm. In this section, we present a VMP scheme guided by IcAPER problem modelling based on request type r^e_{comu}, r_{comu} and r rather than VM_{type} for optimal operations of fogDC. This scheme recognizes inter-communication needs of the requests for tasks placements. Initial placement focuses on placing the tasks of individual requests efficiently so that future migrations are minimal. This also ensures that the network overheads and overheads incurred for identifying suitable VMs and PMs that are candidates for these migrations, are also reduced. Subsequently, placement of elastic needs of requests having inter-communication are confined to the individual PM resulting in better application performance due to nil network latency. To develop such an algorithm, we make some simplifying assumptions as follows:

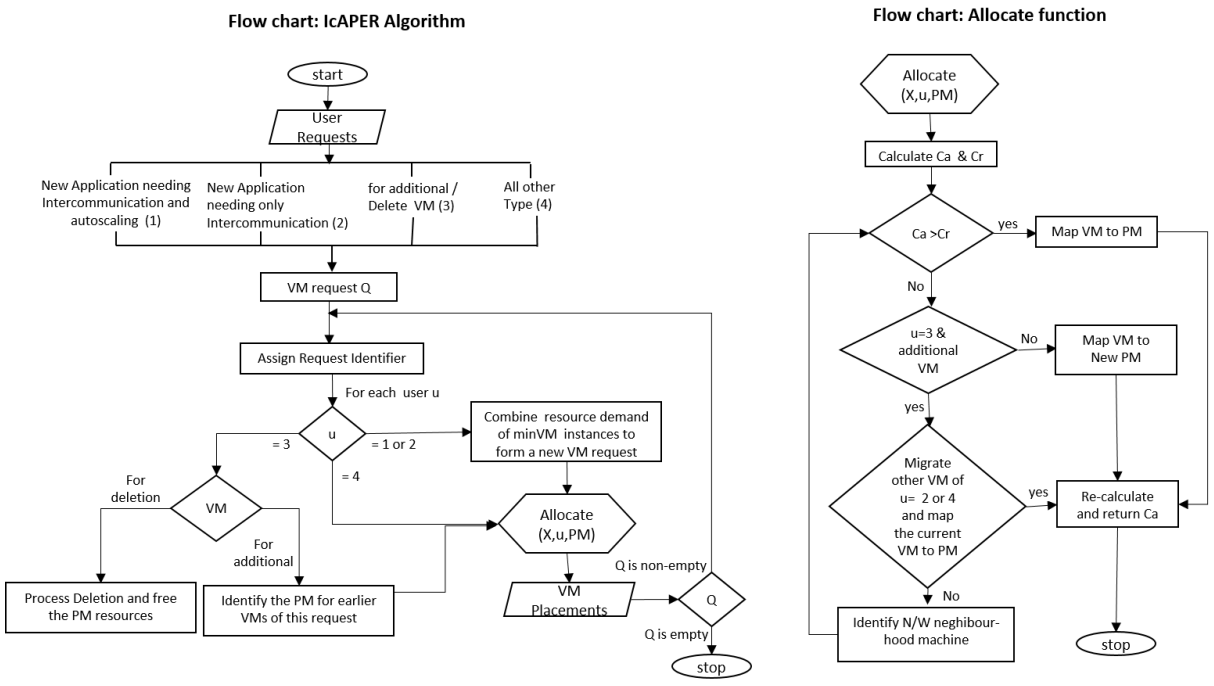

FIG. 4.1. *Flow chart of the proposed Algorithm*

- fogCDC has enough capacity to serve all the requests.
- VM request can be initiated by user or cloud service. In case of elasticity, cloud service monitor shall initiate the event associated with increase and or decrease of VMs.
- Without loss of generality, we assume that by minimizing cross-traffics, overall IoT application performance will be improved.

In this paper, VM placement patterns are formulated in two steps. During the first step initial VMP is achieved subject to constraints Eqs. 3.4, 3.5, 3.8 and Eq. 3.9. Fixed placement pattern that includes network neighborhood machines, is adopted for the additional VMs of request type r^e_{comu} as the final step. We symbolize C_t for total (original) capacity and C_u for current utilization of active PM which is in consideration for placement, while resource demand of VM that needs to be placed, is defined by C_r. Note that all these parameters are defined in terms of resource vectors (number of CPUs, memory size, storage space, bandwidth speed etc.).

