Correlation-based heuristics and evaluation of existing greedy heuristics for VM allocations in cloud datacentres

Viney Sharma*

Department of Computer Science and Engineering,
Anand Engineering College,
Agra, India
Email: viney.sharma@sgei.org
*Corresponding author

Gur Mauj Saran Srivastava

Department of Physics and Computer Science,
Dayalbagh Educational Institute,
Agra, India
Email: gurmaujsaran@gmail.com

Abstract: Cloud computing has taken the world in its strides. In cloud datacentres, thousands of physical machines run continuously to execute incoming workload. Virtual machines are provisioned to incoming requests and are allocated to physical machines. Efficient mapping of virtual machines to physical machines has potential impact on the efficiency of datacentres. This paper proposes two greedy heuristics for virtual machines to physical machines mapping. We have empirically evaluated proposed heuristics and existing greedy heuristics for comprehensive datasets including PlanetLab datasets. Thereafter we have considered issue of hotspot, and proposed two heuristics for hotspot mitigation. We have evaluated our proposed hotspot mitigation heuristics for wide varieties of cases and case of SLA violation is also taken into consideration. Extensive simulation shows that our proposed heuristics are substantially faster than their counterparts. As clouds have strong business perspective also, our heuristics can be seen as prime alternate options for virtual machines to physical machines mapping.

Keywords: energy efficiency; greedy approaches; computing as a service; VM migrations; multidimensional bin packing; first fit; hotspot mitigation.

Reference to this paper should be made as follows: Sharma, V. and Srivastava, G.M.S. (2020) 'Correlation-based heuristics and evaluation of existing greedy heuristics for VM allocations in cloud datacentres', *Int. J. Information Technology, Communications and Convergence*, Vol. 3, No. 4, pp.276–318.

Biographical notes: Viney Sharma is Assistant Professor in the Department of Computer Science and Engineering, Anand Engineering College, Agra, India. He has 20 years of teaching experience. His areas of interest include cloud computing, artificial intelligence and machine learning.

Gur Mauj Saran Srivastava is an Associate Professor in the Department of Physics and Computer Science, Dayalbagh Educational Institute, Agra, India. He has more than 25 years of teaching experience. His areas of interest include data mining and cloud computing.

1 Introduction

Cloud computing model has provided computing as a utility (Persico et al., 2018) just like electricity, water, gas, etc. in which resources are made available as a utility and give an illusion of having as much resources as requested by the user. This computing paradigm is based upon pay-as-use model in which users pay for what they have used only. It emerged as a new paradigm for dynamic provisioning of computing services supported by state-of-the-art datacentres (Yin et al., 2019) that usually employ virtual machine (VM) technologies for consolidation and environment isolation purposes. Earlier companies, irrespective of the nature of their work had to spend large amount of money in setting up computing infrastructures, power supply equipments, cooling equipments, etc. Now people are choosing clouds as better option. In Cloud computing environment, datacentres have thousands of physical machines (PMs) to execute incoming computing load. VMs are provisioned to incoming requests. Once VMs are provisioned, these are assigned to PMs according to some predefined policy. As cloud computing has business perspective also and for running a business for long-term, total cost of ownership (TCO) need to be reduced and return on investment (ROI) need to be increased while maintaining quality at the same time. Cost of energy consumption contributes a big part in overall expenditure of a datacentre.

1.1 Deployment models

Cloud computing is a style of computing in which business processes, application, data, and any type of IT resource can be provided as a service to the users. Cloud provider offers certain deployment models for consumers to opt for.

1.1.1 Public clouds

Public cloud services are offered by third-party datacentre provider to end-user consumers over the internet. Public cloud offers resource pooling, self-service, service accounting, elasticity, multi-tenancy to manage the solutions, deployment, and securing the resources and applications. Companies can use it on-demand and with the pay-as-you-use option, it is much like utility consumption. Enterprises are able to offload commodity applications to third-party service providers. The term ‘public’ does not mean that it is free, even though it can be free or fairly inexpensive to use. It also does not mean that a user’s data is publically visible – public cloud vendors typically provide an access control mechanism for their users. Every workload is not ready for public cloud today. Workloads that depend on sensitive data, normally restricted to an organisation, are not public today. Most companies are not ready to move their LDAP server to a public cloud because of the sensitivity of the employee information. Healthcare

record – until the security of the cloud provider is well established is another example. Some other examples include:

- Workloads composed on various, dependent services.
- High throughput online transaction processing.
- Workloads requiring a high level of auditability and accountability.
- Non-virtualised workloads and non-availability of cloud-based licensing strategy.
- Workloads looking for complex service accounting mechanisms for different services for various departments-based billing.
- Workloads requiring flexibility and customisation.

1.1.2 Private clouds

Private clouds are deployments made inside the company's firewall (on-premise datacentres) and traditionally run by on-site servers. Private clouds offer some of the benefits of a public cloud computing environment, such as elastic on-demand capacity, self-service provisioning and service-based access. Private cloud is suitable when the traditional requirements, such as control, security, and resiliency, are more emphasised by an organisation with the restricted and designated user access and authorisation.

1.1.2.1 Services in private cloud

This section highlights the services provided by a private cloud and the services consumed from public cloud, specifically:

- virtualisation
- government and management
- multi-tenancy
- consistent deployment
- chargeback and pricing
- security and access control.

1.1.2.2 High 'cost of privacy'

Many experts believe a private cloud implemented with internal hosting/running of the infrastructure makes it difficult to realise many key benefits of clouds, including:

- *Eliminating capital expenses and operating costs:* Ownership of the hardware or software eliminates the pay-per-use potential, as these must be upfront purchases. The full cost of operations must be shouldered by them as there is no elasticity. If the private cloud hardware is sized for peak loads, there will be inefficient excess capacity. Otherwise, the owner will face complex procurement cycles.
- *Removing undifferentiated heavy lifting by offloading datacentre operations:* Utility pricing (for lower capital expenses and operating expenses) usually implies an

outside vendor offering on-demand services. It relies on the economies of multiple tenants sharing a larger pool of resources. These higher costs might be justified if the benefits of quicker and easier self-service provisioning and service-oriented access are large.

1.1.2.3 Private clouds provide more control

In traditional security models, location implies ownership which in turn implies control when security is location-specific. Then location, ownership, and control are aligned. Strong requirements for control and security usually drive a preference for a private cloud, where they own the cloud resources and control the location of those resources. For example, government may not want their applications or data to reside outside certain borders. Clouds rely on virtualisation; and in the public model, this loose coupling breaks the link between location and application, and reduces the perceived ownership and control. When we talk about the information control, it is not related to fixed geography or total ownership of the information. One example is public key encryption – the ownership of the key means control over the information without owning the rest of the infrastructure. The information control can be managed over the infrastructure that is trustful on the basis of the contracts, regulations, SLAs, standards, and imposition of the security mechanism on service providers. Compliance is difficult outside of traditional security models. As long as control through technology and contracts can be clearly demonstrated, it is possible to make public cloud computing environment as secure as a privately own facility. Auditors and regulators are continuously adapting to new technologies and business models. Owners can have multiple avenues as:

- full-implementation ownership
- lack of full ownership
- controlled ownership.

There are many possible approaches in between, such as partial control and shared ownership. There are also different levels of limited access – specific departmental access, industry-only access, and controlled partner access.

1.1.3 Hybrid clouds

A hybrid cloud is a combination of an interoperating public and private cloud. This is the model where consumer takes the non-critical application or information and compute requirements to the public cloud while keeping all the critical information and application data in control. The hybrid model is used by both public and private clouds simultaneously. It is an intermediate step in the evolution process, providing businesses an on-ramp from their current IT environment into the cloud. It offers the best of both cloud worlds – the scale and convenience of a public cloud and the control and reliability of on-premises software infrastructure – and let them move fluidly between the two on the basis of their needs. This model allows the following:

- elasticity, which is the ability to scale capacity up or down within minutes, without owning the capital expense of the hardware or datacentre
- pay-as-you-go pricing

- network isolation and secure connectivity as if all the resources were in a privately owned datacentre
- gradually move to the public cloud configuration, replicate an entire datacentre, or move anywhere in between.

1.1.4 Community clouds

This is the cloud managed by groups of people, communities, and agencies especially government to have the common interests – such as maintaining the compliance, regulation, and security parameters – working on the same mission. The members of the community share access to the data and applications in the cloud.

1.1.5 Shared private cloud

This is a shared compute capacity with variable usage-based pricing to business units that are based on service offerings, accounts datacentres. It requires an internal profit centre to take over or buy infrastructure made available through account consolidations.

1.1.6 Dedicated private cloud

Dedicated private cloud has IT service catalogue with dynamic provisioning. It depends on standardised service-oriented architecture (SOA) architectural assets that can be broadly deployed into new and existing accounts and is a lower-cost model.

1.1.7 Dynamic private cloud

Dynamic private cloud allows client workloads to dynamically migrate to and from the compute cloud as needed. This model can be shared and dedicated. It delivers on the ultimate value of clouds. This is a very low-management model with reliable SLAs and scalability.

1.2 Energy efficient datacentres

Datacentre energy consumption (Celesti et al., 2019) has nearly quadrupled in the past decade. It is reported that Celesti et al. (2019), US datacentres consumed 61 billion kilowatt-hours of power in 2006, which constitutes 1.5% of all power consumed in the US and represents a cost of $4.5 billion and its electricity consumption increased by nearly 40% from 2007 to 2012. Recent studies show that energy consumption by datacentres worldwide in year 2015 and 2016 was 405 billion kWh and 473 billion kWh respectively (Son and Buyya, 2019). So cloud providers are seriously concerned with reduction of energy cost. Moreover there has been increasingly pressure from governments (Vasic and Kostic, 2016) all over the world for green computing. In cloud computing datacentres, thousand of servers run continuously to execute load and research shows that their processor utilisation at any time is very low due to volatile resource demand. So, lot of research work has been done to device many heuristics to tightly pack VMs into minimum number of PMs. Existing heuristics are very CPU intensive, i.e., require large computations for VM to PM mapping. Since this allocation of VMs to PMs is a frequent process, so these heuristics make the process slow and result in more energy

consumption indirectly. Moreover when PMs are packed with maximum number of VMs, SLA violation may occur in peak hours and thus require VM migration process (Rastegar et al., 2019) which does not come free of cost. This process becomes more costly for migration on different networks (Qi and Tao, 2019). Several strategies have been proposed by researchers for this assignment. First fit/first fit decreasing (FF/FFD) are commonly implemented in commercial cloud softwares.

