Computers and Electrical Engineering 68 (2018) 44–61

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Scalable and direct vector bin-packing heuristic based on residual resource ratios for virtual machine placement in cloud data centers☆

Saikishor Jangiti, Shankar Sriram. V.S.*

Center for Information Super Highway (CISH), School of Computing, SASTRA Deemed University, Thanjavur, Tamilnadu, India

ARTICLE INFO

Keywords:
Cloud computing
Infrastructure as a service
VM placement
Server consolidation
Scalability
Vector bin-packing
Resource provisioning

ABSTRACT

Virtual Machine (VM) placement consolidates VMs into a minimum number of Physical Machines (PMs), which can be viewed as a Vector Bin-Packing (VBP) problem. Recent literature reveals the significance of first-fit-decreasing variants in solving VBP problems, however they suffer from reduced packing efficiency and delayed packing speed. This paper presents VM NeAR (VM Nearest and Available to Residual resource ratios of PM), a novel heuristic method to address the above said challenges in VBP. Further, we have developed Bulk-Bin-Packing based VM Placement (BBPVP) and Multi-Capacity Bulk VM Placement (MCBVP) as a solution for VBP. The simulation results on real-time Amazon EC2 dataset and synthetic datasets obtained from CISH, SASTRA shows that VM NeAR based MCVBP achieves about 1.6% reduction in the number of PMs and possess a packing speed which was found to be 24 times faster than exisiting state-of-the-art VBP heuristics.

1. Introduction

Rapid growth in the Internet and Information Technology (IT) services has driven the emergence of cloud computing technologies as an alternative to the traditional computing infrastructures. Infrastructure as a Service (IaaS), a cloud service delivery model provides on-demand access to remote computing resources available at various Cloud Data Centers (CDCs) across the globe. The end users hire Virtual Machine (VM) instances from popular Cloud Service Providers (CSPs) (e.g., Amazon EC2,[1] Google Compute Engine,[2] and Microsoft Azure[3]) and run applications on their own. In general, the CSPs offer reliable and robust compute and storage resources based on the users' Quality of Service (QoS) requirements for selected period. According to Gartner, "*The cloud market is growing by 40 percent, and a corporate without cloud will be rare by 2020*". Due to the rapid growth in the adoption of cloud infrastructure and services, the CDCs are expected to hold beyond 10% of the global energy utilisation [1]. Energy-efficient management of CDCs is an active research topic which aims at reducing the operational expenses, thereby making CSPs to sustain in the competitive cloud marketplace. Further, it contributes towards minimizing the carbon footprints and its adverse environmental impacts. A major component of CDC's power consumption is by Physical Machines (PMs), and even when idle, PMs consume 60% of the full load power. From the **United States data center energy usage report-2016**, it was evident that the average resource utilization in

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. L. Bittencourt.

* Corresponding author.
E-mail address: sriram@it.sastra.edu (S. Sriram. V.S.).

[1] https://aws.amazon.com/ec2

[2] https://cloud.google.com/compute

[3] https://azure.microsoft.com

https://doi.org/10.1016/j.compeleceng.2018.03.029
Received 1 May 2017; Received in revised form 19 March 2018; Accepted 19 March 2018
Available online 04 April 2018
0045-7906/

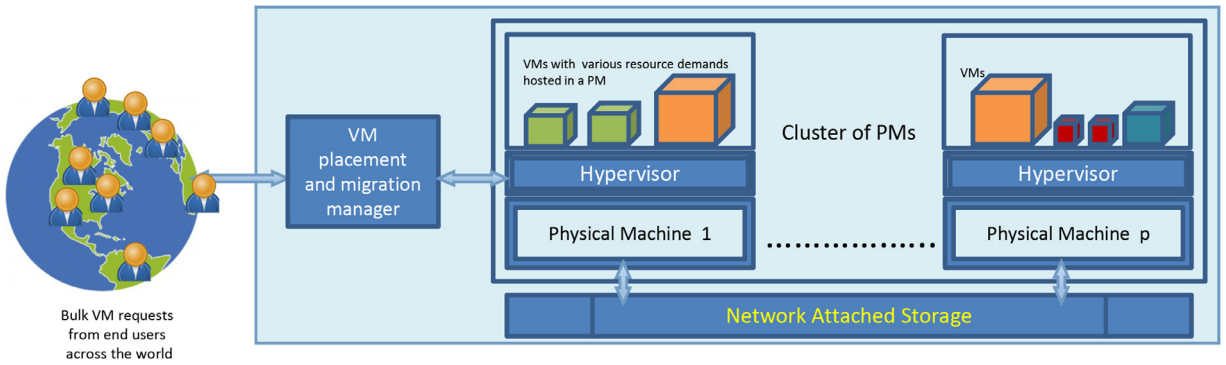

Fig. 1. Architecture model of a cloud data center.

CDCs was 14% and through VM consolidation, CDCs could save 520 billion kWh by 2020, thereby insists the importance of virtualization technologies in improving energy efficiency. In general, virtualization technologies enables sharing of a physical resource by multiple software components to form secure and isolated IT resources in the form of VMs. Fig. 1 represents the generic architecture of a cloud data center. The CDC receives bulk VM requests of different resource requirements from the customers across the globe. The autonomic cloud manager assigns VM requests into fewer PMs to minimize the operational expenses, which is a classical application of multi-capacity or Vector Bin-Packing (VBP).

This work is one another attempt to solve the VM Placement as a vector bin packing problem, since the economic and environmental benefits of this study places it as an active research challenge. The efficiency of the VBP heuristics is determined by two factors, namely packing efficiency and speed to decide whether to use a bin-packing heuristic or not. The state-of-the-art meta-heuristics and greedy heuristics for solving VBP problem either focus on the packing efficiency or speed. Further, the speed and packing efficiency of the existing approaches have a negative correlation between them, scalability is another prime quality metric to be considered along with the speed and packing efficiency for VM placement. In this way, we propose the first direct VBP heuristic, multi-capacity bulk VM placement with improved speed, better packing efficiency and high scalability for energy efficient VM placement.

OpenStack[4] and Eucalyptus[5] are the popular tools used to build public, private and hybrid clouds. The automatic management of these tools use bin-packing heuristics for VM initial placement. From the literature, it was evident that greedy heuristics outperform the meta-heuristic approaches like genetic algorithm and local search in 79.08% instances [2–4]. Variants of First Fit Decreasing (FFD), a one-dimensional bin-packing greedy heuristic are widely used to solve VBP due to the lack of direct VBP heuristic. Panigrahy et al. [5] studied the FFD variants like FFDProd, FFDSum, FFDAvgSum, and FFDExpSum & designed new geometric heuristics, namely Norm-Based Greedy (NBG) and Dot Product (DP) for VM Placement which dominates the other heuristics in terms of solution quality. Microsoft cloud platform system center (virtual machine manager) also uses DP and NBG as an internal function for VM consolidation [6]. Wei Zhu et al. [7] recently proposed a DP based vector bin packing algorithm named Heuristic Virtual Resource Allocation Algorithm (HVRAA). To summarize, we have identified two significant limitations in DP and NBG i.e., *delayed packing speed* and *reduced packing efficiency* due to *continuous sorting* and *collision in the vector-to-scalar conversions* respectively. *Continuous sorting* improves the solution quality; DP and NBG update the scalar weights of items according to the changes in PM residual capacities after placing each VM. This update is followed by a sort to determine the largest item. These repeated sortings will slow down the placement process and also raise the scalability issues when the number of items is large. *Collision in the vector-to-scalar conversions* is that the conversion of two or more completely different vectors may result in the same scalar value. This collision leads to a wrong choice of VMs for placement and has a high impact on the solution quality. Hence, this paper presents a novel VM choosing heuristic called (VM NeAR) which employs a new bin and item representation vector called Resource Ratio Vector (RRV) to address the above-said limitations.

The CSPs offer some standard VM instances, and there will be repeated (more than one) requests of the same VM instance type from customers across the globe. It is observed that in case of repeated requests, more than one PM end up with the same placement. Bulk-Bin-Packing (BBP) is proposed from this observation and is used to speed up the placement process by identifying how many more PMs can be filled with the same kind of VMs.

In addition to VM NeAR, we have also developed:

(i) Bulk-Bin-Packing based VM Placement (BBPVP), a rapid and scalable bin-packing of repeated VM requests with same resource ratios
(ii) Multi-Capacity Bulk VM Placement (MCBVP), the first direct VBP heuristic

VM placement problem has two subproblems: **VM initial placement** - performed once, the first time when a VM request comes.

[4] https://www.openstack.org/
[5] http://www.eucalyptus.com/

Table 1
A review of VM initial placement.