The Inter-communication Aware placement for tasks of an Elastic Request (IcAPER) algorithm, after extracting the VM details from the individual request, assemble the VMs according to various constraints listed in Section 3, and sends them to *Allocate* function along with candidate PM. *Allocate* function performs the actual allocation of VMs to given PM. Upon non-availability of enough resources in the PM, it initiates new PM and places the VMs. Flow chat of IcAPER algorithm and Allocate sub-function is presented in Fig. 4.1(a). The detailed description of the same follows.

4.1. IcAPER Algorithm Description. As a first step IcAPER algorithm collects individual requests, both from cloud service and users onto a batch B. Additional VM of elastic requests emanating from cloud services are also queued to the same batch. Note that each request is defined by four-tuple $\{u, VM_{type}, VM_{min}, VM_{max}\}$. Components of this four-tuple are already explained in Section 3. However if the request is emanated from cloud service, then combination of information indicating whether the request is to scale-up or scale-down the VM, and a unique request identifier through which the four-tuples can be extracted, defines the request.

At fixed time interval, the m requests are differentiated as fresh and additional request. After extracting VM details from VM_{type} of each request in m, they are segregated as r^e_{comu},r_{comu} and r. In adherence to

Algorithm 1 IcAPER Algorithm

Require : request, active number of PM n
Result : tasks placement * Refer Sect. 3 for description of symbols *\
1 fetch user requests m
2 **while** m != NULL **do**
3 $r_k \leftarrow m.k$
4 **for** k = 1 to α
5 $C_r = X.R^d$
6 **if** r_k for deletion
7 process scale-down of VM
8 **else if** r_k for addition
9 *Allocate* $(X, r_k, p_n^{'})$ * adherence to fixed-PM constraint *\
10 **else if** $u \in r^e_{comu}$ or $u \in r_{comu}$
11 $X \leftarrow VM_{min}.R^d_v(VM_{type})$ * adherence to same-PM constraint*\
12 $Cu = Allocate(X, u, P_n)$
13 **else if** $u \in r$
14 $X \leftarrow N.R^d_v(VM_{type})$
15 $Cu = Allocate(X, u, P_n)$
16 **end for**
17 $C_a \leftarrow (C_t - C_u)$
18 **end while**

same-PM constraint of the proposed IcAPER, the demand resource vectors of all the tasks, given by N, of new r^e_{comu} request and new r_{comu} are combined individually to form a new task and considered as single placement [line 10]. If the request is for scale-up or scale-down of VM from an existing r^e_{comu} type requests, then PM details are extracted from the request identifier and processed accordingly [lines 5-9]so as to adhere to fixed-Pm constraints. After the formulation of VMs, sub-function *Allocate* is called with the candidate PM [lines 12,15]. Before addressing the next request available capacity of the PM is re-calculated. [line 17]. Description of the *Allocate* function is presented next.

Algorithm 2 Allocate Sub-function

Input : X (VM to be placed), request r_k, p_n (PM to be used for mapping)
return : utilization of PM * Refer Sect. 3 for description of symbols *\
1 extract u from r_k
2 $C_a \leftarrow (C_t - C_u)$ and $C_r = X.R^d$
3 **if** $u \in r^e_{commu}$ and X is an additional VM
4 **if** $C_a > C_r$) map X onto p_n
5 **else if** $(p_n \exists X^{'}.R^d \in r_{comm} \geq C_r)$
6 migrate $X^{'}$ from p_n using FirstFit
7 map X onto p_n
8 **else** map X onto network neighborhood machine by repeating step 4-8
9 **else if** $(C_a > C_r)$ map X onto p_n
10 **else** map X onto new PM
11 **return** C_u

4.1.1. Allocate function description. For Allocate function, request type along with the VM to be placed and candidate PM are given as input. After calculating available C_a and extracting resource demand vector C_r of current VM in consideration for placement [line 3] function proceeds to map the VM onto candidate PM given as the input. If the request is for an additional VM of r^e_{comu}, on availability of enough capacity, the request C_r is directly mapped on to the PM [line 5] else, migrate suitable VM or combination of VMs belonging to