2 Related work

Research work related to energy efficient computing is going on at various levels. Xiong et al. in 2019 worked towards increasing efficiency of cloud servers and proposed an algorithm for resource management dynamically. They tested their algorithm for reduction in the time in VM migration and network traffic if VMs are migrated to distant location. Since VMs stop functioning during migration, so fast migration process is still area of research. Beloglazov et al. in 2015 analysed the sources of high energy consumption of PMs and then presented the classification of energy efficient operation of hardware and software parts of datacentre. Pedram in 2012 explored the main cause of rising energy consumption in cloud datacentres and proposed techniques for energy efficiency. He worked towards problem formulation for energy efficient resource allocation. Beloglazov et al. in 2016 proposed an algorithm for dynamic allocation and consolidation of VMs and powering down the nodes not in use. While allocating VM to PM, they have used modified best fit decreasing (MBFD) algorithm which chooses that PM among several PMs which resulted in least increase of energy consumption. They worked towards SLA violation and hotspot mitigation also. "When resource requirements of VMs increase in peak hours and cannot be met by PM holding it, it is condition of hotspot. When hotspot occurs, few VMs have to be transferred from current PM to other PMs." They proposed several algorithms for mitigation of hotspot like 'minimisation of migration' (MOM) and 'random choice'. In both of these algorithms for hotspot mitigation, they considered single resource. In MOM, those least number of VMs were selected for migration which could mitigate hotspot, so that overhead of migration could be avoided. Xiao et al. in 2015 calculated difference among values along various dimensions of resource requirements (RRs) vector and tried to accommodate different kind of workload to improve resource utilisation of servers. Bin packing approach was used to map VMs to PMs. Farahnakian and Pahikkala in 2019 formulated VM placement problem as stochastic integer programming problem to find out PMs which have sufficient resources to satisfy the RRs of VMs and keeping check on energy consumption. Xu et al. in 2016, suggested that resource migration could be done using server automation that provided the capability to perform bare-metal provisioning that enabled to deploy OS on physical and virtual hosts consistently. Jia et al. in 2017 studied and analysed the consequences of re-assigning the IP addresses to different sites after migration of VMs. Ceselli et al. in 2017 suggested the SLA-based resource allocation problem for multi-tier cloud applications. Their solution was a distributed solution for CPU, data and network elements. The problem was formulated as a three-dimensional vector optimisation problem. Li et al. in 2019 presented a technique for VM allocation and consolidation of VMs on minimum number of servers thus reducing the energy consumption. Gai et al. in 2019 discussed energy efficient VMs allocations in mobile clouds and VM migration methodologies considering various resources. Jia et al. in 2018

suggested greedy particle swarm optimisation to optimise the allocation of shared resources to have economical VM configuration. Gao et al. in 2018 explored greedy heuristics for VM to PM mapping. The main function of this algorithm was to minimise the estimated cost and enhance the VMs configurations. Chen in 2019 studied particle swam optimisation for VM placement optimisation in large cloud datacentres VM to PM mapping. Song et al. in 2019 proposed a genetic algorithm with resources taken as knapsack. They started with basic solution and then refined it so that chances of hotspot are reduced, thus requiring VM migration less number of times and hence saving energy. Yao and Yu in 2019 projected a strategy for dynamic allocation of resources on demand and reduced the number of active servers, thus saving power. They used bin packing approach. Zhang et al. in 2019 presented a technique for VM allocation and consolidation of VMs on minimum number of servers thus reducing the energy consumption. Dayarathna et al. in 2016 analysed FFD variants and according to them AvgSum variant of FFD was best. Madani and Jamali in 2018 studied multidimensional heuristics and compared various variants. Gergo et al. in 2012 studied the multidimensional vector bin packing (VBP) problem with dynamic costs and reduction of the bin packing problem to the modified multidimensional vector packing. They studied the VM migration issues and discussed provisioning of resources to VMs in energy efficient manner. They also found that avoiding VM migration was a good option as it carries extra overhead and slow down performance. It becomes more cumbersome when VM migration was done over different networks. We present below research analysis in tabular form mentioning merits and demerits of each work.

Table 1 Related research work

Author	*Idea*	*Technique*	*Harnessing components*	*Merits*	*Demerits*
Celesti et al. (2019)	Tightly packing of bins	Linear programming	General resources	Minimum number of resources	Overloaded resources
Zhang et al. (2019)	Comparison of various FFD variants	Bin packing	General resources	Minimum resources	Overloaded resources
Sotiridis et al. (2018)	To achieve minimum power, performance constraints	Switching server power	CPU, drive, network	Less power consumption	Unrealistic assumptions in workload
Xiong et al. (2019)	Minimum energy under performance constraints	DVFS	CPU	Deep analysis of various aspects in energy saving techniques, SLA considered	Single resource under consideration
Beloglazov et al. (2015)	Energy and VM consolidation	FFD	CPU	Workload consolidation to minimum number of PMs, SLA considered	Migration over network can slow down whole process

Table 1 Related research work (continued)

Author	*Idea*	*Technique*	*Harnessing components*	*Merits*	*Demerits*
Farahnakian and Pahikkala (2019)	Keep check on energy consumption	Stochastic integer programming	CPU	Keeps power consumption within range	Single resource under consideration. hotspot and coldspot condition not discussed
Song et al. (2019)	Reducing hotspot and VM migration	Genetic algorithm	CPU	Chances of hotspot reduced	Single resource under consideration. Condition of cold spot not discussed
Yao and Yu (2019)	VM consolidation	Linear programming approach	CPU	Server consolidated. hotspot/coldspot discussed	Single resource under consideration
Xiao et al. (2015)	Turning off lightly loaded PMs	Bin packing	CPU	Workload consolidated to less PMs	VM migration over network may slow down whole process

3 Research gap

Literature survey reveals that many novel approaches have been used by researchers in the field of cloud computing. Prominently, authors have worked upon genetic algorithm, stochastic integer programming, honey bee algorithm and bin packing approach for allocating VMs to PMs in cloud computing environment. Most of authors, through different approaches, have tried to find the mechanism to allocate maximum number of VMs to PMs with an objective to reduced power consumption. In most of research work carried out in recent years, authors have concentrated upon 'how to tightly pack PMs with VMs'. When a PM is tightly packed with VMs and resource demand of VMs residing in that PM increases, it results in hotspot situation. Either the hotspot issue has not been discussed in previous works or if it has been discussed, it incurs extra overhead due to VM migration especially if done over the network. A situation of coldspot may arise when a server runs at low utilisation. How to resolve coldspot issue, is not discussed appropriately in previous work. Also the algorithms discussed in previous works for mapping of VMs into PMs are compute intensive which consumes large processing time and increases power consumption of processor indirectly. Since VM allocation to PM is done frequently in clouds, faster algorithm is needed. So, identified research gap and further direction of research work can be summarised as follows.

- Faster algorithm for VM to PM mapping is needed which takes less CPU cycle, leading to less power consumption.
- Algorithm should avoid hotspot and if it occurs, algorithm should resolve it locally without need of migration over network as far as possible.
- Coldspot condition should be checked periodically and if occurs, workload of underutilised PM should be consolidated with other active PMs.

4 Objectives of paper

Based upon literature review and identified research gap, following objectives have been taken in this paper.

- To develop faster and general purpose heuristic for VM to PM mapping in cloud datacentres.
- To develop faster heuristics for local hotspot mitigation, avoiding migration over the network.

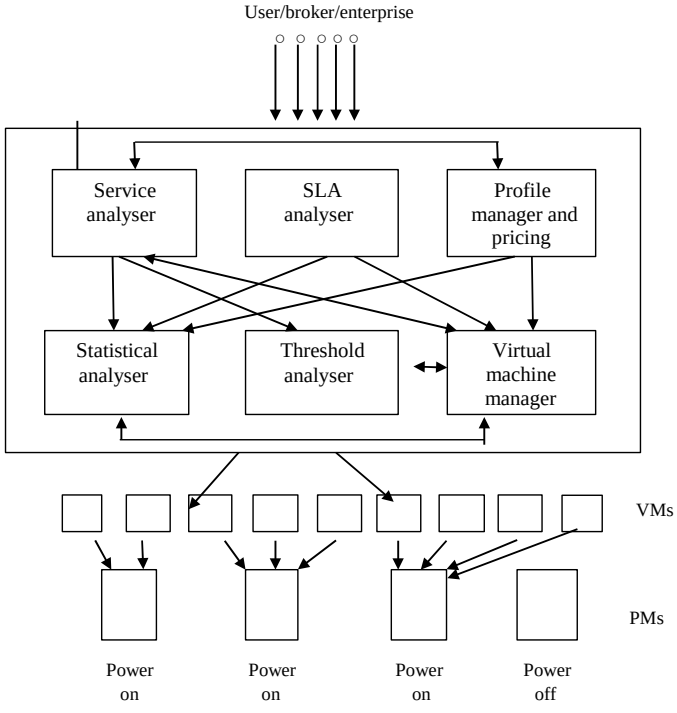
5 Proposed framework

The proposed framework describes how consumers interact with cloud system. It shows various interfaces through which requests pass through after getting accepted. We have upgraded the framework proposed by Beloglazov et al. (2016). In their framework, they have taken eight modules in green service allocator part. Out of eight, three are 'pricing', 'consumer profile', and 'accounting'. These modules are closely interlinked. 'Pricing' module tells how service requests are charged, 'consumer profile' module collects special features of consumer so that privilege may be given to such consumers and 'accounting' module monitors the actual usage of resources by VMs and accounts for the resource usage costs. We have taken six modules, which are interwoven, so interact frequently with each other for sharing information. So, we have taken functionality of above mentioned three modules into one module named as 'profile manager and pricing'. This framework has two new modules named as 'SLA analyser' and 'statistical analyser', for keeping SLA information and complete history of all applications respectively. 'Service analyser' modules continuously interact with 'SLA analyser' module to guarantee that SLA is not compromised. 'Threshold analyser' module is taken exclusively for keeping track of usage of various resources. 'VM manager' periodically interacts with 'threshold analyser' module to know current usage values and threshold values for various resources. The 'VM manager' module comprises of various functions. It is most important part of whole framework. It is responsible for overall performance of whole framework. It is responsible for allocating VMs to incoming requests and then allocating VMs to PMs in energy efficient manner. Algorithms for VMs to PM mapping are part of this module. This module continuously interacts with other modules in framework for

getting updated information about all VMs. It keeps on interacting with 'threshold analyser' module to have latest information about resource usage. As soon as a resource crosses its threshold usage, VMs need to be migrated. VM migration process is part of this module. As Figure 1 shows, users/brokers/enterprises interact with cloud architecture for accessing cloud resources. Here complete architecture is divided into three layers namely top layer, middle layer and bottom layer. Top layer consists of users/brokers/ enterprises. Middle layer consists of following six modules which is core part of cloud architecture. Bottom layer consists of PM which host VMs. Various modules are discussed below.

- *Service analyser* – This module analyses the service requirements of incoming applications. Service requirements can be of two types – static and dynamic. Static requirements infrequently changes with time while dynamic requirements frequently changes with time. So, analysing service requirements of applications is essential for efficient working of framework. It also takes information regarding resource utilisation and energy consumption from 'threshold analyser', current statistics from 'statistical analyser' and availability of VMs from 'VM manager' respectively.
- *SLA analyser* – Responsibility of this module is to make sure that SLA is not violated. It keeps record of SLA details, priorities, privileges of various customers and continuously monitors these parameters so that SLA is not violated. When workload for a particular application changes and if customer has requested for resources scaling, it informs 'VM manager' module regarding it.
- *Threshold analyser* – It keeps utilisation threshold for all resources. 'Service analyser' continuously interacts with 'threshold analyser' while accepting new application. During execution of current requests, it makes sure that utilisation of resources does not cross their threshold values.
- *Statistical analyser* – It records information regarding current utilisation of all resources and past history of applications related to their resource utilisation. 'Service analyser' interacts with 'statistical analyser' while serving requests from applications.
- *Profile manager and pricing* – It decides priority of customer so that special status can be given to higher priority customer over the others. It also saves information regarding auto scaling requests by customers. It charges the service requests as per defined policy. It monitors the actual usage of resources by VMs and maintains accounts for the resource usage costs. Billing monitor interacts with 'statistical analyser' for extracting statistics for billing purposes. It interacts with 'service analyser' to get information of new requests.
- *VM manager* – This module keeps track of available VMs and resource usage of existing VMs. As new request comes in, it allocates VMs to new applications in consultation with 'service analyser' module. It is also responsible for VM migration in hotspot mitigation and coldspot mitigation. It interacts with 'statistical analyser' for coldspot and hotspot mitigation.

Figure 1 Proposed framework for cloud computing



6 Methodology

We have formulated problem of VM to PM mapping as a multidimensional bin packing problem. Given n items with sizes $s_1, s_2, \ldots, s_n$ such that $0 \leq s_i \leq 1$ for $1 \leq i \leq n$, pack them into the fewest number of unit capacity bins. Problem is NP-hard (NP – complete for the decision version). There is no known polynomial time algorithm for its solution, and it is conjectured that none exists. There is difference between multiple knapsack problem and bin packing problem. Knapsack problem pack a subset of the items into a fixed number of bins, with varying capacities, so that the total value of the packed items is a maximum. Bin packing problem suggest: given as many bins with a common capacity as necessary, find the fewest that will hold all the items. In this problem, the items are not assigned values, because the objective does not involve value.