Authors	Description	VM initial placement focused on	Approach	Max. VM requests placed	VM requests dataset
David Wilcox et al. [3]	**RGGA** -To minimise the number of PMs through multi-capacity Bin-packing	• CPU Cores • RAM	Reordering and grouping genetic algorithm	4500	HARD data sets, item weights 20,000–35,000 and bin capacities of 100,000 are not realistic for VM placement problem
Jiang et al. [8]	Power efficient VM placement by minimising total PMs	• CPU Cores • RAM • Disk storage • Network	Genetic algorithm	370	Synthetic
Khosravi et al. [9]	**ECE** reduces the total power consumption and carbon footprint by minimising total PMs	• Carbon footprint • Power Usage Effectiveness	Best fit decreasing (derivative)	2000	Real Dataset (Amazon EC2 instance types)
Yusen Li et al. [10]	**Hybrid First Fit packing** minimises the PMs used over time	• After classification, large and small items are packed separately	Online bin-packing	Theoretically proved that the competitive ratio of the proposed Hybrid First Fit packing is better than other heuristics.	
Canali and Lancellotti [11]	**Class-based placement,** a scalable VM placement technique minimises the PMs used	• CPU usage	Bin-packing	1200	Real world application CPU usage traces
Hallawi et al. [12]	**COFFGA** minimises the total number of PMs	• CPU Cores • RAM	Combinatorial ordering genetic algorithm	4500	HARD data sets, item weights 20,000–35,000 and bin capacities of 100,000 are not realistic for VM placement problem
Wei Zhu et al. [7]	**HVRAA** allocates the given VMs into a minimum number of PMs	• The weighted dot product of resource capacities	Vector packing	100	Synthetic
Rahman and Graham [13]	**CSVP,** Compatibility based Static VM Placement	• Resource Criticality in a PM	Bin-packing	1000	Google traces

Table 2
A review of VM reallocation.

Authors	Description	VM reallocation focused on	Approach	Max. VM requests placed	VM utilizations dataset
Hermenier et al. [14]	**Bin repacking** - PM consolidation by scheduling VM migrations based on new resource and placement requirements	• Minimizing average migration completion time	Bin-packing with constraint programming	10,000	Real Dataset (Amazon EC2 instance types)
Chen et al. [15]	**Effective VM Resizing** - PM consolidation by formulating as a stochastic bin- packing problem	• Server capacity • Allowed server overflow probability	Stochastic bin-packing	40,400	Real Datacenter trace
Masson et al. [4]	To maximise the resource usage in PMs	• Resource capacity constraints	Iterated local search	50,000	2012 ROADEF Challenge data
Gao et al. [16]	**VMPACS** does consolidation to minimise the number of PMs	• Minimize resource wastage	Ant colony optimisation	2000	Synthetic Dataset
Shojafar et al. [17]	**Joint dynamic Lyapunov based scheduler**	• CPU Usage • Network	Modified best fit decreasing	10	Real world application CPU usage traces
Canali et al. [18]	**Joint computation-plus-communication model**	• CPU Usage • Network	Bin-packing	140	
Canali et al. [19]	Minimizes computing, data transmission, and migration energy costs in CDCs	• CPU Usage • Network	Optimization of a constraint satisfaction problem	500	

Table 3
A review of one-dimensional Bin-packing heuristics.

Greedy heuristic	Packing Strategy Description *(Items are supplied in descending order)*	Time Complexity
Next-fit decreasing	Place the item in the current bin if fits, otherwise open a new bin and place.	$O(n)$
First-fit decreasing	Place the item in the first bin it fits, among the opened bins. Open a new bin if not.	$O(n \log n)$
Best-fit decreasing	Place the item in the bin with the least gap after item placement among all the opened bins. Open a new bin if the item does not fit in any bin.	$O(n \log n)$
Max-Min packing [21]	Open a new bin and place as many large items that fit from one end of the sorted item list, and then place as many small items that fit from the other end.	$O(n)$

VM reallocation - triggered several times, based on the optimization criteria. Several methods available in the existing literature for VM initial placement are reported in Table 1. The maximum VM requests supplied for experimentation of VM Initial placement is 4500. The common experimentation is through simulation, synthetic datasets, and real-time Amazon EC2 dataset. The VM reallocation methods which are based on probabilistic models and meta-heuristic techniques are discussed in Table 2.

The remainder of the article is organised as follows: Section 2 presents the preliminaries about the bin-packing, VBP and various notations used. Section 3 presents the proposed methods and VM placement heuristics, namely BBP, BBPVP, VM NeAR and MCBVP. Section 4 portrays the statistical analysis and comparisons, and Section 5 summarizes and draws the attention to future work.

2. Preliminaries

2.1. Bin-Packing Problem (BPP)

BPP is the standard one-dimensional packing problem. The goal of BPP is to assign the given set of n items to minimum number of equal capacity bins. BPP, a one-dimensional problem deals with only one capacity constraint. It is known to be strongly NP-hard. Based on the items' arrival, BPP can be broadly classified into two: (i) online and (ii) offline. Online BPP is where the items arrive one at a time, while in offline BPP, all the items arrive upfront. Due to the lack of standard algorithms, the various greedy heuristics approaches such as next-fit, best-fit, and their variants are used to solve the BPP. The most widely used greedy heuristics approaches are namely, Next-Fit Decreasing (NFD), FFD and Best-Fit Decreasing (BFD) [20] are discussed in Table 3. For NFD, BFD and FFD, items are supplied in decreasing order to minimize the required bins by placing smaller items in the gaps of already used bins instead of fresh bins. The two important factors that decide the quality of a packing heuristic is its packing efficiency (minimum bins) and speed. Next-fit is the fastest among the BPP heuristics, but its packing efficiency is low.

2.2. Vector Bin-Packing (VBP)

VBP, also known as the multi-capacity bin-packing problem, is a generalisation of BPP, where an item is represented by a vector (of sizes in d dimensions) and can deal with the problems like VM Placement (multiple resource capacity constraints). The goal of VBP is the same as BPP that pack the given items into as few (multi-capacity) bins as possible. VBP is strongly NP-hard as well as APX-hard (when $d \geq 2$). At present, there is no direct heuristic (or) approximate algorithm to solve VBP. As shown in Fig. 2, VBP involves multiple resource capacity constraints.

2.3. Resource Ratio Vector (RRV)

RRV, a vector of length $d-1$ denoted by r is obtained from either PM or VM with size representation vectors of length d. Eq. (1) shows the computation of RRV (r).

$$r = \left(\frac{resReq_2}{resReq_1}, \frac{resReq_3}{resReq_1}, \frac{resReq_4}{resReq_1}, ..., \frac{resReq_d}{resReq_1} \right), \tag{1}$$

where $resReq_i$ is the i^{th} resource requirement of a VM request.

The vector-to-scalar conversion method, called as dot-product used in existing VBP heuristics, may assign the same scalar value, even for VMs with opposite resource requirements in two (or more) dimensions. Consider 2 VM requests, <2 *CPU Cores*, 1 *GB RAM* $>$ and <1 *CPU Core*, 2 *GB RAM*$>$, yield the same scalar value $(2\times1) = (1\times2) = 2$ by dot-product. However, their resource ratios completely differ $\left(\frac{1}{2} \neq \frac{2}{1}\right)$. Hence, resource ratio is a better VM representation to avoid collision while selecting the next VM to be placed in PM residual capacities, which is highlighted in Fig. 2. If all the VMs have same resource ratios, then the VM placement based on one capacity constraint will fill the other resources accordingly in proportion. VBP is reduced to BPP (one-dimensional bin packing) in this case. Dealing with VMs that have different resource ratios formulates a typical VBP.

2.4. Repeated items

According to the pigeon-hole principle, when the number of items (n) is more than the number of item types (m), atleast one item

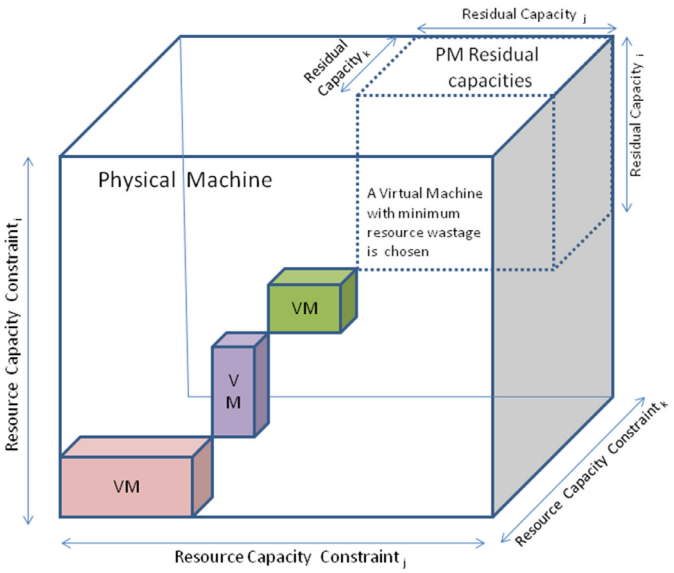

Fig. 2. Vector bin-packing of VM requests.

will be repeated (supplied more than once), which is termed as a *repeated item* in this paper.

A summary of notations used in this paper is presented in Table 4.

3. Problem formulation

In this research of scalable multi-capacity VM placement, a large number of repeated VM requests were considered for VM placement. This is formulated as a VBP problem with an objective to minimise the number of PMs used. Generally, VBP is an NP-Hard problem, and there exists no deterministic algorithm to achieve an optimal solution.