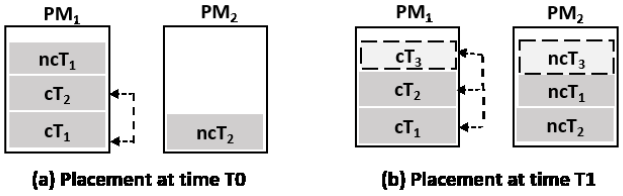

FIG. 4.2. *Placement of sample Tasks using IcAPER*

TABLE 5.1
Host configuration used in simulation

Core	RAM	VMM	Storage	MIPS
16	64 GB	Xen	1 TB	100,000

r from that PM using First Fit algorithm and map the additional VM [lines 6-8]. Upon unsuccessful placement suitable PM from the network neighborhood is identified and repeats steps 6-8 until successful placement happens. If the request is new one (r^e_{comu},r^e_{comu} and r), then on availability of enough resources, we directly map the VM onto the given PM, failing which new PM is initiated and mapping of VM is done[lines 10-11]. Finally the current utilization of the PM used for the latest placement process is returned to the Algorithm 1.

4.2. Cost Analysis of IcAPER Algorithm. Number of VM requests m in the batch queue will be a modest constant. For example, if scan interval of a batch is 5 minutes and on an average 50 VM requests get queued up, then Algorithm 1 needs to scan [lines 3-16] these 50 requests only and hence the cost is *O(m)*. Cost of sorting the segregated batch B is *O(m log m)* [line 17, 18]. Since *eList* and *neList* lists are of size *O(m/2)*, the following loop[lines 19 - 28] will also execute *O(m)* times, implying that *Allocate*, will also be called *O(m)* times. The inner loop having break [line 26] will execute once sufficient non-elastic VMs are placed matching to that of an elastic VM. As the number of VMs in a PM is bounded by a constant, the cost of inner loop is *O(1)*. Last loop [line 29-31] that places the leftover VMs also has a cost of *O(m)*. Let n be the number of candidate PM in *Allocate* sub-function, then packing the VMs onto PM, results in cost of *O(n)*. Hence *O(m log m) + O(n)* is the total cost of IcAPER, which is asymptotic to FFD and FCFS algorithms.

4.3. IcAPER for sample VMs. Let us look at the changes that the balanced FCFS VM placement scheme described in c*f. Sect. 1*, for the sample VMs as depicted in Fig. 1.2 (a) goes through during initial VMP (T_0) and subsequently T_1, according to IcAPER. IcAPER scheme for these VMs is depicted in Fig. 4.2. As per IcAPER, at time T_0 the initial requests of elastic user are placed together on to the same PM as shown in Fig. 4.2(a), whereas at time T_1, the VM placements resembles to Fig. 4.2(b), in which ncT_1 is migrated to second PM so that additional request cT_2 can be placed on to the first PM itself. This clearly reduces the cross PM communications and hence the network traffic resulting in conservation of energy. Here the trade off is between the cost incurred to migrate ncT_1 and expected application performance. We will explain the experimental setup and the analysis of results in the subsequent sections.

5. Implementation. Efficiency of any VMP algorithm, needs to be evaluated in a large-scale cloud platform. However, it is difficult to conduct the repeatable VMP experiments in a real production cloud platform. Hence, the proposed IcAPER algorithm is simulated using *CloudSimPlus* [30] simulator. The algorithms were implemented as an extension to SimpleVmAllocationPolicy, which determines how VMs are assigned to the host.

Since our focus in this work is to improve VMP that is common to both fog and cloud resources, we carried out the experiments using *CloudSimPlus* simulator having Xen as the virtual machine monitor (hypervisor). The proposed, Inter-communication Aware Elastic Request Placement (IcAPER) algorithm, written in Java, is tested on a Dell workstation with Intel Xeon 3.30Ghz and 16Gb memory, having x64 architecture. The simulation is performed over a datacenter made up of twenty homogeneous physical machines whose configuration is presented in Table 5.1. VM configuration of popular service provider Amazon EC2 [31] is adopted and the same is shown

TABLE 5.2
Virtual Machine configuration used in simulation

resource	Micro	Small	Medium	Large	Xlarge
vCPU	1	1	2	2	4
RAM (GB)	1	2	4	8	16

TABLE 5.3
Characteristics of requests in a batch

Characteristic	Number
Number of Request	40
Requests type r^e_{comu}	12
Requests type r_{comu}	10
Requests type r	18
New VM requests by type r^e_{comu}	22
New VM requests by type r_{comu}	32
New VM requests by type r	45
Requests for additional VM	7
Requests for delete VM	3

in Table 5.2.