6.1 Vector bin packing

Generally VM placement problem has been solved by considering only one resource, i.e., one dimension while VMs require multiple resources. So, this problem can be simulated as VBP problem. In VBP we are given n things having *d*-dimensions and we are to place these things into minimum number of bins which are represented by same number of dimensions. Let $V1, V2, V3, \ldots, Vn$ be n number of vectors representing n number of things. VBP maps these n vectors to r number of bins $B1, B2, B3, \ldots, Br$ having same dimensions as that of vectors where for each bin x and for each dimension y.

$$\sum m \in Bx\, Vy^m <= 1$$

Here objective is to minimise *r*. VBP is NP-hard for every *d* and APX-hard for *d* >= 2. VBP can be applied in cloud computing environment for allocating VMs to PMs. VMs are treated as things and bins can be treated as PMs. Each VM has requirement of various resources like processor, memory and I/O. Similarly PM has capacity across these resources. So while assigning VM to PM in VBP problem, the demand across each dimension in VM should not cross remaining capacity across each respective dimension in PM. This restriction across each dimension in each individual PM makes this assignment less liberal. FF/FFD and its variants are primary heuristics to solve this problem. Some weight functions are applied to the demands of VM across each dimension to get a scalar value. Applying weights have other advantages also. If mentioning resource demand across few particular dimensions is not important (say PM has sufficient resources along few dimensions) then problem can be reduced in lesser number of dimensions or in single dimension. If allocation is to be favoured towards a subset of resources then weights in other dimensions can be kept lower. So weights help in moulding the solution as per requirement (Zhang et al., 2019).

6.2 Uniqueness of work

The uniqueness of proposed work is that both of the proposed heuristics are uniquely efficient in terms of speed and PMs used. Heuristics have been tested upon wide range of datasets. Most of the works by researchers in this field have been tested with PlanetLab datasets only, but we have tested our heuristics on low workload, medium workload and high workload and PlanetLab datasets. The reason is that PlanetLab includes datasets of few prominent datacentres only; but cloud computing has penetrated into small and medium level enterprises also. So, workloads of organisations vary from low to high. Our work has been tested to be more efficient than others work in all test cases. We have presented this model for each and every type of workload datacentres. Other uniqueness of our work is that while other works have concentrated on either resource optimisation or time efficiency of their heuristics; we have taken both of these aspects into consideration. Another specialty of our work is that we have considered two resources (processor and memory) into consideration while most of previous works have taken single resource.

6.3 Studying existing FFD variants

Greedy algorithms have come up with promising candidates for solving single dimension bin packing problems. One of the heuristics which has been used by researchers is FF/FFD with several variants. By assigning some appropriate values (weights) to multidimensions, we can generate FFD for multidimensional bin packing problem too. Two broad categories of variants used by researchers are first fit decreasing product (FFDP) and first fit decreasing sum (FFDS).

6.3.1 First fit decreasing product

Here product of dimensions of vector is taken and then sorted in decreasing order. Then vectors in decreasing order are assigned to bins.

$$Product\ (v) = \prod j <= d \quad Vj$$

6.3.2 First fit decreasing sum

Here sum of dimensions of vector is taken and then sorted in decreasing order. Then vectors are assigned to bins in decreasing order. But in FFDS weights are used to mould solution.

$$Sum\ (v) = \sum j <= d \quad (bj\ Vj)$$

Values of *bj* depend upon how we want to modulate dimensions. Value of *bj* can be taken as average demand across dimension *j* in all VM candidates.

i.e., $bj = 1/n \sum r = 1$ to $n\ (Vj^r)$ across dimension j.

When value of *bj* is taken as average across all dimensions, we call it FFD average summation (FFDAVG). Son and Buyya (2019) shows FFDS showed the best performance.

6.3.3 First fit decreasing dot product (FFDDP)

Let at time *t*, *R*(*t*) denote the remaining capacity vector of bin under consideration. Then this algorithm places that VM first which has highest value of the weighted dot product $\sum j\ bj\ (Vj)^r\ R(t)\ j$.

6.3.4 Norm-based greedy approach (NBGA)

This approach calculates $\sum j\ bj(Vj^r - R(t)\ j)^2$ and places the VM to PM in the increasing order of this value.

7 Proposed heuristics for VMs allocations in cloud computing environment

Datacentres host PMs, which host VMs to execute incoming workloads. Here we propose two heuristics for VMs to PMs mapping.

7.1 Proposal one – correlation-based first fit decreasing

One of our proposed heuristics follows correlation-based approach. We name it as correlation-based first fit decreasing (CBFFD). We have taken the case of two dimensional RR vector (processor and memory). If requirement along two dimensions are correlated, i.e., requirements along both dimensions are in same range, we delay that VM for placement. VM with least correlation (largest difference of RRs along two dimensions) will be taken first for allocation otherwise at later stage such VM will create problem due to mismatch in remaining vector and RR vector. Propose heuristics first takes difference between RRs of VM along both dimensions. Then our algorithm sorts

VM in the decreasing order of difference between RRs of VM along both dimensions. Let at time *t*, *D*[*r*] = absolute value of (*VR*1 – *VR*2) for all VMs and R1 and R2 are RRs of VMs along two dimensions, i.e., along two resources (processor and memory). *r* varies from 1 to maximum number of VMs. Outline of algorithm is given below.

1 Input: List of VMs and PMs

2 Output: Mapping from VM list to PM list

3 Calculate difference between RRs along both dimensions

4 Sort VMs in the decreasing order of the difference

5 Repeat step 6 for each VM in sorted VMList

6 Repeat step 7 for each PM in PMList

7 If (VM requirements along each dimension is less than remaining capacity along respective dimension in PM) then
   a allocate VM to PM
   b update remaining vector in PM
   c take next VM (next iteration in step 5)

8 Exit.

7.2 Proposal two – first fit increasing product

Our second proposed heuristic is first fit increasing product (FFIP).We calculate product of RR along both dimensions for each VM. Then VMs are sorted in increasing order of this product value and allocation to VMs to PMs is done in increasing order of this product value. FFIP has performed outstandingly for low RR. Outline of algorithm is given below.

1 Input: List of VMs and PMs

2 Output: Mapping from VM list to PM list

3 Calculate product of the RRs along both dimensions *D*[*r*] = *VR*1 * *VR*2

4 Sort VMs in the increasing order of the product

5 Repeat step 6 for each VM in sorted VMList

6 Repeat step 7 for each PM in PMList

7 If (VM requirements along each dimension is less than remaining capacity along respective dimension in PM) then
   a allocate VM to PM
   b update remaining vector in PM
   c take next VM (next iteration in step 5)

8 Exit.

8 Energy efficient hotspot mitigation

In cloud datacentres, RR of VMs may increase over time. When RR increases, increased demand is first satisfied by the PM holding that VM. But when PM is unable to satisfy the increased demand of VM, it is called hotspot. Then hotspot mitigation is done by moving VM to some other PM which has sufficient available capacity of resources. The judicious choice of choosing destination PM has major role in decreasing the energy consumption and overall enhancement in performance. In the present work, we have proposed two algorithms. Our algorithms are variants of MOM algorithm proposed by Beloglazov et al. (2016).

8.1 Proposed first mitigation algorithm (Miti)

In our first hotspot mitigation algorithm, we pick PM which is observing hotspot. Then algorithm replaces the VMs in that PM until it reaches to state where no further VM can be placed. To place remaining VMs instead of switching on new PM, search starts from first PM once again to check if any of PM can accommodate the remaining VMs of overloaded PM. This strategy is followed because it is also possible that some PMs may be having necessary resources to accommodate few of VMs of overloaded PM. If none of already running PM can satisfies the RRs of VMs in question then only new PM is switched on. The first proposed algorithm for hotspot mitigation is presented ahead.

Algorithm Miti

```
1  Input: PM with hotspot
2  Output: Reallocation of those VMs
3  Repeat steps 4–7 for j = 1 to n                    // for total number of PMs
4  If PM[j] is having hotspot, then
     1  Extract VMs residing in PM in VMList          // suppose total s
5  Repeat step 6 for each VM[i] in VMList (i varies from 1 to m, where m is total no. of VMs)
6  If current PM can meet the resource requirements of current VM in consideration then
     1  Place it in current PM
     2  Keep its entry in allocation table allocation[i][j] = 1   // allocation[i][j] stores VM to
                                                                  PM mapping
     3  Set VM[i].vmplaced = True
     Else
        1  Repeat for k = 1 to n                      // search all PMs
        If PM[k] can meet resource requirements of any leftover VMs then
           a  Place VM in current PM
           b  Make entry in allocation table allocation[i][k] = 1
7  Repeat step 8 for k = 1 to s
8  If VM[i].vmplaced <> True, then
     1  Switch on new PM
     2  Allocate current VM to new PM
     3  Make entry in allocation table
```

```
    4   VM[i]. vmplaced = True
9   Exit
```

8.2 Proposed second mitigation algorithm (Miti1)

In second algorithm, we take difference of RRs along both dimensions of those VMs. Let it is denoted by *d*. Then we sort those VMs in decreasing order of this value. Larger the difference of RRs along both dimensions, lesser is the correlation between them. It means we will reallocate VMs with high value of *d* first. Then we map these VMs in decreasing order of value d on PM which was having hotspot. We allocate as many VM so that PM is not overloaded. For those VMs which could not be placed, we start searching PM from first one to find if any PM has sufficient leftover resources along respective dimensions to satisfy RR of remaining VMs. If such PMs exist then VMs are placed in those PM otherwise new PM is switched on to accommodate leftover VMs. We have taken memory RR in MB/GB and CPU requirement in percentage of total CPU share available. The outline of algorithm is presented below.

Algorithm Miti1

```
1   Input: PM with hotspot
2   Output: Reallocation of those VMs
3   Repeat steps 4–7 for j = 1 to n                    // n for total number of PMs
4   If PM[j] is having hotspot, then
    1   Extract VMs residing in PM                     // suppose total no. of VMs in current PM is r
    2   Calculate correlation along two                // (correlation is difference of resource
        resource dimensions                            requirement along two dimensions // i.e.,
                                                       d = VM[i].CPU – VM[i].MEM)
    3   Arrange those VMs in decreasing order of d in VMlist.
5   Repeat for each VM[i] in VMList                    // suppose total no. of VMs in system is m
    1   Place it in current PM
    2   Keep its entry in allocation table allocation[i][j] = 1
    Else
    1   Repeat for k = 1 to n                          // search all PMs
        If PM[k] can meet resource requirements of any leftover VMs then
            a   Place VM in current PM
            b   Make entry in allocation table allocation[i][k] = 1
6   Repeat step 7 for k = 1 to r
7   If VM[i]. vmplaced <> True, then
    1   Switch on new PM
    2   Allocate current VM to new PM
    3   Make entry in allocation table
    4   VM[i]. vmplaced = True
8   Exit
```

9 Empirical evaluation

9.1 Proposed heuristics for VMs allocations in cloud computing environment

We have written an ad hoc simulator (*VMSimul*) to analyse different variants of FF/FFD and our proposed heuristics. *VMSimul* has been designed to simulate cloud computing environment. Requests for resources by VMs have been generated randomly in *VMSimul*. We have taken wide spectrum of requests for resources which match cloud environment. We have examined variety of cases. First we have divided our simulation process for four categories of resource demand.

1 Low resource demand (up to 15% of total available resource).

2 Arbitrary (from least to maximum).

3 Within realistic range for cloud environment (between 15% and 60% of total available resource along both dimensions).

4 PlanetLab data.

In first three cases we have taken two subcases – first of incoming request of 500 VMs and second of 1,000 VMs. Resource demand by VMs have been generated randomly as per requirement of first three cases. So we have six cases (3 * 2) in all. For each case, simulation has been run for five times and then average of data has been taken. For each case two Bar charts have been drawn: one for number of comparisons done in VM to PM mapping and second for number of PMs used. RR are taken in percentage of total available resource.