The following assumptions were made while formulating the VBP problem:

1) CSP offers IaaS, a set of standard VM instance types with fixed resource offerings.
2) The end users choose an appropriate VM instance type for their application and CSP is not aware of the application's power profiling, varying resource demands and job completion time.
3) CSP provides the promised resources throughout the reservation period.
4) Since many customers across the globe may choose the same kind of VMs, there will be multiple requests for same VM instance type.
5) Each VM request is to be assigned to at most one PM.

Table 4
Summary of Notations.

Notation	Meaning
$\boldsymbol{n}$	Total number of items (VMs supplied for placement)
$\boldsymbol{item_i}$	i^{th} item among n items
$\boldsymbol{m}$	Number of item types, $m \leq n$
$\boldsymbol{itemType_i}$	i^{th} item type among m types
$\boldsymbol{Count_i}$	Number of items of $itemType_i$ supplied; $\sum_{i=1}^{m} \boldsymbol{Count_i} = n$
$\boldsymbol{packed_i}$	Number of items of $itemType_i$ packed in the current bin-packing configuration
$\boldsymbol{remaining_i}$	Number of items of $itemType_i$ available more for packing
$\boldsymbol{r}$	Resource ratio vector
$\boldsymbol{d}$	Number of dimensions (resources)
$\boldsymbol{VM_{RequestList}\langle r,S,N\rangle}$	Bulk VM requests and their resource ratios
$\boldsymbol{S}$	Size of the VM in a resource ratio group (CPU Cores in VM)
$\boldsymbol{N}$	Number of repeated requests of a VM type
$\boldsymbol{R_{placement}}$	The VM group nearest to PM residual ratios, among the available groups
Bulk packing	Current packing configuration is repeated (packed in more than one bins)
Single packing	Current packing configuration cannot be repeated due to non-availability of one or more items used in the current packing configuration
item by item	Some of the existing packing heuristics packs item by item
bin by bin	Bulk-bin-packing packs a bin as much as possible and opens a new bin

6) PMs are homogenous.

3.1. Optimization formulation

In this research work, multiple resources like CPU Cores, RAM, Network Bandwidth and Disk Size of the PM as well as VMs were considered. The n VM requests (m types of repeated requests) are grouped based on *resource ratios* (r); m groups are available as $VM_{RequestList}\langle r, S, N\rangle$ for placement. N denotes the count of a repeated VM request type and S denotes CPU cores as size of the VM in the same resource ratio group. The assignment solution using b PMs is represented by a $b \times n$ placement matrix P, where $P_{ki} = 1$, if VM_i is assigned to PM_k and $P_{ki} = 0$, otherwise. Let $f(PM_k)$ be a function such that $f(PM_k) = 1$, if PM_k is loaded with at least one VM and $f(PM_k) = 0$ otherwise. The problem of assigning all VMs to the least number of PMs, are subjected to few constraints which are formulated as follows,

$$\text{minimise} \sum_{k=1}^{b} f(PM_k) \tag{2}$$

$$\text{subject to} \sum_{i=1}^{n} P_{ki}.\, R_{ji} \le PM_{R_j} \;\; \forall \;\; k \in \{1, 2, 3, ...,b\} \; and \; \forall \;\; j \in \{1, 2, 3, ...,d\} \tag{3}$$

$$\sum_{k=1}^{b} P_{ki} \le 1 \;\; \forall \; i \in \{1, 2, 3, ...,n\} \tag{4}$$

Eq. (2) defines the objective of the optimisation problem, i.e., to minimise the PMs used while satisfying the constraints in Eqs. (3) and (4). The constraint in Eq. (3) imposes a limit on maximum utilisation of each PM resource to support the assumption 3 in the problem formulation. The constraint in Eq. (4) ensures the assumption 5.

3.2. Bulk-Bin-Packing (BBP)

BBP is a scalable variant of bin-packing to pack repeated items. Repeated VM requests (items) exist in VM placement and many other real-world bin-packing applications. It is observed that in case of repeated requests, several PMs end with the same placement and the observation is used to speed up the placement process by identifying how many more PMs can be filled with the same kind of VMs. BBP can speed up packing by identifying bulk numbers using Eq. (5), and the idea of BBP is depicted in Fig. 3.

Definition 1. Bulk number: The number of bins that can be filled with same items as in the current bin-packing configuration.

$$\boldsymbol{BulkNumber} = \boldsymbol{minimum}\left(\frac{\boldsymbol{remaining_i}}{\boldsymbol{packed_i}}\right),\;\; 1 \le \boldsymbol{i} \le \boldsymbol{m} \;\&\; \boldsymbol{packed_i} \neq 0 \tag{5}$$

where $\boldsymbol{packed_i}$ & $\boldsymbol{remaining_i}$ denotes the number of $\boldsymbol{itemType_i}$ items packed in the current bin and remaining for packing respectively.

Definition 2. Bulk-bin-packing: Given $\boldsymbol{n}$ items of $\boldsymbol{m}$ different types in the form of ordered pairs ($\boldsymbol{item_i}$, $\boldsymbol{count_i}$), where $\boldsymbol{i} = 1..\boldsymbol{m}$ and $\sum_{i=1}^{m} \boldsymbol{count_i} = \boldsymbol{n}$. The items are to be packed into minimum bins of capacity $\boldsymbol{C}$. A bulk-bin-packing solution is in the form of ordered pairs {($\boldsymbol{BulkNumber}_1$, $\boldsymbol{binConfig}_1$), ($\boldsymbol{BulkNumber}_2$, $\boldsymbol{binConfig}_2$),…., ($\boldsymbol{BulkNumber_k}$, $\boldsymbol{binConfig_k}$)},where $\boldsymbol{binConfig}_{1\ldots k}$are $\boldsymbol{k}$different packing configurations and $\boldsymbol{k} \le 2\boldsymbol{m}$.

Theorem 1. Bulk-Bin-Packing (BBP) packs m types of items in not more than 2m steps, its computational complexity is $O(m)$.

There are two phases in BBP: Phase-I identify the next bin-packing configuration, and Phase-II will either do ***bulk packing*** or ***single packing*** . There will be a ***bulk packing*** with respect to every $\boldsymbol{itemType_i}$ which may be followed by a ***single packing*** if $\boldsymbol{itemType_i}$ is not completely packed by its corresponding ***bulk packing***.

Every $\boldsymbol{itemType_i}$ will be completely packed in a maximum of two iterations (their respective ***bulk packing*** and ***single packing***). Together, $\boldsymbol{m}$ item types will be packed in a maximum of $\boldsymbol{2m}$ steps and configurations (each step a packing configuration). Hence the time complexity is $\boldsymbol{O(m)}$, which is independent of the number of items, $\boldsymbol{n}$. Thus BBP with any packing strategy in Phase-I is scalable to large number of items, while $\boldsymbol{m}$ (item types) remains the same.

The BBP is a *bin by bin* packing approach with two phases. The two important factors that decide the quality of a bin-packing heuristic is

Repeat *until all items are packed*
  Phase - I: *Open a new bin and pack as many items as possible using any* ***bin by bin*** *packing strategy*
  Phase - II: *Perform* ***bulk packing*** *if possible based on equation (5)*
End Repeat

Fig. 3. An overview of bulk-bin-packing.

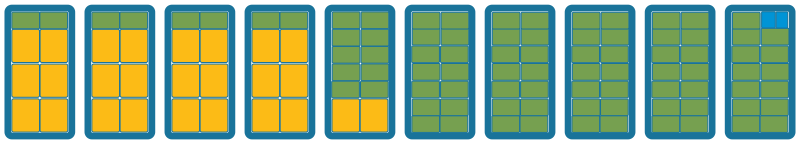

Fig. 4. A sample outcome of bulk-bin-packing with three repeated items.

- **Packing efficiency**: Packing into a minimum number of equal capacity bins exhibits this quality, Phase-I (Ref. Fig. 3) uses an efficient packing *bin by bin* strategy to place items into the next bin.
- **Speed & Scalability:** Phase-II (Ref. Fig. 3) improves the speed by identifying how many times the same packing can be repeated, returns a solution quickly while dealing with repeated items.

The *item by item* packing approach used by the greedy heuristics FFD, BFD, NFD and Max-Min (MM) packing makes them consume time, atleast in the order of $O(n)$. FFD, BFD and NFD heuristics will first sort the n items in decreasing order and pack all the items of the largest item size first, and then packs items with immediate smaller size up to the smallest item size. When items are repeated, the packing strategy can be also seen as *itemsize by itemsize* and can speed up using the BBP − *bin by bin* packing approach. BBP with FFD or MM in Phase-I requires sorting of only m representative items (each from an item type). Since $m \ll n$, the time to sort m items is far less than the time taken to sort n items.

Any *bin by bin* packing strategy can be sped up using BBP, provided the items are repeated. Fig. 4 shows a sample outcome of BBP with a group of bins packed with the same set of items. The other important factor to check is the packing efficiency and it depends on the packing strategy (Phase I). The end result of FFD and MM bin-packing strategies with or without incorporating in BBP is the same, so FFD and MM can be used for Phase-I of BBP. However, the packing efficiency of another well-known heuristic permutation pack varies (good as well as bad results), so it is not suitable for BBP. The power aware VM allocation like modified best fit decreasing [22] can also speed up using BBP, if VM requests are repeated. CSPs offer several standard VM instances. Customers can choose VMs according to their computational needs from the standard types. Maurer et al., did a cost-benefit analysis for standardization of cloud computing goods and suggested that the less the number of standard VM types, the more is the market liquidity and profit to CSP [23]. BBP based placement methods are scalable for large supply of standard VMs.