Since we are addressing the application elasticity along with inter-communication of its tasks in this work, for each experiment, we created three bunches of randomly generated requests in the ratio of 2:1:1 for time slot T_1, T_2 and T_3.The total resource requirement of these requests is approximately equal to the capacity of our datacenter. Note that each request is represented by a four-tuple as explained in Section 3. Characteristic of requests for single experiment is presented in Table 5.3. Column 8 is valid only for the request belonging to time T_1 onwards whereas, column 9 is valid for the request belonging to time T_2 onwards only. As explained in Section 1, each VM is attached with a cloudlet whose input file size is 1000 bytes. We have compared the proposed IcAPER algorithm with FCFS, Random and FFD algorithms for an experiment. To avoid transient anomalies we conducted ten such experiments and collected the statistics for evaluation parameters: (a) resource utilization, (b) resource fragmentation, (c) reduced VM migrations and (d) Number of requests having intercommunicating tasks placed on to same PM.

The results presented in Figs. 5.1 – 5.4 demonstrate that IcAPER algorithm is preferred one when compared with other algorithms which are considered for the comparison. From Fig. 5.1, we can find out that resource utilization by IcAPER increase when more requests are served than other algorithms. When individual PMs may have enough resources in one dimension (CPU, memory etc.) but not in all dimensions and hence requests cannot be placed on to it, resulting in resource fragmentation. The increase in resource fragmentation, over a period of time, implies scope for improvement in fogCDC management. As shown in Fig. 5.2 IcAPER algorithm perform best when compared with other algorithms. This work proposes to host all the tasks of r^e_{comu} and r_{comu} type requests, along with its additional(elastic) tasks on the same PM. When compared all the algorithm for this parameter, as depicted in Fig. 5.3, IcAPER outperforms all other algorithms.

Next evaluation parameter is potential migration needed in all algorithms that are in our consideration to ensure nil network latency. It is to be noted that as by design, IcAPER has to place maximum possible tasks on the same PM at T_1. However, from T_2 onwards all algorithm makes placement decision based on the availability of resource on PM and individual algorithmic constraints. For example, IcAPER forces the migration of tasks belongs to r_{comu} and r type request, so that co-allocation constraint for elastic needs of r^e_{comu} type requests can be adhered. As shown in Figure 5.4 IcAPER consistently outperforms other algorithms over multiple time slot. It is also to be noted that the application performance and cost of migration are directly proportional to the number of additional tasks requested by elastic request when fogCDC is deployed with IcAPER algorithm.

6. Conclusions. In this paper, after formulating Inter-communication Aware Elastic Request Placement (IcAPER) problem, pertaining to a fog-cloud environment, authors propose an efficient IcAPER algorithm to improve the resource utilization of fog-cloud datacenter. IcAPER considers individual request encompassing one or more VMs as a whole for placement along, with the life-time of the request rather than individual VMs. This