Case 1 No. of VMs = 500, 0 < RR <= 15%

Table 2 shows that for Case 1, FFDDP, FFDAVG, FFDP, FFDSUM, FFIP need 23 PMs followed by CBFFD which require 22 PMs, followed by NBGA which needs 27 PMs. Average number of comparisons for VMs to PMs mapping are 4885 for FFIP, 5513 for CBFFD, which are much less as compared to NBGA, FFDDP, FFDAVG, FFDP, FFDSUM, which require on average 7,100 comparisons.

Table 2 Simulation results for Case 1 (n = 500, 0 < RR <= 15%)

Heuristics	*Trial I*		*Trial II*		*Trial III*		*Trial IV*		*Trial V*		*Average*	
	No. of PMs used	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*Average no. of PMs used*	*Average no. of comparisons*
NBGA	27	7,108	27	7,205	27	6,955	26	6,837	26	6,848	27	6,991
FFDDP	23	7,288	24	7,275	23	7,057	23	6,988	23	6,975	23	7,117
FFDAVG	23	7,288	24	7,275	23	7,057	23	6,988	23	6,975	23	7,117
FFDP	23	7,230	24	7,233	23	6,997	23	6,941	23	6,904	23	7,061
FFDSUM	23	7,262	24	7,270	23	7,058	23	6,990	23	6,977	23	7,111
FFIP	24	4,989	24	5,023	23	4,924	23	4,638	23	4,853	23	4,885
CBFFD	22	5,487	23	5,480	22	5,625	23	5,497	22	5,479	22	5,513

For low RR, our proposed heuristics have given much better performance with FFIP taking least number of comparisons. Comparing statistically, Figure 2 and Table 2 show that FFIP is about 32% and CBFFD is about 23% faster as compared to their counterparts. FFDDP and FFDAVG are much slower and giving same level of performance. Among all heuristics (other than ours), NBGA is little bit better in terms of number of comparisons performed. Figure 3 and Table 2 depict that NBGA has used about 15% more PMs as compared to best counterparts. CBFFD has used about 18% less PMs as compared to NBGA. FFIP is at par with others while CBFFD is best. FFDDP, FFDAVG, FFDP and FFDSUM have used same number of PMs.

Figure 2 No. of comparisons for Case 1 (n = 500, 0 < RR <= 15%) (see online version for colours)

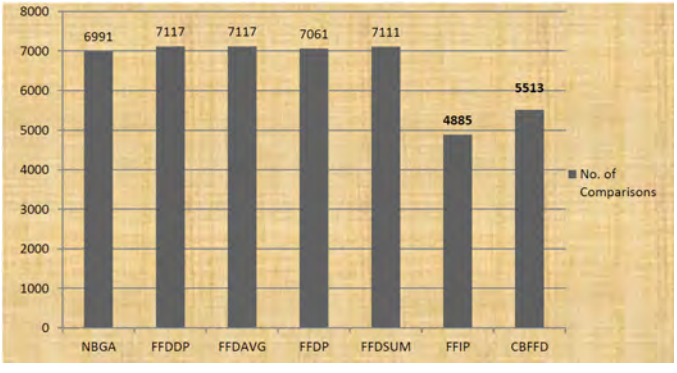
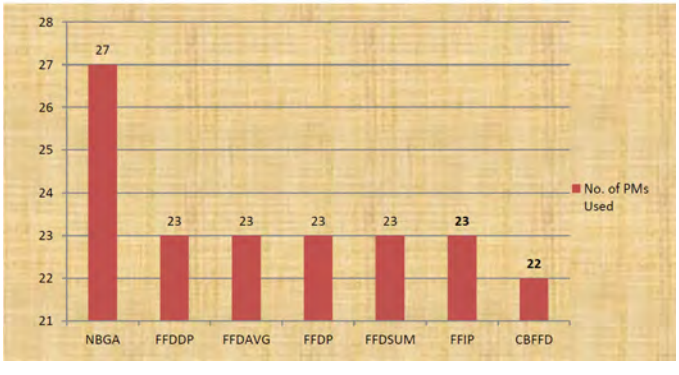


Figure 3 No. of PMs used for Case 1 (n = 500, 0 < RR <= 15%) (see online version for colours)



Case 2 No. of VMs = 1,000, 0 < RR <= 15%

Table 3 shows that for Case 2, FFIP and CBFFD have taken 47 and 45 PMs respectively. FFDSUM need 47 while FFDDP, FFDAVG, FFDP need 46 PMs followed by NBGA which needs 52 PMs. Talking about efficiency, FFIP and CBFFD are most efficient ones in terms of number of comparisons followed by remaining all with an average number comparisons of about 27,955 for VMs to PMs mapping. FFIP has performed 18,809 and CBFFD has performed 21,491 comparisons.

Table 3 Simulation results for Case 2 (n = 1,000, 0 < RR <= 15%)

Heuristics	*Trial I*		*Trial II*		*Trial III*		*Trial IV*		*Trial V*		*Average*	
	No. of PMs used	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*Average no. of PMs used*	*Average no. of comparisons*
NBGA	52	27,466	53	27,331	52	27,034	52	27,596	53	27,535	52	27,392
FFDDP	46	28,363	47	28,294	45	27,805	46	28,323	45	27,913	46	28,139
FFDAVG	46	28,363	47	28,294	45	27,805	46	28,323	45	27,913	46	28,139
FFDP	46	28,175	47	28,097	45	27,575	45	28,216	45	27,757	46	27,964
FFDSUM	47	28,355	48	28,284	46	27,777	47	28,344	46	27,948	47	28,141
FFIP	47	19,168	48	19,086	47	18,513	48	18,788	47	18,492	47	18,809
CBFFD	45	21,622	46	21,550	46	21,554	45	21,304	45	21,424	45	21,491

Figure 4 and Table 3 illustrate that FFIP is 33% and CBFFD is 24% faster as compared to its counterparts. FFDDP and FFDAVG have performed same number of comparisons. Among all heuristics (other than ours) NBGA has performed comparatively lesser number of comparisons. Figure 5 and Table 3 show that CBFFD has taken about 5% less PMs as compared to their best counterparts and about 12% lesser than NBGA, while FFDDP, FFDAVG and FFDP have used same number of PMs, i.e., 46. FFIP is almost at par with others. NBGA has taken maximum number of PMs, about 12% more than the best.

Figure 4 No. of comparisons for Case 2 (n = 1,000, 0 < RR <= 15%) (see online version for colours)

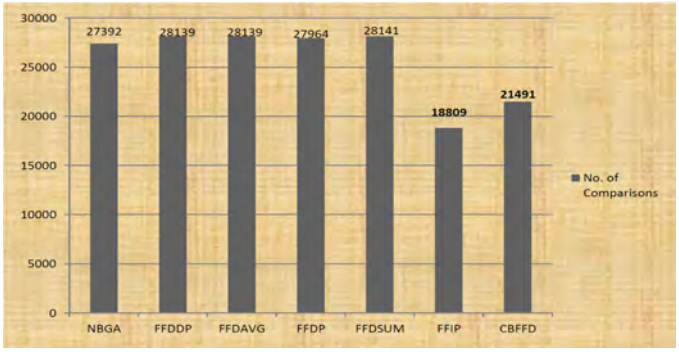
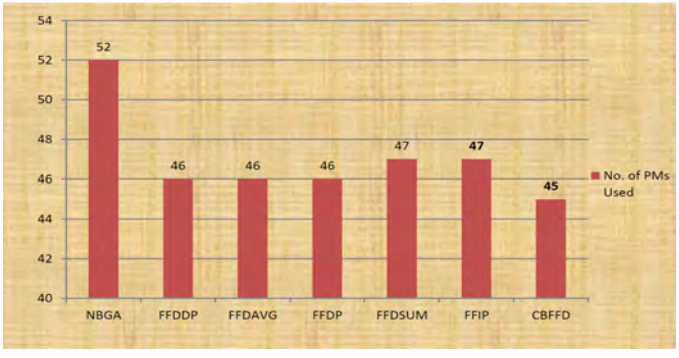


Figure 5 No. of PMs used for Case 2 (n = 1,000, 0 < RR <= 15%) (see online version for colours)



Case 3 No. of VMs = 500, arbitrary RR

Arbitrary RRs have been generated between 0 and 100% of available resources using random function. Table 4 shows that for Case 3, CBFFD has used 252 PMs followed by FFDDP, FFDAVG, FFDP, FFDSUM, which have used around 260 PMs followed by NBGA which has used 302 PMs which is followed by FFIP which has used 314 PMs. Talking of average number of comparisons, CBFFD has performed fastest with 60,983 comparisons followed by FFIP with 65,194 which is followed by FFDDP, FFDAVG, FFDP, FFDSUM with about 65,200 comparisons, which has been followed by NBGA with 69,075 comparisons.

Table 4 Simulation results for Case 3 (n = 500, arbitrary RR) (see online version for colours)

Heuristics	*Trial I*		*Trial II*		*Trial III*		*Trial IV*		*Trial V*		*Average*	
	No. of PMs used	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*Average no. of PMs used*	*Average no. of comparisons*
NBGA	305	69,458	291	67,905	308	69,675	297	67,264	307	71,075	302	69,075
FFDDP	262	64,787	259	64,483	257	65,350	251	63,289	271	68,740	260	65,330
FFDAVG	262	64,787	259	64,483	257	65,350	251	63,289	271	68,740	260	65,330
FFDP	263	64,774	261	64,082	258	65,355	252	63,268	271	68,989	261	65,294
FFDSUM	262	64,823	259	64,566	257	65,541	251	63,370	271	68,745	260	65,409
FFIP	310	65,412	316	63,780	312	66,280	316	63,809	318	66,687	314	65,194
CBFFD	254	60,057	251	61,209	250	61,845	257	59,355	251	62,450	252	60,983

Figure 6 and Table 4 show that CBFFD has performed 12% faster than NBGA and 8% faster as compared to remaining heuristics. FFIP has performed 6% faster as compared to NBGA and at par with remaining heuristics. FFDDP and FFDAVG have shown exactly same performance while FFDP and FFDSUM are almost near to them. Our CBFFD heuristics is best in this case. Figure 7 and Table 4 show that CBFFD has taken about 3% lesser number of PMs as compared to its best counterparts and 16% lesser number of PMs as compared to NBGA. FFDDP, FFDSUM and FFDAVG have taken same number of PMs.

Figure 6 No. of comparisons for Case 3 (n = 500, arbitrary RR) (see online version for colours)

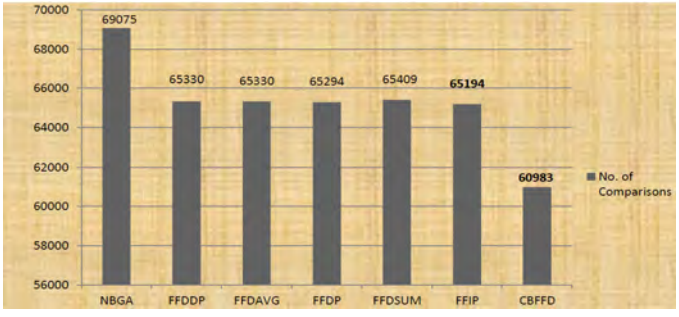
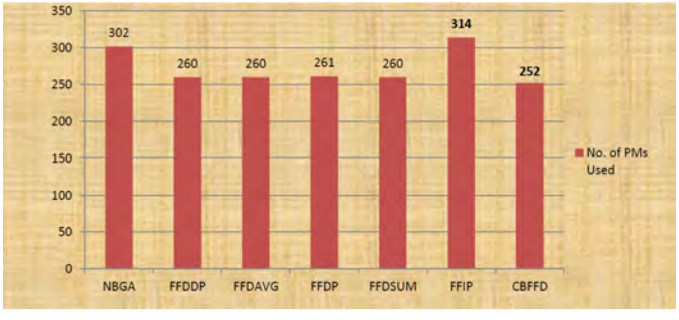


Figure 7 No. of PMs used for Case 3 (n = 500, arbitrary RR) (see online version for colours)



Case 4 No. of VMs = 1,000, arbitrary RR

Table 5 shows that CBFFD, taking 497 PMs, has performed more efficiently as compared to all other counterparts. Talking of speed of heuristic, CBFFD is on top with 243,234 comparisons as compared to average comparisons of about 257,000 of other heuristics.