3.3. Bulk-Bin-Packing based VM Placement (BBPVP) focusing on single capacity constraint

If all the VM requests have equal resource ratios, then the VBP is reduced to BPP. A sample set of VMs that have the same resource ratios is extracted from the real-time Amazon EC2 dataset and presented here in Table 5. An optimal solution for a Bin-packing problem is packing items into a minimum number of possible bins. In general, there is no guarantee for zero gaps (resource wastage). But when all the VMs and PM have same resource ratios, FFD can place the given VMs into an optimal number of PMs. So a modified FFD is used in Phase-I of BBP method for scalable VM placement and is presented in BBPVP (Algorithm 1). All the VM requests in the input to BBPVP belong to the same resource ratio group and are variable in size S. (S_i, N_i) means N_i VMs of size S_i are to be placed and is similar to a one-dimensional BPP. The placement process carried out with respect to one capacity constraint will ensure the other capacity constraints also because of equal resource ratios.

3.3.1. BBPVP working with a case study

The step by step working of BBPVP algorithm is explained with a sample VM placement problem. Assume the resource ratios of all the VM types and PM is the same. VMs of different sizes (4, 2, 1 − *core*) need to be placed onto several PMs (each 63 − *core*). According to theorem-1, packing three types of VMs needs a maximum six iterations, packing of 4599 VMs is illustrated in Table 6.

Initially, the VM requests waiting for placement were 2621 (4 − *core*), 453 (2 − *core*) and 1525 (1 − *core*)

- **In iteration-1**: (of while loop), the first PM is filled with {15x4, 1x2, 1x1}-core VMs by steps 2 to 9 (Phase-I) and step 10 (Phase-II) does a *bulk packing* in 173 more PMs (i.e, total 174 PMs).
- **In iteration-2**: The remaining eleven 4-core VMs were completely placed in the next PM. The non-availability of 4-core VMs lead to a *single packing*.

Table 5
A group of VMs with same resource ratios.

VM Instance name	*CPU Cores (Size)*	*RAM (in GB)*	$\frac{RAM}{Cores}$
r3.large	2	15.25	7.625
r3.xlarge	4	30.5	7.625
r3.2xlarge	8	61	7.625
r3.4xlarge	16	122	7.625
r3.8xlarge	32	244	7.625

Algorithm 1
Bulk-bin-packing based VM Placement (BBPVP) focusing on single capacity constraint.

Input. PM capacities, VM capacities and repeat count
Output: Placement map
1. **while** VM requests available for placement
2. Launch new PM
3. **for each** VM request type in decreasing order of CPU Cores
4. **if** (remaining CPU Cores in PM > CPU Cores in Current VM type)
5. No of VMs can be placed (v) = (Integer) $\left(\frac{\textbf{remaining CPU Cores in PM}}{\textbf{CPU Cores in Current VM type}}\right)$
6. v = minimum(v, Number of VMs of this type available)
7. Place 'v' VMs in the current PM
8. **end if**
9. **end for**
10. Calculate bulk number using equation (5) and Perform **bulk packing** if possible.
11. **end while**

- **In iteration-3:** The next PM is filled with {31*x*2, 1*x*1}-core VMs by Phase-I and Phase-II does a *bulk packing* in 7 more PMs.
- **In iteration-4:** The remaining 2-core VMs are completely utilised in the *single packing*.
- **In iteration-5:** *Bulk packing* of the next PM configuration {63*x*1}-core VMs
- **No iteration-6:** Since the 1-core VMs are completely packed in iteration-5, there is no *single packing* followed.

3.4. Multi-Capacity Bulk Virtual machine Placement (MCBVP)

The virtual allotment and release of the resources is called virtual resource management and is a key function of a CSP. A PM has multiple resources like CPU, RAM, DISK, Network Bandwidth, and GPU. CSP offers several VM instances with different amounts of these resources. To minimise the multiple resource wastage, a greedy heuristic VM NeAR is proposed for choosing a VM based on PM's residual resource ratios to fill all the PM resources as much as possible. Using VM NeAR as the Phase-I of BBP, we propose a Multi-Capacity Bulk Virtual machine Placement (MCBVP) in Algorithm 2 for scalable and efficient VBP of VMs.

3.4.1. VM Nearest and Available to Residual resource ratios of PM (VM NeAR)

VM NeAR is a geometric VM selection heuristic based on PM residual resource ratios. It is to choose the best VM to place subsequently to minimise the resource wastage. Based on the VM's resource ratio vectors, all the VM instances are divided into subgroups as shown in Fig. 5. The tree structure formed is called a Resource Ratio Tree (RRT). The root node of RRT represents all VM types as a single group. The *i*th level in RRT represents the separation of VMs based on *i*th resource ratio. At last, each leaf node points to a set of VMs having same resource ratios and arranged in descending order of size (*CPU Cores*). Once a type of request is packed completely, the corresponding entry is removed and RRT shrinks dynamically.

The well-known best-first search traversal can be employed to identify the best VM subgroup by expanding towards the nearest child node (least absolute difference of each resource ratio, tie favours to the lesser value) at each resource ratio-level in the RRT. At each level, the next best node in the path is one among the child nodes. Each value in the PM residual resource vector is compared with the next set of child nodes. This process of VM subgroup identification needs a maximum of m comparisons, when m types of VMs are supplied for placement, a maximum of m sub groups possible.

Algorithm 2
Multi-capacity bulk VM Placement (MCBVP) for VMs with different resource ratios.

Input: PM capacities, VM capacities and repeat count ($\boldsymbol{VM_{RequestList}\langle r,S,N\rangle}$)
Output: Placement map
1. i←0
2. **while** VM requests available for placement
3. **while** PM[i] can accommodate any of the available VM requests
4. Update $\boldsymbol{R_{placement}}$← VM sub group nearest to(PM[i] residual resource ratios)
5. **while** PM[i] cannot accommodate the smallest VM in the subgroup $\boldsymbol{R_{placement}}$ then
6. Update $\boldsymbol{R_{placement}}$← VM sub group next nearest to(PM[i] residual resource ratios)
7. **end while**
8. PM[*i*].place(The largest VM that fits based on S among the VM subgroup $\boldsymbol{R_{placement}}$)
9. **end while**
10. BulkNumber = getBulkNumber(PM[*i*].configuration) using equation (5)
11. **for** j = 1 to BulkNumber do
12. place VMs of same configuration as in PM[i] into PM[i + j]
13. **end for**
14. i←(i + 1 + BulkNumber)
15. **end while**
16. **return PM**

Table 6
Illustration of bulk-bin-packing with three types of VMs in a maximum of six iterations.

Iteration	*Remaining VMs available for placement*			*VMs placed in Current PM*			*Number of PMs with same placement (BulkNumber)*
	$4-core$	$2-core$	$1-core$	$4-core$	$2-core$	$1-core$	
1	2621	453	1525	15	1	1	$Min\left(\frac{2621}{15}, \frac{453}{1}, \frac{1581}{1}\right) = 174$
2	11	279	1351	11	9	1	$Min\left(\frac{11}{11}, \frac{279}{9}, \frac{1351}{1}\right) = 1$
3	0	270	1350	–	31	1	$Min\left(\frac{270}{31}, \frac{1350}{1}\right) = 8$
4	0	22	1342	–	22	19	$Min\left(\frac{22}{22}, \frac{1342}{1}\right) = 1$
5	0	0	1323	–	–	63	$\frac{1323}{63} = 21$
6	0	0	0	–	–	–	–

3.4.2. Working of MCBVP with a case study

Although VM placement with MCBVP is not restricted to limited resources, for ease of understanding, a VM placement problem is illustrated here with only two resources (*CPU cores* and *RAM*). The VM placement will be based on 1 resource ratio $\frac{RAM}{CPU\ Cores}$, where VMs are grouped based on the resource ratio and each sub group may contain multiple VMs, with different size (*CPU Cores*) and same resource ratio. In the Algorithm 2, steps 3–9 (Phase-I) handles a typical VBP and the outcome of the first iteration is detailed in Table 7.