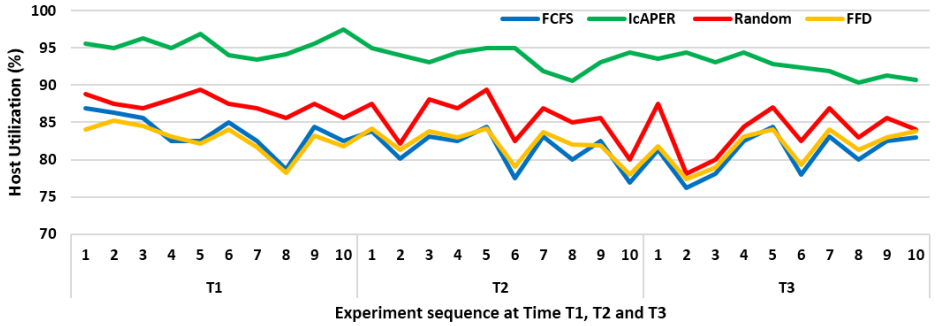

FIG. 5.1. *Physical Machine Utilization*

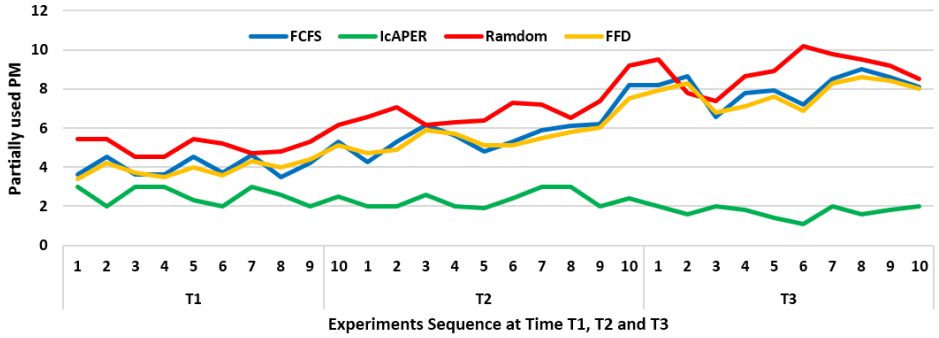

FIG. 5.2. *Physical Machine Fragmentation*

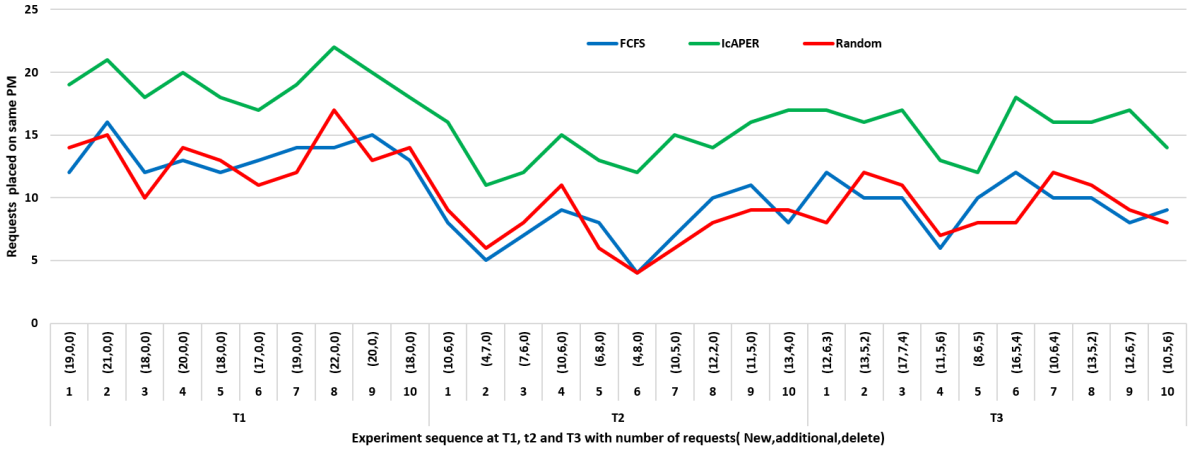

FIG. 5.3. *Number requests needing intercommunicating tasks mapped to same PM*

proposed algorithm places all the tasks of requests needing intercommunication and elasticity, on to same PM and/or on to network neighborhood PMs by implementing same-PM, fixed-PM and balanced-PM constraints for task placements. These constraints are adhered by maintaining the identity of tasks with reference to its application across fogCDC resulting in improved performance of fog application.