Figure 8 and Table 5 show that CBFFD has performed 13% faster as compared to NBGA and 6% faster as compared to others in terms of number of comparisons. FFIP has performed about 1% slower as compared to its counterparts. FFDDP and FFDAVG have done same number of comparisons while FFDP and FFDSUM are approaching them. Figure 9 and Table 5 show that CBFFD has taken about 2.5% lesser number of PMs to allocate incoming VMs as compared to its best counterparts and about 16% lesser number of PMs as compared to NBGA. FFDDP and FFDAVG have used same number of PMs, while FFDP and FFDSUM are approaching them in terms of number of PMs used.

Table 5 Simulation results for Case 4 (n = 1,000, arbitrary RR)

Heuristics	*Trial I*		*Trial II*		*Trial III*		*Trial IV*		*Trial V*		*Average*	
	No. of PMs used	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*Average no. of PMs used*	*Average no. of comparisons*
NBGA	589	27,1059	599	27,9139	586	26,8309	604	27,4452	603	28,2623	596	27,5116
FFDDP	502	25,3655	513	25,9466	503	24,8840	509	25,6247	520	26,7048	509	25,7051
FFDAVG	502	25,3655	513	25,9466	503	24,8840	509	25,6247	520	26,7048	509	25,7051
FFDP	502	25,3406	515	25,8935	501	24,9365	511	25,5434	525	26,7080	511	25,6844
FFDSUM	503	25,3540	515	25,9437	504	24,8752	510	25,6196	521	26,6931	511	25,6971
FFIP	634	25,5579	638	26,1700	637	25,7283	638	25,8379	653	26,7600	640	26,0108
CBFFD	492	24,0055	497	24,8464	501	23,9805	506	24,0753	493	24,7093	497	24,3234

Figure 8 No. of comparisons for Case 4 (n = 1,000, arbitrary RR) (see online version for colours)

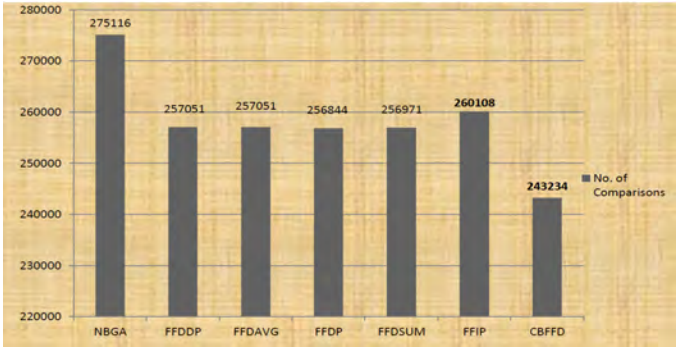
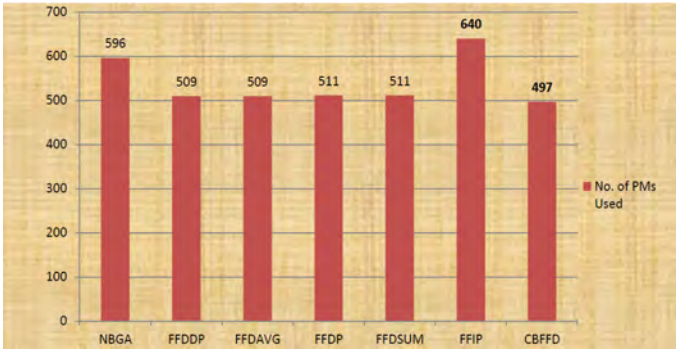


Figure 9 No. of PMs used for Case 4 (n = 1,000, arbitrary RR) (see online version for colours)



Case 5 No. of VMs = 500, 15% <= RR <= 60%

Table 6 shows that CBFFD has performed more efficiently than NBGA with CBFFD occupying 186 PMs while NBGA taking 218 PMs. It is also more efficient than others which are taking on average 191 PMs. In terms of number of comparisons FFIP and CBFFD have performed much faster as compared to their counterparts. FFIP has done 46,018 comparisons; CBFFD has performed 47,297 comparisons while on average the heuristics except NBGA have performed about 53,300 comparisons. NBGA has not performed well in this case, performing 57,200 iterations.

Figure 10 and Table 6 show that CBFFD has performed about 18% lesser number of comparisons as compared to NBGA and 12% lesser as compared to other counterparts. FFIP has performed about 20% lesser number of comparisons as compared to NBGA and 14% lesser as compared to its counterparts. FFDDP and FFDAVG have done same number of comparisons while FFDSUM is approaching them. FFDP has performed few more comparisons (about 0.5%) as compared to FFDP and FFDAVG. Figure 11 and Table 6 show that CBFFD has used about 16% lesser number of PMs than NBGA to allocate VMs and about 3% lesser number of PMs as compared to others. FFIP is at par with NBGA. FFDDP, FFDAVG, FDDP and FFDSUM have used same number of PMs to allocate VMs.

Table 6 Simulation results for Case 5 (n = 500, 15% <= RR <= 60%)

Heuristics	*Trial I*		*Trial II*		*Trial III*		*Trial IV*		*Trial V*		*Average*	
	No. of PMs used	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*Average no. of PMs used*	*Average no. of comparisons*
NBGA	217	56,235	223	58,010	211	56,603	217	56,921	224	58,234	218	57,200
FFDDP	187	53,098	194	53,687	188	53,050	190	52,813	195	54,332	191	53,396
FFDAVG	187	53,098	194	53,687	188	53,050	190	52,813	195	54,332	191	53,396
FFDP	189	53,019	192	52,863	187	52,607	192	52,841	195	54,067	191	53,079
FFDSUM	187	52,736	192	53,381	188	52,912	192	53,130	194	54,484	191	53,329
FFIP	217	45,527	220	47,031	213	45,029	212	45,409	223	47,096	217	46,018
CBFFD	187	47,104	183	47,715	189	47,120	183	46,787	190	47,761	186	47,297

Figure 10 No. of comparisons for Case 5 (n = 500, 15% <= RR <= 60%) (see online version for colours)

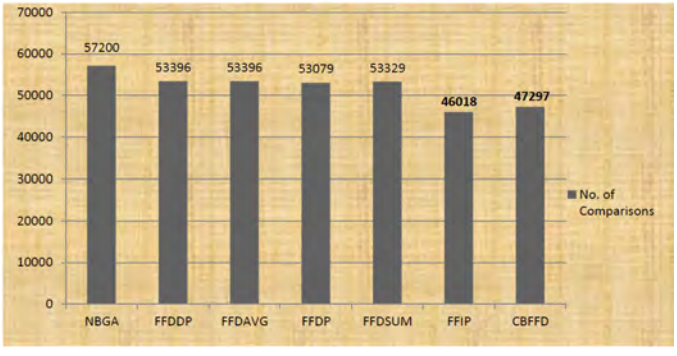
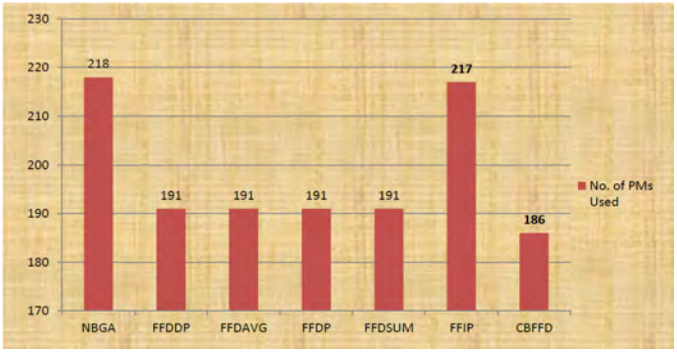


Figure 11 No. of PMs used for Case 5 (n = 500, 15% <= RR <= 60%) (see online version for colours)



Case 6 No. of VMs = 1,000, 15% <= RR <= 60%

Data in Table 7 shows that CBFFD has performed more efficiently than NBGA, with CBFFD taking 377 PMs and NBGA taking 429 PMs. Other heuristics on average have taken 379 PMs. Talking of number of comparisons, our both heuristics have performed much faster as compared to other heuristics with FFIP performing 180,829 comparisons and CBFFD performing 186,834 comparisons. Other heuristics, on average, have performed about 212,600 comparisons.

Figure 12 and Table 7 show that CBFFD has performed 18% lesser number of comparisons as compared to NBGA and 13% lesser number of comparisons as compared to its other counterparts. FFIP has performed 21% faster as compared to NBGA and 15% faster as compared to others. FFDDP and FFDAVG have done same number of comparisons while FFDP and FFDSUM are approaching them. Figure 13 and Table 7 show that CBFFD has taken about 12% lesser PMs as compared to NBGA and about 1% lesser number of PMs as compared to other counterparts. FFIP is at par with NBGA. FFDDP and FFDAVG have used same number of PMs, while FFDP and FFDSUM are approaching them in terms of number of PMs used.

Table 7 Simulation results for Case 6 (n = 1,000, 15% < = RR <= 60%)

Heuristics	*Trial I*		*Trial II*		*Trial III*		*Trial IV*		*Trial V*		*Average*	
	No. of PMs used	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*Average no. of PMs used*	*Average no. of comparisons*
NBGA	428	228,571	436	227,592	429	227,728	429	228,046	424	224,693	429	227,326
FFDDP	375	209,989	384	215,034	382	215,997	374	210,582	374	210,007	378	212,322
FFDAVG	375	209,989	384	215,034	382	215,997	374	210,582	374	210,007	378	212,322
FFDP	381	214,200	378	212,848	382	214,780	375	211,096	378	211,844	379	212,954
FFDSUM	381	213,188	381	213,866	383	214,785	376	211,033	377	211,527	380	212,880
FFIP	434	183,642	429	182,026	430	180,496	424	178,476	425	179,507	428	180,829
CBFFD	378	188,223	380	186,943	381	187,553	371	184,407	377	187,047	377	186,834

Figure 12 No. of comparisons for Case 6 (n = 1,000, 15% <= RR <= 60%) (see online version for colours)

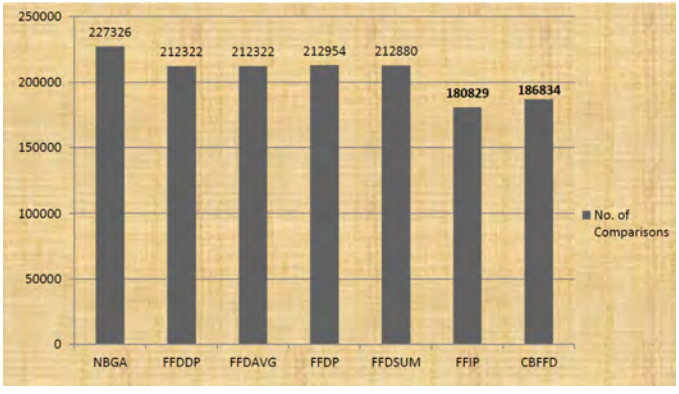
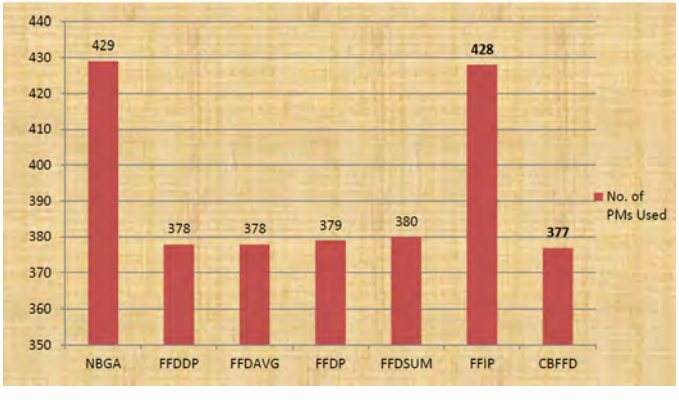


Figure 13 No. of PMs used for Case 6 (n = 1,000, 15% <= RR <= 60%) (see online version for colours)



Case 7 Simulation of various heuristics with workload traces from PlanetLab

Data in Table 8 shows that our both heuristics are much faster as compared to other heuristics with FFIP performing 38,648 comparisons and CBFFD performing 40,643 comparisons. Other heuristics, on average, have performed about 41,077 comparisons. Talking of number of PMs used, CBFFD is close to its counterparts.