Five different types of VMs are placed with no resource wastage. VM instances with $\frac{RAM}{CPU\ Cores} = \{1, 2, 4\}$ are supplied to place in PMs with 55 CPU cores and 180 GB RAM. The allocation details are as follows:

- In placement-1, the nearest and available $\boldsymbol{R_{placement}}$ is 4, for the PM residual resource ratio $r_{res} = 3.27$

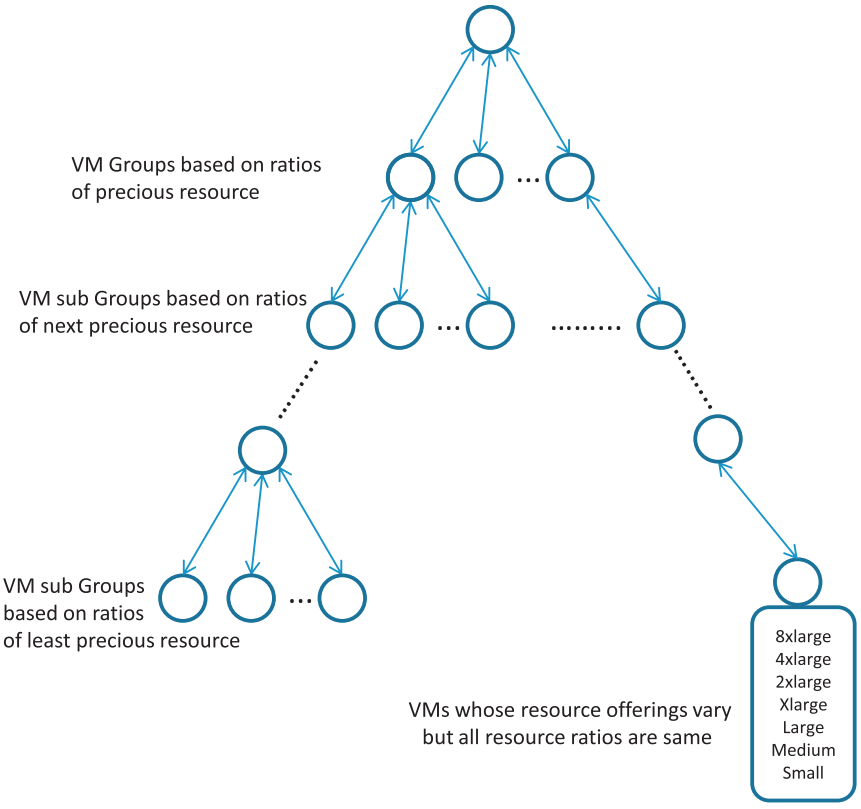

Fig. 5. Resource ratio tree with VMs grouped based on resource ratios.

Table 7
Case study – outcome of Phase-I in the first iteration of MCBVP.

Placement Number	PM			Place largest VM available in the subgroup		
	Residual Cores	Residual RAM	Residual *RAMperCore*(r_{res})	$R_{placement}$ nearest to r_{res} (*among* 1,2 *and* 4)	VM CPU Cores	VM RAM
1	55	180	3.272727273	4	8	32
2	47	148	3.14893617	4	8	32
3	39	116	2.974358974	2	8	16
4	31	100	3.225806452	4	8	32
5	23	68	2.956521739	2	8	16
6	15	52	3.466666667	4	8	32
7	7	20	2.857142857	2	4	8
8	3	12	4	4	2	8
9	1	4	4	4	1	4
	0	**0**				

- The largest VM {Cores = 8, RAM = 32} from the group $\boldsymbol{R_{placement}}$ (4) is selected and placed.
- After placement-1, r_{res} is 3.148 and is still near to same group of $\boldsymbol{R_{placement}}$ (4), so one more VM {Cores = 8, RAM = 32} is placed.
- After placement-2, r_{res} is 2.97, thus $\boldsymbol{R_{placement}}$ is (2) and the largest VM {Cores = 8, RAM = 16} from the group $\boldsymbol{R_{placement}}$ (2) is placed
- At last, the PM is hosting VMs with the following configuration ({Cores = 8, RAM = 32} *x* 4, {Cores = 8, RAM = 16} *x* 2, {Cores = 4, RAM = 8}, {Cores = 2, RAM = 8}, {Cores = 1, RAM = 4}).

In Algorithm 2, the steps 10–14 (Phase-II) of the MCBVP algorithm is same as the Phase-II of BBPVP algorithm and repeats the outcome (bin-packing configuration) of Phase-I. Thus, MCBVP also returns a solution in not more than $2\boldsymbol{m}$ iterations, where $\boldsymbol{m}$ denotes number of item types. Hence, the following case study is limited to Phase- I of MCBVP.

4. Simulation experiments and result analysis

4.1. Existing algorithms for comparison

The result of the proposed MCBVP is compared with recent works like Dot Product (DP) based HVRAA [7] and Norm-Based Greedy (NBG), VBP heuristics. Microsoft virtual machine manager uses both DP and NBG vector packing heuristics for VM consolidation [5].

Another proposed BBPVP is a rapid one-dimensional bin-packing heuristic and is compared with four well-known one-dimensional bin-packing heuristics: FFD, BFD, NFD [20] and MM [21] to test the speed and scalability of all these heuristics.

The implementations were carried out in an Intel® Core ™ i7 processor @ 2.5 GHz system with 4GB RAM running Windows 10 operating system. The existing VBP algorithms such as HVRAA, Norm-Based Greedy and the proposed MCBVP are implemented in a C + + environment. However, the one-dimensional standard Bin-packing algorithms like FFD, BFD, NFD, and MM, as well as the proposed algorithm BBPVP, are extended for VM placement using a standard bin-packing Java template. Three different datasets: Real Dataset – EC2 and two synthetic datasets were used to evaluate the speed, scalability and packing efficiency of the proposed BBPVP and MCBVP algorithms.

4.2. Dataset description

We consider all VM instances from a Real Dataset – EC2 [24] (Amazon EC2) which consists of 17 instances grouped into compute-optimized (5 instances), memory-optimized (5 instances), and 7 general purpose VM instances as shown in Table 8. A synthetic dataset was generated from 6 VM types with a random number of requests of each type, where the maximum number of requests varied from 10^2 to 10^9 which was further used to test the scalability of the bin-packing algorithms. Another synthetic dataset was generated for experimenting multi-capacity VM placement with random number of VM requests, each request consists of four resource requirements namely CPU Cores, RAM, Network bandwidth and Disk Storage.

4.3. Results & discussion

The Experimental analysis of the proposed heuristics was carried out to test the speed, scalability, and packing efficiency under two scenarios: (i) single capacity constraint and (ii) multi-capacity constraint VM placements. Experiment 1 shows that the proposed BBPVP is more scalable than the other one-dimensional bin-packing heuristics in terms of placement speed. Experiment 2 demonstrates the scalability of both BBPVP and MCBVP to handle bulk requests of order greater than 10^{10}. Experiments 3–5 were carried out to test the speed and packing efficiency of MCBVP in multi-capacity constraint VM Placement. The proposed VM initial placement methods are compared with the existing approaches based on the maximum number of VM requests handled (Fig. 6).

Experiment 1: The speed and scalability of the proposed BBPVP heuristic were evaluated by comparing with existing FFD, BFD,

Table 8
Standard VM instances from Real Dataset – EC2 with its resource ratio and sizes.

		VM instance Specs				*PM Specs*	
id	*model*	*Instance name*	*Cores (Size)*	*RAM (in GB)*	$\frac{RAM}{Cores}$ *(RpC)*	*logical processors*	*RAM(GB)*
1	General Purpose	t2.micro	1	1	1	24	32
2		t2.small	1	2	2		
3		t2.medium	2	4	2		
4		m3.medium	1	3.75	3.75	40	128
5		m3.large	2	7.5	3.75		
6		m3.xlarge	4	15	3.75		
7		m3.2xlarge	8	30	3.75		
8	Compute optimized	c3.large	2	3.75	1.875	80	128
9		c3.xlarge	4	7.5	1.875		
10		c3.2xlarge	8	15	1.875		
11		c3.4xlarge	16	30	1.875		
12		c3.8xlarge	32	60	1.875		
13	Memory optimized	r3.large	2	15.25	7.625	80	512
14		r3.xlarge	4	30.5	7.625		
15		r3.2xlarge	8	61	7.625		
16		r3.4xlarge	16	122	7.625		
17		r3.8xlarge	32	244	7.625		

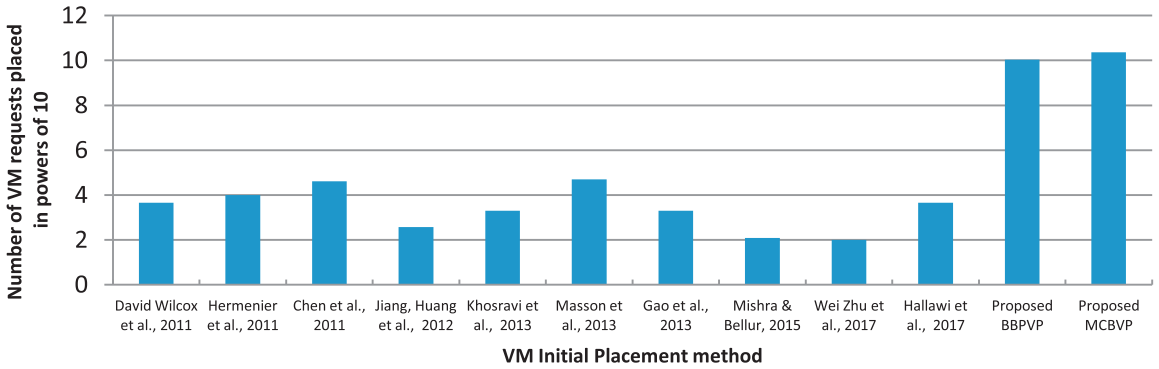

Fig. 6. A comparison of maximum VM requests placed by proposed MCBVP and other VM placement methods in the simulation experiments.