Performance of IcAPER is compared with FFD, FCFS and Random, well-known placement algorithms for metrics resource utilization, resource fragmentation, migrations needed to maintain minimal network latency and number of requests having intercommunicating tasks placed on to same PM for evaluation. Based on average taken out of ten experiments, the simulation results shows proposed technique is attractive compared

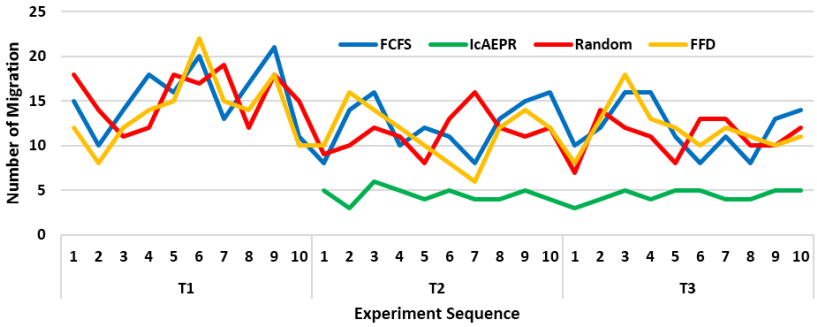

FIG. 5.4. *Potential Migration needed to minimize the latency*

to other algorithms that are in authors consideration. Studying the performance impact of IcAPER algorithm to a real time video surveillance application using distributed camera will be of future interest. Pan-tilt-zoom (PTZ) parameters of multiple cameras requires low-latency communication between the cameras for which we can use the fog resources. Whereas, for processing the captured video frames and for long-term processing (analysis) of control strategy to obtain the optimal PTZ parameter cloud resource can be used.

REFERENCES

[1] S.SING, A.NAYYAR, R. KUMAR, AND A.SHARMA, *Fog computing: from architecture to edge computing and big data processing* , in The Journal of Supercomputing, 2018, PP. 1-36

[2] S. YI, Z. HAO, Z. QIN, AND Q. LI, *Fog computing: Platform and applications*, in Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). IEEE, 2015, pp. 7378

[3] M. SATYANARAYANAN, Z. CHEN, K. HA, W. HU, W. RICHTER, AND P. PILLAI, *Cloudlets: at the leading edge of mobile-cloud convergence*, in IEEE International Conference on Mobile Computing, Applications and Services (MobiCASE), 2014.

[4] M. SATYANARAYANAN, P. BAHL, R. CACERES, AND N. DAVIES , *The case for vm-based cloudlets in mobile computing* , in Pervasive Computing, 2009.

[5] D. F. WILLIS, A. DASGUPTA, AND S. BANERJEE, *Paradrop: a multi-tenant platform for dynamically installed third party services on home gateways*, in ACM SIGCOMM workshop on Distributed cloud computing, 2014.

[6] J. WANG, J. PAN, AND FLAVIO ESPOSITO,*Elastic Urban Video Surveillance System Using Edge Computing*, In Proceedings of SmartIoT17, 2017. https://doi.org/10.1145/3132479.3132490.

[7] N. MKITALO, F. NOCERA, M. MONGIELLO, AND S. BISTARELLI, *Architecting the Web of Things for the fog computing era*, doi: 10.1049/iet-sen.2017.0350

[8] M. ANAWAR, S. WANG, M. ZIA, A. JADOON, U. AKRAM,AND S. RAZA, *Fog Computing: An Overview of Big IoT Data Analytics*, 2018. ID 7157192 https://doi.org/10.1155/2018/7157192.

[9] . WANG, H. CHEN, AND X. CHEN, *An availability-aware virtual machine placement approach for dynamic scaling of cloud applications*, in Proc. IEEE Ninth Inter. Conf. Ubiquitous Intelligence & Computing and Autonomic & Trusted Computing (UIC/ATC), pp.509-516, 2012.

[10] X. MENG, V. PAPPAS, L. ZHANG, *Improving the scalability of data center networks with traffic-aware virtual machine placement*, in Proc IEEE INFOCOM2010, pp. 1-9, 2010.

[11] F. BONOMI, R. MILITO, P. NATARAJAN, J. ZHU. 2014,*Fog computing: a platform for internet of things and analytics*, In: Big Data and Internet of Things: A Roadmap for Smart Environments Springer; 169-186.

[12] M. R. GARREY, R. L. GRAHAM, AND D. S. JOHNSON, *Resource constrained scheduling as generalized bin packing*, J. Combinatorial Theory, Series A, vol. 21, no. 3, pp.257-298, 1976.