Figure 14 and Table 8 show that CBFFD has performed about 3% lesser number of comparisons as compared to NBGA and about 2% lesser number of comparisons as compared to its other counterparts. FFIP has performed 7.22% faster as compared to NBGA and 6% faster as compared to others. FFDDP and FFDAVG have done same number of comparisons while FFDP and FFDSUM are approaching them. Figure 15 and Table 8 show that CBFFD has used same number of PMs as that of NBGA and is at par with others. FFIP has taken about 8% more PMs as compared to its counterparts.

Table 8 Simulation results for workload traces from PlanetLab

Heuristics	*Trial I*		*Trial II*		*Trial III*		*Trial IV*		*Trial V*		*Average*	
	No. of PMs used	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*No. of PMs used*	*No. of comparisons*	*Average no. of PMs used*	*Average no. of comparisons*
NBGA	157	41,601	157	41,551	155	41,291	156	41,765	166	41,843	159	41,610
FFDDP	154	40,804	155	40,869	153	40,558	153	40,639	161	41,678	156	40,909
FFDAVG	154	40,804	155	40,869	153	40,558	153	40,639	161	41,678	156	40,909
FFDP	157	41,101	155	40,671	155	40,790	154	40,523	161	41,701	157	40,957
FFDSUM	155	40,826	155	40,855	145	40,788	155	40,825	161	41,702	155	40,999
FFIP	168	38,209	170	38,314	166	37,950	168	38,246	180	40,522	170	38,648
CBFFD	158	40,409	158	40,689	157	40,333	157	40,296	167	41,489	159	40,643

Source: https://github.com/beloglazov/planetlab-workload-traces/tree/master/20110303

Figure 14 Number of comparisons for PlanetLab workload traces (see online version for colours)

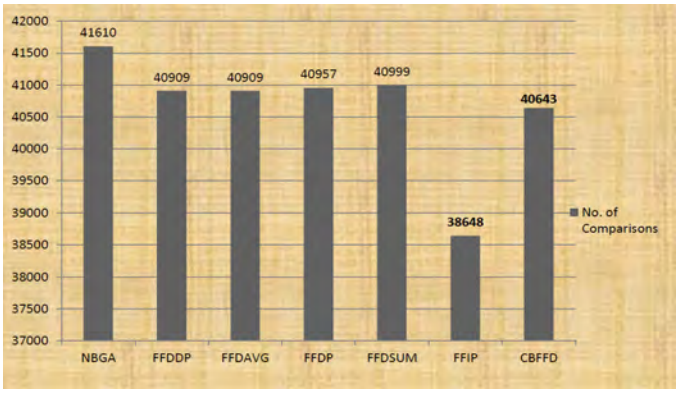
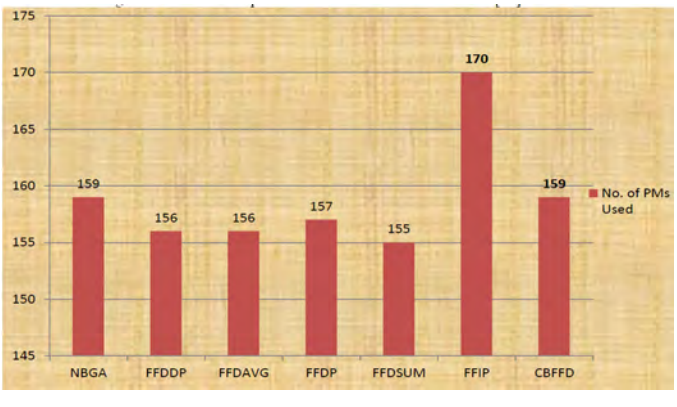


Figure 15 Number of PMs used for PlanetLab workload traces (see online version for colours)



9.2 Hotspot mitigation

We have run our algorithms in various scenarios. We have taken six cases with various ranges of demand and increase in demands. Though in MOM algorithm, single resource is taken but we have taken two resources in our algorithms. We have written an ad hoc simulator (*HotSimul*) in C language to analyse our algorithms and compared the results with MOM algorithm given by Beloglazov et al. (2016). *Hotsimul* has been designed to simulate cloud computing environment. Request for resources by VMs has been generated randomly in *HotSimul*. We have taken wide spectrum of requests for resources which match cloud environment. We have compared our algorithms with MOM and also compared our algorithm with variant of MOM (when taken two resources) (MOM1) (Beloglazov et al., 2016). In our simulator, we have assumed that eighty PMs are available which can be increased as needed.

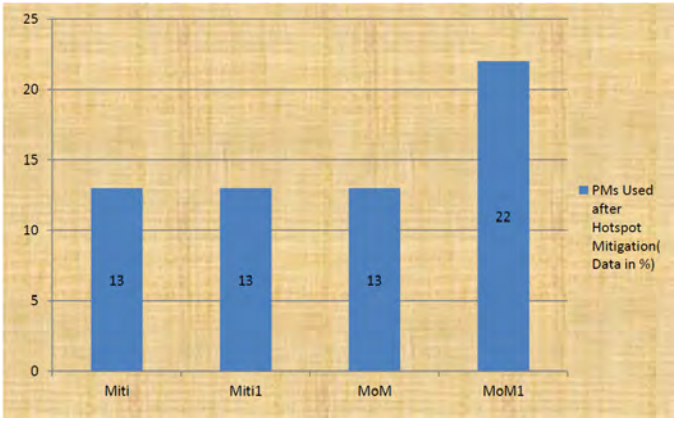
Case 1.1 Demand of resources (max 20% of total available resources) and increase in demand (max 20% of demand)

As Table 9 shows, in this case our both algorithms are performing better than their counterparts. In this case as Figure 16 and Table 9 show that before occurrence of hotspot average number of PMs used is 12%. After hotspot mitigation our first algorithm (Miti) has used average of 13% and second algorithm (Miti1) has used average of 13% while MOM and MOM1 have used 13% and 22% respectively.

Table 9 Comparison of our hotspot mitigation algorithms and others for Case 1.1

S. no.	*Comparisons*	*PM used*	*Miti*	*MOM*	*MOM1*	*Miti1*
1	365	9	10	10	16	9
2	332	8	9	9	16	9
3	337	8	10	10	16	10
4	356	9	10	10	18	10
5	364	9	10	10	17	10
6	312	7	8	8	14	8
7	351	8	9	9	16	9
8	333	8	9	9	16	9
9	345	9	11	11	18	10
10	322	8	9	10	16	9
11	360	9	10	10	16	9
12	341	8	11	11	18	10
13	368	9	10	10	16	10
14	334	8	9	9	16	9
15	362	9	10	10	18	10
16	377	9	10	10	17	10
	Average	8.43 = 12%	9.1 = 13%	9.75 = 13%	16.5 = 22%	9.43 = 13%

Figure 16 Number of PMs used after hotspot mitigation for Case 1.1 (see online version for colours)



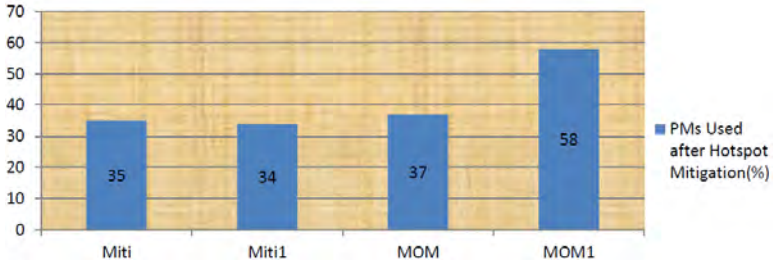
Case 1.2 Demand of resources (max 20% of total available resources) and increase in demand (max 50% of demand)

Figure 17 and Table 10 show that before occurrence of hotspot average number of PMs used is 27%. After hotspot mitigation our first algorithm (Miti) has used 35% of PMs and second algorithm (Miti1) has used 34% of PMs, while MOM and MOM1 have used 37% and 58% of PMs respectively. Miti algorithm has taken 2% less PMs than MOM and 23% less PMs than MOM1. Miti1 algorithm has taken 3% less PMs than MOM and 24% less PMs than MOM1.

Table 10 Comparison of our hotspot mitigation algorithms and others for Case 1.2

S. no.	*Comparisons*	*PM used*	*Miti*	*MOM*	*MOM1*	*Miti1*
1	882	19	25	26	41	23
2	768	18	23	24	39	22
3	819	19	25	26	41	24
4	884	21	29	30	48	27
5	857	21	29	30	47	28
6	848	20	27	28	46	26
7	953	21	27	29	46	28
8	773	18	24	24	40	22
9	981	22	30	32	51	29
10	843	20	25	26	44	25
11	943	22	30	32	50	30
12	931	21	28	29	46	26
13	985	23	32	33	51	31
14	912	21	30	31	47	29
15	883	19	25	26	43	23
16	829	19	26	27	43	25
Average		20.25 = 27%	27.18 = 35%	28.31 = 37%	45.18 = 58%	26.12 = 34%

Figure 17 Number of PMs used after hotspot mitigation for Case 1.2 (see online version for colours)



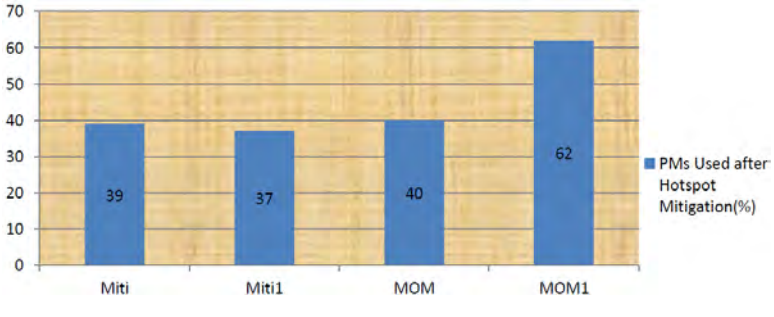
Case 1.3 Demand of resources (max 20% of total available resources) and increase in demand (max 80% of demand)

In this case as Figure 18 and Table 11 show that before occurrence of hotspot average number of PMs used is 27%. After hotspot mitigation our first algorithm has used 39% and second algorithm has used 37% of PMs while MOM and MOM1 have used 40% and 62% of PMs respectively. Our first algorithm (Miti) has taken 1% less PMs than MOM and 23% less PMs than MOM1. Our second algorithm (Miti1) has taken 3% less PMs than MOM and 25% less PMs than MOM1.