NFD and MM heuristics. In 80 schedules, each time a different set of requests were supplied. The set of requests supplied in a particular schedule is same to all the bin-packing heuristics. Six VM types were used each with the number of cores as 1, 2, 4, 8, 16, and 32. The VM requests are generated by varying the maximum repetition every ten schedules starting from 10^2 up to 10^9. From Fig. 7 and Table 9, there is an exponential increase in processing time of FFD and BFD, while handling million VM requests. Otherside, NFD and MM can place 100 times more VM requests than FFD and BFD. The processing time of BBPVP is not increasing in the same order compared to the other four heuristics. According to Coffman et al., FFD gives an optimal solution when item sizes are powers of 2, hence the items supplied for packing is of size (1, 2, 4, 8, 16, and 32) [25]. Since FFD is used in Phase 1 of BBPVP, the packing efficiency of BBPVP is also the same as FFD and is optimal.

Experiment 2 (a): This experiment is focused on testing whether BBPVP is capable of placing more than 10^{10} (10,789,926,669) VM requests. The VM types are sized 1–10 (CPU cores), and each PM has 116 logical processors. It is observed that BBPVP completely utilizes all the PMs except the last, which is an optimal solution. Since BBPVP is using FFD, we can expect the same optimal result from FFD and BFD also. But their processing time is high and further Experiment 1 shows that both FFD and BFD cannot handle more than 10^6 requests. The VM placement plan, the outcome of BBPVP is listed in Table 10, and the number of VM requests supplied for BBPVP is as follows (number of Cores ***x*** number of VM Requests). **{1 *x* 1,929,291,575, 2 *x* 967,898,840, 3 *x* 1,725,148,964, 4 *x* 1,032,752,920, 5 *x* 100,084,302, 6 *x* 255,891,346, 7 *x* 495,724,728, 8 *x* 1,951,885,190, 9 *x* 874,012,777, 10 *x* 1,457,236,027}**.

Experiment 2 (b): This experiment is similar to 2 (a) and is focused on testing whether MCBVP is capable of placing more than 10^{10} VM requests. All the 17 VM types in the Real dataset - EC2 are supplied as repeated requests (Table 11). The requests are to be placed in PMs, each with 160 CPU cores and 512GB RAM. MCBVP generated a placement plan for the given VM requests in 30 ms. HVRAA and Norm-based Greedy cannot generate a solution immediately since they need to sort all the remaining items every time to choose the best item to place. The VM placement plan, the outcome of MCBVP is listed in Table 12.

Experiment 3: This experiment is to analyse the speed and packing-efficiency of the multi-capacity VM placement heuristics. Four different clusters are considered each with different PMs (t2, m3, c3 and r3 specs detailed in Table 8) and corresponding VMs

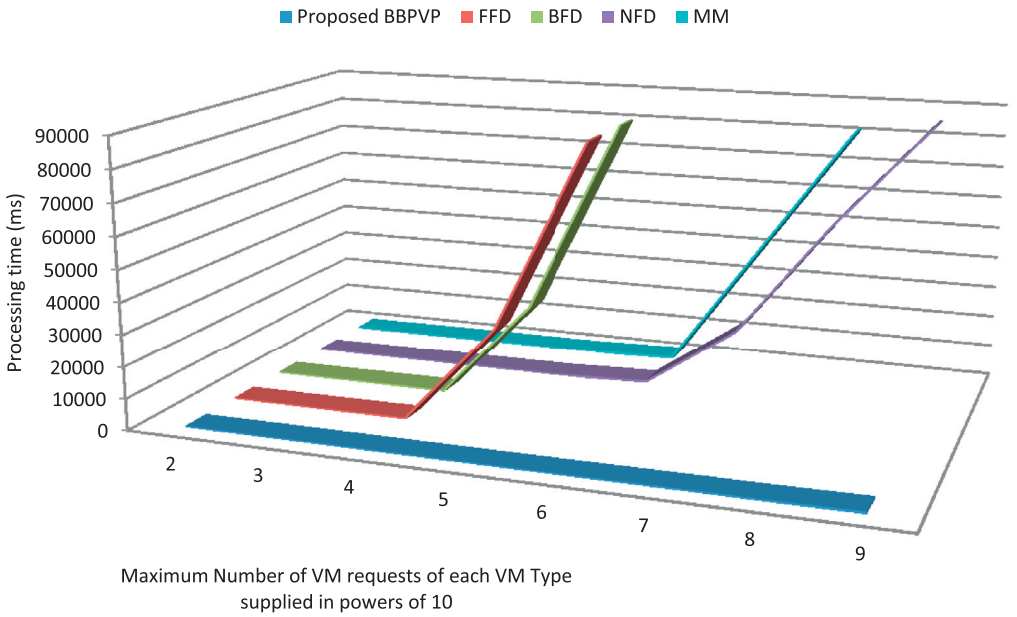

Fig. 7. Comparison of Total processing time taken by FFD, BFD, NFD, MM and BBPVP for ten placements.

Table 9
Total time for ten schedules with a request count up to $6 \times 10^{9.}$

Bin-packing heuristic		Maximum Number of VM Placement Requests supplied of each VM Type							
		10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
Proposed BBPVP	Processing Time (ms)	3	6	20	5	9	115	154	386
FFD [20]		0	5	154	31,398	∞	∞	∞	∞
BFD [20]		16	64	152	30,871	∞	∞	∞	∞
NFD [20]		2	7	6	86	921	20,813	57,604	∞
MM [21]		4	7	30	116	748	41,286	81,532	∞

'∞' represents processing time about an hour, which is not feasible for VM placement.

Table 10
Bulk-bin-packing more than 10^{10} VM requests - VM placement plan.

S.No	*Number of PMs*	*Bulk / Single Packing*	*VM placement Configuration* (*size x count*)	*Utilization* (%)
1	132,476,002	Bulk Packing	{6 *x* 1, 10 *x* 11}	100.0
2	1	Single Packing	{3 *x* 1, 9 *x* 7, 10 *x* 5}	100.0
3	72,834,397	Bulk Packing	{8 *x* 1, 9 *x* 12}	100.0
4	1	Single Packing	{6 *x* 1, 8 *x* 7, 9 *x* 6}	100.0
5	134,217,913	Bulk Packing	{4 *x* 1, 8 *x* 14}	100.0
6	1	Single Packing	{7 *x* 12, 8 *x* 4}	100.0
7	30,982,794	Bulk Packing	{4 *x* 1, 7 *x* 16}	100.0
8	1	Single Packing	{2 *x* 1, 6 *x* 5, 7 *x* 12}	100.0
9	6,495,544	Bulk Packing	{2 *x* 1, 6 *x* 19}	100.0
10	1	Single Packing	{4 *x* 1, 5 *x* 20, 6 *x* 2}	100.0
11	4,351,490	Bulk Packing	{1 *x* 1, 5 *x* 23}	100.0
12	1	Single Packing	{4 *x* 14, 5 *x* 12}	100.0
13	29,915,593	Bulk Packing	{4 *x* 29}	100.0
14	1	Single Packing	{1 *x* 1, 3 *x* 37, 4 *x* 1}	100.0
15	45,398,655	Bulk Packing	{2 *x* 1, 3 *x* 38}	100.0
16	1	Single Packing	{2 *x* 4, 3 *x* 36}	100.0
17	15,793,183	Bulk Packing	{2 *x* 58}	100.0
18	1	Single Packing	{1 *x* 72, 2 *x* 22}	100.0
19	16,594,310	Bulk Packing	{1 *x* 116}	100.0
20	1	Single Packing	{1 *x* 52}	44.83
	Total PMs: 489,059,891			

Table 11
$2.33x10^{10}$ instances supplied for MCBVP.

VM type	*Instance name*	*Demand*	*VM type*	*Instance name*	*Demand*
1	t2.micro	177,562,506	10	c3.2xlarge	2,079,402,956
2	t2.small	934,689,041	11	c3.4xlarge	1,925,494,887
3	t2.medium	2,139,256,487	12	c3.8xlarge	990,996,182
4	m3.medium	2,136,254,106	13	r3.large	1,697,511,457
5	m3.large	1,334,905,515	14	r3.xlarge	2,117,234,957
6	m3.xlarge	1,461,080,068	15	r3.2xlarge	2,027,326,367
7	m3.2xlarge	51,584,515	16	r3.4xlarge	1,470,140,885
8	c3.large	7,524,717,620	17	r3.8xlarge	1,528,284,239
9	c3.xlarge	472,102,851		**Total VM requests**	**23,296,298,779**

Table 12
VM placement plan of 2.33×10^{10} requests, generated by MCBVP in 30 ms.