[13] R. BUYYA, S. K. GAR, AND R. N. CALHEIROS, *SLA-oriented resource provisioning for cloud computing: Challenges, architecture and solutions*, in Proc. Inter. Conf. Cloud. Ser. Comput., pp. 1-10, 2011.

[14] A. BHADANI AND S. CHAUDHARY, *Performance evaluation of web servers using central load balancing policy over virtual machines on cloud*, in Proc. Third Annual ACM Conference, ACM, 2010, Article no.16.

[15] V. V. VAZIRANI, *Approximation algorithms*, Springer, 2001.

[16] R. PANIGRAHY, K. TALWARE, LINCOLN UYEDA, AND U. WIDEDER, *Heuristics for Vector Bin Packing*, Microsoft Research, 2011.

[17] X. ZHENG AND Y. CAI, *Dynamic virtual machine placement for cloud computing environments*, in 2014 43rd International Conference on Parallel Processing Workshops, Sept 2014, pp. 121128.

[18] A. SINGH, M. KORUPOLU, AND D. MOHAPATRA, *Server-storage virtualization: Integration and load balancing in data centers*, in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, ser. SC 08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 53:153:12. [Online]. Available: http://dl.acm.org/ citation.cfm?id= 1413370.1413424

[19] G. Khanna, K. Beaty, G. Kar, and A. Kochut, *Application performance management in virtualized server environments*, in Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, April 2006, pp. 373381. [Online]. Available: http://dx.doi.org/10.1109/ NOMS.2006.1687567

[20] B. Speitkamp and M. Bichler, *A mathematical programming approach for server consolidation problems in virtualized data centers*, Services Computing, IEEE Transactions on, vol. 3, no. 4, pp. 266278, Oct 2010.

[21] M. Rahman and P. Graham, *Compatibility-based static VM placement minimizing interference*, J. Netw. and Comp. Appl.,vol. 84, pp. 68-81, 2017.

[22] E. Feller, C. Morin, A. Esnault, *A case for fully decentralized dynamic vm consolidation in clouds*, in: Cloud Computing Technology and Science (Cloud Com), 2012 IEEE 4th International Conference on, IEEE, 2012, pp. 2633.

[23] L. Cui, F. Tso, P. Pezaros, W. Jia, and W. Zhao, *PLAN: Joint policy- and network-aware VM management for cloud data centers*, in IEEE Trans. Parallel and Distributed Systems, vol. 28, no. 4, pp. 1163-1175, 2017.

[24] A. Beloglazov, J. Abawajy, R. Buyya, *Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing*, Future Generation Computer Systems 28 (5) (2012) 755768.

[25] Q. Zheng, R. Li, X. Li, N. Shah, J. Zhang, F. Tian, K.-M. Chao, J. Li, Virtual machine consolidated placement based on multi-objective biogeography-based optimization, inFuture Generation Computer Systems 54 (2016) 95122.

[26] J. Shi, F. Dong, J. Zhang, J. Luo, D. Ding, *Two-phase online virtual machine placement in heterogeneous cloud data center*, in: Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on, IEEE, 2015, pp. 13691374.

[27] F. Lpez-Pires, B. Barn, L. Bentez, S. Zalimben, and A. Amarilla, *Virtual machine placement for elastic infrastructures in overbooked cloud computing datacenters under uncertainty*, Future Gen. Comp. Sys., Vol. 79, Part 3, pp. 830-848, Feb 2018.

[28] S.K.Mishra, P. Deepka, B. Sahoo, P.P. Jayaraman, S. jung, A.Y. Zomaya, R. Ranjan, *Energy-efficient VM-Placement in Cloud Data Center*,Sustainable Computing : Informatics and Systems, (2018) https::/ doi.org/10.1016/j.suscom.2018.01.002.

[29] B. Heidelberg, *Bin-packing, in Combinatorial Optimization, ser. Algorithms and Combinatorics 21. Springer, 2006, vol. 21, pp. 426-441.*

[30] http:// www.cloudsimplus.org

[31] https://aws.amazon.com/ec2/instance-types/

Edited by: Anand Nayyar
Received: Mar 15, 2019
Accepted: Apr 4, 2019