Table 11 Comparison of our hotspot mitigation algorithms and others for Case 1.3

S. no.	*Comparisons*	*PM used*	*Miti*	*MOM*	*MOM1*	*Miti1*
1	821	20	30	29	47	26
2	871	20	29	30	47	26
3	942	22	32	32	50	30
4	861	19	29	30	47	27
5	907	20	30	31	47	27
6	837	20	30	31	48	28
7	923	21	30	30	49	28
8	896	21	32	32	49	29
9	922	21	31	32	50	28
10	896	23	36	36	55	33
11	929	21	34	33	52	29
12	861	21	32	33	51	30
13	921	21	30	32	49	29
14	860	21	32	33	51	31
15	700	16	23	24	37	23
16	815	19	31	32	47	28
Average		20.37 = 27%	30.68 = 39%	31.25 = 40%	48.5 = 62%	28.25 = 37%

Figure 18 Number of PMs used after hotspot mitigation for Case 1.3 (see online version for colours)



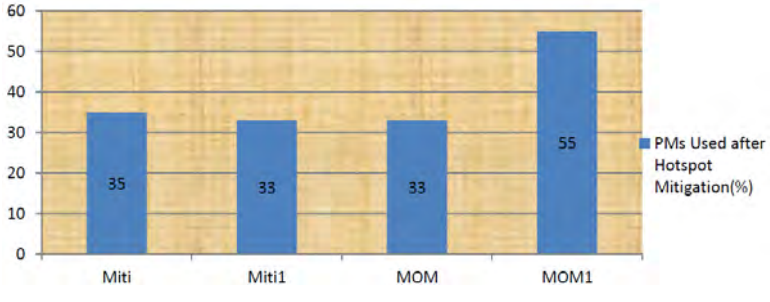
Case 2.1 Demand of resources (max 50% of total available resources) and increase in demand (max 20% of demand)

In this case as Figure 19 and Table 12 illustrate that before occurrence of hotspot, average number of PMs used is 28%. After hotspot mitigation our first algorithm has used 35% and second algorithm has used 33% of PMs, while MOM and MOM1 have used 33% and 55% of PMs respectively. Our first algorithm has taken 2% more PMs than MOM and 20% less PMs than MOM1. Our second algorithm has taken same number of PMs as MOM but 22% less PMs than MOM1.

Table 12 Comparison of our hotspot mitigation algorithms and others for Case 2.1

S. no.	*Comparisons*	*PM used*	*Miti*	*MOM*	*MOM1*	*Miti1*
1	777	21	25	26	42	25
2	863	23	28	28	44	27
3	850	23	32	31	46	27
4	792	22	24	25	41	25
5	846	23	28	29	44	27
6	770	20	25	27	41	25
7	908	24	31	32	47	30
8	815	22	26	27	41	25
9	799	22	28	27	41	25
10	812	22	29	27	43	26
11	929	23	27	27	46	25
12	841	22	27	28	46	26
13	784	20	27	26	41	23
14	809	20	30	27	42	25
15	906	23	28	28	48	26
16	710	19	24	26	39	23
Average		21.81 = 28%	27.43 = 35%	25.87 = 33%	43.25 = 55%	25.6 = 33%

Figure 19 Number of PMs used after hotspot mitigation for Case 2.1 (see online version for colours)



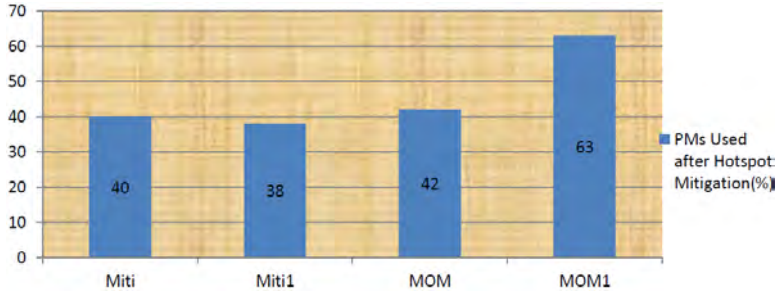
Case 2.2 Demand of resources (max 50% of total available resources) and increase in demand (max 50% of demand)

In this case as Figure 20 and Table 13 show that before occurrence of hotspot, average number of PMs used is 29%. After hotspot mitigation our first algorithm (Miti) has used 40% and second algorithm (Miti1) has used of 38% of PMs while MOM and MOM1 have used 42% and 63% of PMs respectively. In this case our first algorithm (Miti) has taken 2% less PMs than MOM and 23% less PMs than MOM1. Our second algorithm (Miti1) has used 4% less PMs than MOM and 25% less PMs than MOM1.

Table 13 Comparison of our hotspot mitigation algorithms and others for Case 2.2

S. no.	*Comparisons*	*PM used*	*Miti*	*MOM*	*MOM1*	*Miti1*
1	841	22	31	32	48	31
2	815	22	33	34	51	32
3	856	22	32	32	50	29
4	917	24	33	36	55	34
5	795	21	34	32	48	29
6	833	21	30	32	46	28
7	877	23	31	32	50	32
8	913	24	35	33	53	29
9	785	20	29	30	45	26
10	877	23	31	31	50	32
11	820	23	32	33	51	29
12	934	24	35	35	54	31
13	814	21	28	29	46	28
14	777	21	33	33	48	29
15	779	20	26	27	44	26
16	860	24	36	38	56	35
Average		22.18 = 29%	31.81 = 40%	32.43 = 42%	49.68 = 63%	30 = 38%

Figure 20 Number of PMs used after hotspot mitigation for Case 2.2 (see online version for colours)



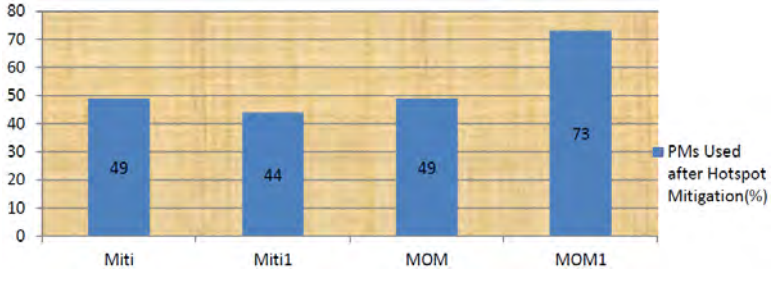
Case 2.3 Demand of resources (max 50% of total available resources) and increase in demand (max 80% of demand)

Figure 21 and Table 14 show that before occurrence of hotspot, average number of PMs used is 29%. After hotspot mitigation our first algorithm (Miti) has taken 49% and second algorithm (Miti1) has taken 44% of PMs, while MOM gives 49%, MOM1 gives 73%. In this case our first algorithm (Miti) has taken same number of PMs as MOM but 24% less PMs than MOM1. Second algorithm (Miti1) has taken 5% less PMs than MOM and 29% less PMs than MOM1.

Table 14 Comparison of our hotspot mitigation algorithms and others for Case 2.3

S. no.	*Comparisons*	*PM used*	*Miti*	*MOM*	*MOM1*	*Miti1*
1	809	22	38	38	57	36
2	875	23	42	40	60	35
3	869	22	38	36	56	33
4	836	21	35	36	55	34
5	732	20	36	36	51	31
6	817	22	37	38	57	35
7	821	22	39	38	58	35
8	886	23	35	36	57	34
9	855	22	40	40	56	33
10	858	22	40	38	58	33
11	945	24	40	41	62	38
12	896	23	39	40	59	38
13	873	24	43	42	62	38
14	844	23	40	41	59	37
15	920	24	41	41	61	36
16	770	21	37	37	54	34
Average		22.37 = 29%	38.75 = 49%	38.62 = 49%	57.6 = 73%	35 = 44%

Figure 21 Number of PMs used after hotspot mitigation for Case 2.3 (see online version for colours)



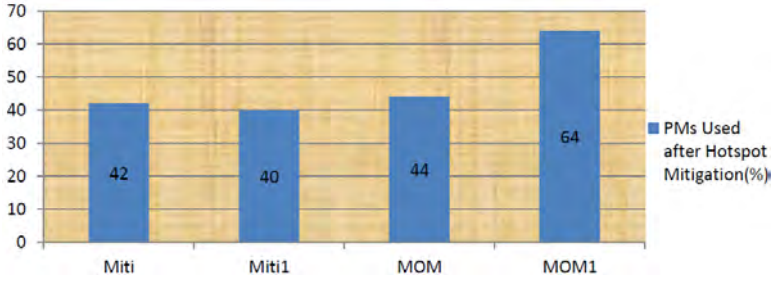
Case 3.1 Demand of resources (max 60% of total available resources) and increase in demand (max 20% of demand)

Figure 22 and Table 15 show that before occurrence of hotspot average number of PMs used is 34%. After hotspot mitigation our first algorithm (Miti) has taken 42% and second algorithm (Miti1) has taken 40% of PMs, while MOM and MOM1 have used 44% and 64% of PMs respectively. In this case our first algorithm (Miti) has taken 2% less PMs than MOM and 22% less PMs than MOM1. Our second algorithm (Miti1) has taken 4% less PMs than MOM and 24% less PMs than MOM1.

Table 15 Comparison of our hotspot mitigation algorithms and others for Case 3.1

S. no.	*Comparisons*	*PM used*	*Miti*	*MOM*	*MOM1*	*Miti1*
1	1,029	26	32	34	53	32
2	1,029	29	33	35	49	33
3	1,018	26	32	32	51	30
4	1,037	27	33	35	54	33
5	998	28	33	36	56	33
6	962	25	33	33	46	30
7	978	27	32	33	51	30
8	1,040	27	33	35	52	32
9	944	25	29	31	45	29
10	944	25	31	32	49	28
11	1,055	28	33	35	54	32
12	1,042	30	35	37	54	35
13	1,000	28	36	35	53	34
14	941	27	31	34	51	30
15	937	26	36	34	48	31
16	1,101	28	34	35	53	33
Average		27 = 34%	32.87 = 42%	34.12 = 44%	51.18 = 64%	31.56 = 40%

Figure 22 Number of PMs used after hotspot mitigation for Case 3.1 (see online version for colours)



Case 3.2 Demand of resources (max 60% of total available resources) and increase in demand (max 50% of demand)

In this case as Figure 23 and Table 16 show that before occurrence of hotspot average number of PMs used is 34%. After hotspot mitigation our first algorithm (Miti) has used 53% and second algorithm (Miti1) has used 48% of PMs, while MOM and MOM1 have used 54% and 79% of PMs respectively. Our first algorithm (Miti) has used 1% less PMs than MOM and 26% less PMs than MOM1. Our second algorithm (Miti1) has used 6% less PMs than MOM and 31% less PMs than MOM1.

Table 16 Comparison of our hotspot mitigation algorithms and others for Case 3.2

S. no.	*Comparisons*	*PM used*	*Miti*	*MOM*	*MOM1*	*Miti1*
1	998	25	36	38	57	35
2	997	26	40	42	62	36
3	1006	27	43	43	66	38
4	963	26	46	41	61	37
5	1040	29	46	47	69	43
6	1073	28	40	43	62	39
7	923	26	40	42	61	38
8	1035	27	40	41	62	37
9	1043	27	40	44	63	39
10	1045	30	53	48	68	44
11	1089	29	50	49	68	41
12	906	25	37	38	58	33
13	983	24	33	35	53	31
14	1059	28	50	44	67	39
15	1100	29	42	43	67	42
16	1039	25	36	37	58	33
Average		26.92 = 34%	42 = 53%	42.18 = 54%	62.62 = 79%	37.81 = 48%

Figure 23 Number of PMs used after hotspot mitigation for Case 3.2 (see online version for colours)

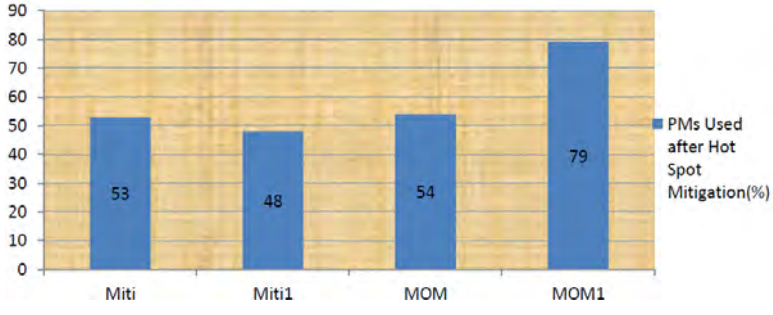


Table 17 Comparison of our hotspot mitigation algorithms and others for Case 3.3

S. no.	*VM placed*	*No. of comparisons*	*PM used*	*Miti*		*MOM*		*MOM1*		*Miti1*	
				PM used	*VM placed*	*PM used*	*VM placed*	*PM used*	*VM placed*	*PM used*	*VM placed*
1	80	1,010	28	53	79	47	79	67	79	40	80
2	80	1,042	27	46	80	43	80	65	80	39	80
3	80	1,081	30	45	77	46	77	68	77	41	79
4	80	1,141	29	59	78	52	78	75	78	43	80
5	80	1,040	28	51	79	49	79	74	79	45	80
6	80	1,058	29	50	77	46	77	72	77	42	78
7	80	1,013	27	49	79	46	79	69	79	41	80
8	80	1,136	30	57	79	51	79	74	79	44	80
9	80	979	27	47	79	46	79	72	79	41	80
10	80	1,008	28	49	78	44	78	67	78	40	79
11	80	1,062	28	51	79	49	79	68	79	41	80
12	80	966	25	47	78	45	78	63	78	38	80
13	80	955	26	51	79	45	79	67	79	38	80
14	80	983	26	41	76	43	76	64	76	37	78
15	80	1,042	28	50	78	48	78	67	78	44	80
16	80	1,027	28	49	79	43	79	66	79	40	80
Average			27.75 = 35%	49.68 = 63%	78.37 = 97.5%	46.43 = 59%	78.37 = 97.5%	64.68 = 81%	78.37 = 97.5%	40.87 = 52%	79.62 = 98.75%