Resource Wastage		*Resource Utilization* %	*Bulk* (*or*) *Single packing*	*VM Placement Configuration* (type x count)
vCPU	*RAM* (*GB*)			
0	2.75	99.5355	17,194,838	m3.large, r3.large, m3.2xlarge *x* 3, r3.xlarge, r3.4xlarge, r3.8xlarge
0	0.25	99.9578	Single	t2.small, m3.medium, t2.medium, m3.xlarge *x* 3, m3.2xlarge, r3.2xlarge, r3.4xlarge, r3.8xlarge
3	0	99.4932	19,729,167	t2.micro *x* 9, m3.xlarge, r3.8xlarge *x* 2
6	0.25	98.9443	Single	t2.micro x 3, t2.small, c3.large, m3.xlarge, r3.8xlarge *x* 2
8	1.5	98.3953	472,102,851	c3.xlarge, m3.xlarge, r3.8xlarge *x* 2
8	1.5	98.3953	263,712,681	c3.large *x* 2, m3.xlarge, r3.8xlarge *x* 2
0	0.25	99.9578	141,107,073	t2.small, m3.medium, t2.medium, m3.xlarge *x* 5, r3.2xlarge, r3.4xlarge *x* 3
0	0.25	99.9578	131,771,067	t2.small, m3.medium, t2.medium, m3.large *x* 10, r3.2xlarge, r3.4xlarge *x* 3
0	0.25	99.9578	Single	t2.small, m3.medium *x* 7, t2.medium, m3.large *x* 7, r3.2xlarge, r3.4xlarge *x* 3
0	0.25	99.9578	88,732,188	t2.small, m3.medium *x* 21, t2.medium, r3.2xlarge, r3.4xlarge *x* 3
2	1	99.4932	Single	m3.medium *x* 10, t2.medium *x* 4, r3.xlarge, r3.2xlarge, r3.4xlarge *x* 3
4	1	99.1554	92,028,764	t2.medium *x* 2, c3.2xlarge, r3.4xlarge *x* 4
4	0	99.3243	208,214,504	t2.medium *x* 6, r3.2xlarge *x* 8
0	3.25	99.451	Single	t2.medium *x* 9, r3.large, r3.xlarge *x* 7, r3.2xlarge *x* 4
0	3.25	99.451	38,255,732	t2.medium *x* 9, r3.large, r3.xlarge *x* 15
0	3.25	99.451	Single	t2.small *x* 10, t2.medium *x* 4, r3.large, r3.xlarge *x* 15
0	3.25	99.451	31,837,705	t2.small *x* 18, r3.large, r3.xlarge *x* 15
0	4.25	99.2821	Single	t2.small *x* 10, c3.2xlarge, r3.large, r3.xlarge *x* 15
0	86	85.473	87,386,543	r3.xlarge *x* 12, c3.8xlarge
0	86	85.473	Single	r3.large *x* 4, r3.xlarge *x* 10, c3.8xlarge
0	86	85.473	67,092,632	r3.large *x* 24, c3.8xlarge
0	281.5	52.4493	Single	c3.large, r3.large *x* 7, c3.8xlarge *x* 2
0	362	38.8514	418,258,502	c3.4xlarge, c3.8xlarge *x* 2
0	362	38.8514	301,447,277	c3.4xlarge *x* 5
0	362	38.8514	198,737,419	c3.2xlarge *x* 10
0	362	38.8514	Single	c3.large *x* 36, c3.2xlarge
0	362	38.8514	5,626,159	c3.large *x* 40
Total PMs used			2,583,235,112	

Table 13
Comparison of PMs used and processing time taken by Multi-Capacity VM placement heuristics.

VM instance type	*Number of requests*	*PMs used* by			*Processing Time* (*ms*)		
		NormBased Greedy	*HVRAA*	*MCBVP*	*NormBased Greedy*	*HVRAA*	*MCBVP*
t2	78,282	4952	4952	4931	4,002,993	1,990,357	1
m3	9003	1091	1035	1035	41,424	43,671	1
c3	3016	3317	3146	3146	102,909	113,063	2
r3	3122	567	553	553	1278	1444	1

were supplied as bulk requests. The comparision of number of PMs used and processing time (in milliseconds) is presented in Table 13, also depicted by the histograms in Fig. 8. It is observed that only in the 't2' cluster, where the items have two different resource ratios, MCBVP is dominating HVRAA by using less PMs. In the other three clusters, since the items have same resource ratio, both HVRAA and MCBVP exhibited equal packing efficiency.

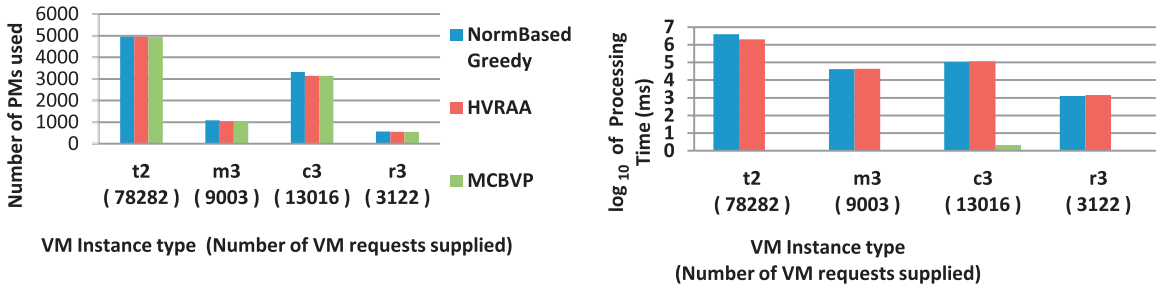

Fig. 8. Comparison of PMs used and processing time taken by Norm-Based Greedy, HVRAA and proposed MCBVP.

Table 14
VM Placement when 630 instances of all VM types are supplied.

	NormBasedGreedy	*HVRAA*	*MCBVP*
PMs used	66	62	61
Processing Time(ms)	121	137	5

Experiment 4: This experiment is to analyse and compare the packing efficiencies of HVRAA, Norm-Based Greedy, and proposed MCBVP heuristics with mixed types of VM instances (630 VMs with all 17 types). The comparison of total PMs and processing time (ms) used by each algorithm is tabulated in Table 14 and represented graphically in Fig. 9; which depicts that MCBVP is dominating the other two heuristics both in terms of processing time and packing efficiency.

Experiment 5: The VM instances in the *Real Dataset- EC2* have only two resource requirements RAM and CPU Cores. This experiment is carried out with a synthetic dataset with VMs having four resource requirements (CPU Cores, RAM, Network Bandwidth and Disk Storage) for further analysis of MCBVP performance in multi-capacity VM placement. A random amount of VM requests were supplied for each VM placement. The PM resource capacities, number of PMs used, processing time taken by both HVRAA and the proposed MCBVP is tabulated in Table 15 which depicts that MCBVP uses less PMs and faster in processing when compared to HVRAA. Thus MCBVP demonstrates better packing efficiency and high speed processing in the placement of repeated VM requests with multiple resources.

From the simulation experiments, it is inferred that bulk-bin-packing based packing techniques are scalable for large-scale placements by analyzing the processing time taken by all the one-dimensional packing techniques used in Experiment 1. The VM placement results shown in Tables 10 and 12 are obtained from Experiment 2, and they demonstrate the very-large-scale ($>10^{10}$) VM Placement capability of BBP based BBPVP and MCBVP. The results in Figs. 8 and 9 and Table 15 obtained from Experiments 3–5 reveal that BBP based MCBVP is rapid in placing repeated requests and also there is an improvement in packing efficiency compared to the well-known DP, and NBG approaches.

4.4. Time complexity analysis

The idea of FFD is to sort all the items in descending order and place the largest item that fits in the bin. While FFDProd, FFDSum, FFDAvgSum and FFDExpSum carries out sorting of n items only once in the beginning of placement process, the latest FFD variants Dot product, Norm-based Greedy and HVRAA performs n sortings, each for placing an item. If items are repeated, the time complexity of proposed BBPVP and MCBVP is same as time complexity of bulk-bin-packing i.e., $O(m)$ from Theorem 1. If items are not repeated ($m = n$), the time complexity of BBPVP is $O(n \log n)$, since it uses FFD strategy and MCBVP is $O(n)$ for n VM selections using VM NeAR. Table 16 provides the time complexities of VBP heuristic algorithms proposed as well as used for comparison.

5. Conclusion and future work

Energy efficient virtual machine placement is a major concern in the efficient energy management of cloud data centers. The proposed BBPVP and MCBVP heuristics advances the initial placement in two ways: (i) plays a significant role in the reduction of operational expenses and thus helps to sustain the competitive cloud resource provisioning industry. (ii) bulk-bin-packing based heuristic approaches had significantly improved the scalability of initial placement.

The limitations in the state-of-the-art VBP heuristics were addressed through a new item and bin representation called resource ratio vector; further, it was applied to item selection heuristic for bin-filling, the item available and nearest to residual resources ratios of the bin (VM NeAR). The performance evaluation of the proposed VMNeAR based MCBVP heuristic over the existing VBP heuristics was carried out on the real-time Amazon EC2 dataset and synthetic datasets in terms of number of PMs and packing speed which was reduced by 1.6% and enhanced 24 times faster respectively. Further, MCBVP placed 2.33×10^{10} VM requests in 30 ms and is faster than the fastest one-dimensional bin packing heuristic next-fit (when repeated VM requests are supplied bulk). There are

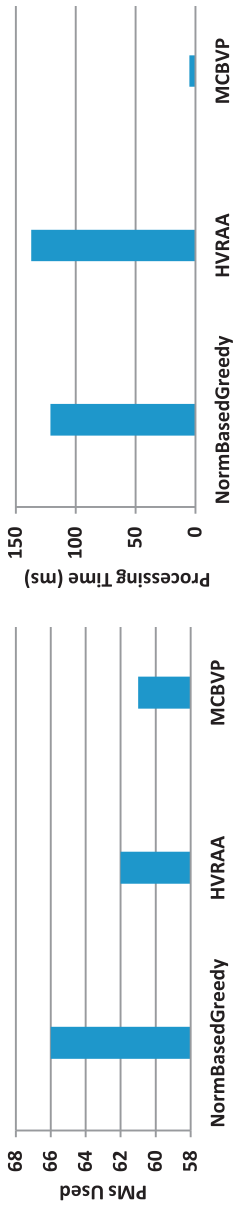

Fig. 9. MCBVP uses fewer PMs in less processing time than other multi-capacity VM placement heuristics.

Table 15
Comparison of multi-capacity VM placement results.

	Physical Machine Capacities				*PMs used by*		*Processing Time (ms)*	
Run	*CPU Cores*	*RAM*	*Bandwidth*	*Disk Storage*	*MCBVP*	*HVRAA*	*MCBVP*	*HVRAA*
1	500	500	500	500	526	528	163	1313
2	800	600	800	600	1077	1078	180	308
3	1000	1000	1000	1000	800	802	271	2583
4	1000	750	1000	1000	2661	2663	802	22,123
5	1000	1000	1500	1000	1352	1356	293	5434
6	1000	750	1200	1000	799	802	160	2156
7	500	750	500	250	3942	3946	1692	59,435
Total					11,157	11,175	3561	93,352

Table 16
Time complexity of VBP heuristics.

VBP Heuristic	Time complexity
FFDProd [5]	$O(n \log n)$
FFDSum [5]	$O(n \log n)$
FFDAvgsum [5]	$O(n \log n)$
FFDExpSum [5]	$O(n \log n)$
Dot Product [5]	$O(n^2 \log n)$
Norm-Based Greedy [5]	$O(n^2 \log n)$
HVRAA [7]	$O(n^2 \log n)$
Proposed BBPVP	$O(m)$
Proposed MCBVP	$O(m)$

*n − number of items, m − number of item types, $m << n$.

two reasons behind this speed (i) Continuous sorting is eliminated and (ii) Use of bulk-bin-packing to handle repeated requests efficiently. As a promising future work, the proposed bulk-bin-packing based heuristics can be extended to a cloud data center with heterogeneous physical machines and a cloud data center offering a standard list of software-as-a-service.

Acknowledgements

The authors thank the Department of Science and Technology, India for their financial support (SR/FST/ETI-349/2013) under Fund for Improvement of S&T Infrastructure in Universities and Higher Educational Institutions.

References

[1] Meijer GI. Cooling energy-hungry data centers. Science 2010;328(80-):318–9. http://dx.doi.org/10.1126/science.1182769.
[2] Stillwell M, Schanzenbach D, Vivien F, Casanova H. Resource allocation algorithms for virtualized service hosting platforms. J Parallel Distrib Comput 2010;70:962–74. http://dx.doi.org/10.1016/j.jpdc.2010.05.006.
[3] Wilcox D, McNabb A, Seppi K. Solving virtual machine packing with a reordering grouping genetic algorithm. 2011 IEEE Congr Evol Comput CEC 2011 2011. p. 362–9. http://dx.doi.org/10.1109/CEC.2011.5949641.
[4] Masson R, Vidal T, Michallet J, Penna PHV, Petrucci V, Subramanian A, et al. An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. Expert Syst Appl 2013;40:5266–75. http://dx.doi.org/10.1016/j.eswa.2013.03.037.
[5] Panigrahy R, Talwar K, Uyeda L, Wieder U. Heuristics for vector bin packing. Res Microsoft 2011:1–14. doi:10.1109/GreenCom-CPSCom.2010.137.
[6] Lee S, Prabhakaran RPV, Ramasubramanian V, Uyeda KTL, Wieder U. Validating heuristics for virtual machines consolidation. Res Microsoft Com 2010:81–97.
[7] Zhu W, Zhuang Y, Zhang L. A three-dimensional virtual resource scheduling method for energy saving in cloud computing. Futur Gener Comput Syst 2017;69:66–74. http://dx.doi.org/10.1016/j.future.2016.10.034.
[8] Jiang D, Huang P, Lin P, Jiang J. Energy efficient VM placement heuristic algorithms comparison for cloud with multidimensional resources. Lect Notes Comput Sci (Including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) 2012;7473 LNCS:413–20. doi:10.1007/978-3-642-34062-8_54.
[9] Khosravi A, Garg SK, Buyya R. Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8097 LNCS, 2013, p. 317–28. doi:10.1007/978-3-642-40047-6_33.
[10] Li Y, Tang X, Cai W. Dynamic bin packing for on-demand cloud resource allocation. IEEE Trans Parallel Distrib Syst 2016;27:157–70. http://dx.doi.org/10.1109/TPDS.2015.2393868.
[11] Canali C, Lancellotti R. Scalable and automatic virtual machines placement based on behavioral similarities. Computing 2017;99:575–95. http://dx.doi.org/10.1007/s00607-016-0498-5.
[12] Hallawi H, Mehnen J, He H. Multi-capacity combinatorial ordering GA in application to cloud resources allocation and efficient virtual machines consolidation. Future Gener Comput Syst 2017;69:1–10. http://dx.doi.org/10.1016/j.future.2016.10.025.
[13] Rahman M, Graham P. Compatibility-based static VM placement minimizing interference. J Netw Comput Appl 2017;84:68–81. http://dx.doi.org/10.1016/j.jnca.2017.02.004.
[14] Hermenier F, Demassey S, Lorca X, Chen M, Zhang H, Su YY, et al. Bin repacking scheduling in virtualized datacenters. Lect Notes Comput Sci (Including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) 2011;6876 LNCS:594–601. doi:10.1109/INM.2011.5990564.
[15] Chen M, Zhang H, Su YY, Wang X, Jiang G, Yoshihira K. Effective VM sizing in virtualized data centers. Proc 12th IFIP/IEEE Int Symp Integr Netw Manag IM 2011 2011. p. 594–601. http://dx.doi.org/10.1109/INM.2011.5990564.
[16] Gao Y, Guan H, Qi Z, Hou Y, Liu L. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J Comput Syst Sci

2013;79:1230–42. http://dx.doi.org/10.1016/j.jcss.2013.02.004.
[17] Shojafar M., Canali C., Lancellotti R., Baccarelli E.. Minimizing computing-plus-communication energy consumptions in virtualized networked data centers 2016:1–8.
[18] Canali C, Lancellotti R, Shojafar M. A computation- and network-aware energy optimization model for virtual machines allocation 2016.
[19] Canali C, Emilia R, Chiaraviglio L, Emilia R. Joint minimization of the energy costs from computing, data transmission, and migrations in cloud data centers. IEEE Trans Green Commun Netw 2018:1–16. http://dx.doi.org/10.1109/TGCN.2018.2796613.
[20] Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL. Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J Comput 1974;3:299–325.
[21] Zhu D. Max–min bin packing algorithm and its application in nano-particles filling. Chaos, Solitons Fractals 2016;89:83–90.
[22] Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. Future Gener Comput Syst 2012;28:755–68. http://dx.doi.org/10.1016/j.future.2011.04.017.
[23] Maurer M, Emeakaroha VC, Brandic I, Altmann J. Cost – benefit analysis of an SLA mapping approach for defining standardized cloud computing goods. Future Gener Comput Syst 2012;28:39–47. http://dx.doi.org/10.1016/j.future.2011.05.023.
[24] Zheng Q, Li R, Li X, Shah N, Zhang J, Tian F, et al. Virtual machine consolidated placement based on multi-objective biogeography-based optimization. Future Gener Comput Syst 2016;54:95–122. http://dx.doi.org/10.1016/j.future.2015.02.010.
[25] Coffman EG, Garey MR, Johnson DS. Bin packing with divisible item sizes. J Complex 1987;3:406–28. http://dx.doi.org/10.1016/0885-064X(87)90009-4.

Saikishor Jangiti is an Assistant Professor and pursuing Ph.D. in School of Computing, SASTRA Deemed University, Thanjavur, India. He received his B.Tech. in CS&IT from Jawaharlal Nehru Technological University, Hyderabad in 2005 and M.Tech. in CSE from Sri Venkateswara University, Tirupati in 2007. His research interests include artificial intelligence, greedy algorithms, multi-agent systems, machine learning, and cloud computing.

Shankar Sriram V.S. is an Associate Professor in School of Computing, SASTRA Deemed University, India. He received his B.Sc., & M.C.A. from Madurai Kamaraj University, Madurai, M.E. from Thapar University, Patiala, Ph.D. from Birla Institute of Technology, Mesra in 1997, 2000, 2004 and 2012 respectively. His research interests include information & network security, MANETs, steganography, and cloud computing.