Case 3.3 Demand of resources (max 60% of total available resources) and increase in demand (max 80% of demand)

In this case as Figure 24 and Table 17 show that after occurrence of hotspot, average number of PMs used is 35%. After hotspot mitigation our first algorithm has used 63% PMs and 97.5% of VMs have been packed in available PMs and second algorithm has used 52% PMs and 98.75% of VMs have been packed in available PMs while MOM1 has used 59% PMs and 97.5% of VMs have been packed in available PMs, MOM2 has used 81% PMs and 97.5% of VMs have been packed in available PMs. Our first algorithm (Miti) has taken 4% more PMs than MOM and 18% less PMs than MOM1. Our second algorithm (Miti1) has taken 7% less PMs than MOM and 29% less PMs than MOM1. Talking about number of VMs placed after migration, Figure 25 and Table 17 show that our second algorithm (Miti1) has placed 1% more VMs. Our both algorithm have performed better than MOM and MOM1. Our second algorithm is better than remaining all algorithms in all cases. Our first algorithm is better than MOM and MOM1 in all cases except Cases 2.1 and 3.3.

Figure 24 Number of PMs used after hotspot mitigation for Case 3.3 (see online version for colours)

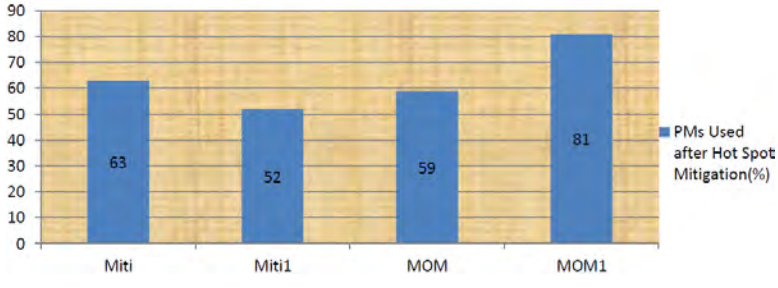
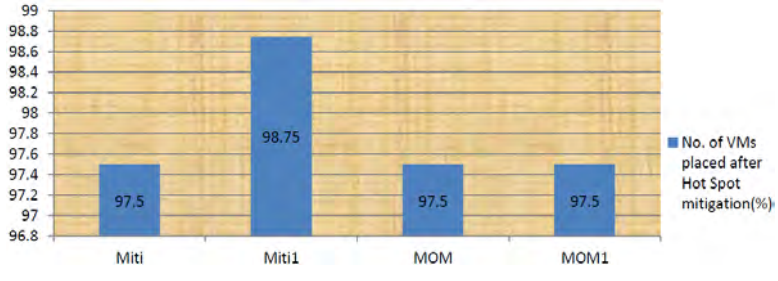


Figure 25 Number of VMs placed after hotspot mitigation for Case 3.3 (see online version for colours)



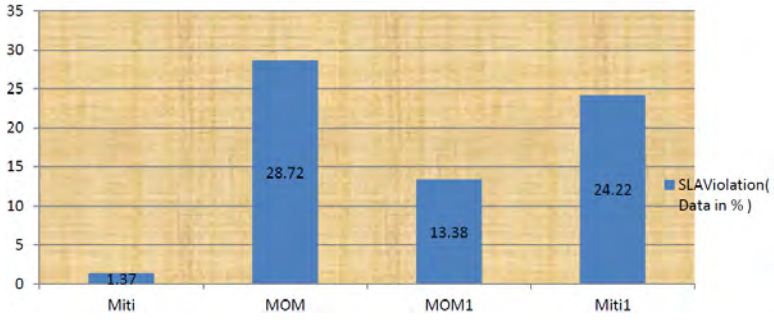
9.3 SLA violation

Here we discuss the SLA violation caused by these algorithms in simulation. Results obtained after simulation are presented in percentage in Table 18 and displayed in Figure 26. Table 18 and Figure 26 show that Miti algorithm has caused 1.37% of SLA violation which is least among all algorithms. Miti1 has caused 4.50% less SLA violation than MOM but 10.84% more than MOM1.

Table 18 SLA violation in different algorithms

S. no.	*Miti (%)*	*MOM (%)*	*MOM1 (%)*	*Miti1(%)*
1	1.25	28.50	13.50	24.255
2	1.50	29.75	12.75	26.50
3	1.25	28.50	14.75	22.25
4	1.25	29.50	12.50	19.75
5	1.50	27.75	14.50	23.50
6	1.60	26.50	12.50	25.50
7	1.25	28.75	13.75	26.25
8	1.25	29.75	12.75	25.25
9	1.50	29.50	13.50	24.75
	Average = 1.37%	Average = 28.72%	Average = 13.38%	Average = 24.22%

Figure 26 SLA violation in different algorithms (see online version for colours)



Our hotspot mitigation algorithms (Miti and Miti1) try to use minimum number of PMs after mitigation to make mitigation process energy efficient. During hotspot mitigation our algorithms try to minimise number of PMs used after hotspot mitigation. Both algorithms have shown promising results in direction of energy efficiency and also in establishing acceptability of cloud computing paradigm.

10 Summary

As clouds have strong business perspective also, our heuristics has been tested as a prime alternate options for VMs to PMs mapping Heuristics in proposed work reduces the number of PMs used, thus increase ROI for service provider. Our FFIP heuristic performed remarkably well when RR is low. It is about 32% faster as compared to others and at par with others in terms of PMs used. Our second heuristics, CBFFD remained best throughout simulation. It is 24% faster for low RR and 10% faster for average and arbitrary RRs while taking 2–3% lesser number of PMs as compared to its best counterparts. CBFFD is about 15% more efficient as compared to NBGA in term of number of PMs used. Among other heuristics, FFDDP and FFDAVG performed about 0.5–1% lesser comparisons but used 0.5–1% more number of PMs as compared to others. For average load these are 14–15% faster but use more number of PMs. CBFFD and

FFIP can be seen as promising candidates for all bin packing problems and for allocating VMs to PMs in energy efficient manner in cloud computing environment. For low RR, NBGA performed 2% lesser comparisons but it used 12% more PMs as compared to its counterparts. Our hotspot mitigation algorithm (Miti) is taking 1.42% less PMs than MOM algorithm and 23% less PMs than MOM1 algorithm after hotspot mitigation. Second algorithm (Miti1) is taking 3.57% less PMs than MOM and 25.71% less PMs than MOM1 after hotspot mitigation. In all cases except last case all VMs were placed after hotspot mitigation. In our last case when input is 60% (max) and increase in demand is 80% (max), then our second algorithm (Miti1) placed 98.75% of VMs as compared to 97.5% for other algorithm after hotspot mitigation.

11 Future work

Proposed heuristics would perform with greater efficiency if it were able to predict live incoming workload (demand of various resources). For that we could train our heuristics using machine learning techniques like decision tree, neural networks.

References

[online] https://github.com/beloglazov/planetlab-workload-traces/tree/master/20110303 (accessed 11 December 2017).

Beloglazov, A., Abawajy, J. and Buyya, R. (2016) 'Energy aware resource allocation heuristics for efficient management of data centers for cloud computing', *Future Generation Computer Systems*, Vol. 11, No. 4, pp.755–768, Elsevier.

Beloglazov, A., Buyya, R., Lee, Y.C. and Zomaya, A. (2015) 'A taxonomy and survey of energy efficient datacenters and cloud computing systems', in *Advances in Computers*, Vol. 82, No. 3, pp.78–91, Elsevier.

Celesti, A., Galletta, A., Fazio, M. and Villar, M. (2019) 'Towards hybrid multi-cloud storage systems: understanding how to perform data transfer', *FGCC*, Vol. 16, No. 2, pp.1–17, Elsevier.

Ceselli, A., Premoli, M. and Secci, S. (2017) 'Mobile edge cloud network design optimization', *IEEE/ACM Trans. Netw*., June, Vol. 25, No. 3, pp.1818–1831.

Chen, X. (2019) 'Decentralized computation offloading game for mobile cloud computing', *IEEE Trans. Parallel Distrib. Syst*., March, Vol. 26, No. 4, pp.974–983.

Dayarathna, M., Wen, Y. and Fan, R. (2016) 'Datacenter energy consumption modeling: a survey', *IEEE Communications Surveys & Tutorials*, Vol. 18, No. 1, pp.732–794.

Farahnakian, F. and Pahikkala, T. (2019) 'Energy-aware VM consolidation in cloud data centers using utilization prediction model', *IEEE Digital Explore*, Vol. 43, No. 5, pp.524–536.

Gai, K., Qiu, M., Zhao, H., Tao, L. and Zong, Z. (2019) 'Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing', *J. Netw. Comput. Appl*., Vol. 59, No. 3, pp.46–54.

Gao, F., Thiebes, S. and Sunyaev, A. (2018) 'Rethinking the meaning of cloud computing for health care: a taxonomic perspective and future research directions', *J. Med. Internet Res.*, Vol. 20, No. 7, pp.112–119.

Gergo, L., Niedermeier, F. and Meer, H.D. (2012) *Performance Tradeoffs of Energy-Aware Virtual Machine Consolidation*, p.20, Springer Science + Business Media, [online] http://www.zdnet.com/blog/btl/cloud-computing-market-241-billion-in-2020/47702 (accessed 3 September 2017).

Jia, M., Cao, J. and Liang, W. (2017) 'Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks', *IEEE Trans. Cloud Comput.*, October–December, Vol. 5, No. 4, pp.725–737.

Jia, M., Cao, J. and Liang, W. (2018) 'Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks', *IEEE Trans. Cloud Comput.*, Vol. 39, No. 7, p.1.

Li, Y., Dai, W., Ming, Z. and Qiu, M. (2019) 'Privacy protection for preventing data over-collection in smart city', *IEEE Trans. Comput.*, May, Vol. 65, No. 5, pp.1339–1350.

Madani, S. and Jamali, S. (2018) 'A comparative study of fault tolerance techniques in cloud computing', *International Journal of Research in Computer Applications and Robotics*, March, Vol. 6, No. 3, pp.7–15.

Pedram, M. (2012) 'Energy efficient datacenters', *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 31, No. 10, pp.46–59.

Persico, V., Pescape, A., Picariello, A. and Sperli, G. (2018) 'Benchmarking big data architectures for social networks data processing using public cloud platforms', *FGCS*, Vol. 89, No. 9, pp.98–109, Elsevier.

Qi, Q. and Tao, F. (2019) 'A smart manufacturing service system based on edge computing, fog computing and cloud computing', *IEEE Digital Explore*, Vol. 27, DOI: 10.1109/ACCESS. 2019.2923610.

Rastegar, F., Dehghan, M. and Fooladi, T. (2019) 'Online virtual machine assignment using multi-armed bandit in cloud computing', *International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), IEEE Digital Explore*, DOI: 10.1109/ COMITCon.2019.8862268.

Son, J. and Buyya, R. (2019) 'SDCon: integrated control platform for software – defined clouds', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 30, No. 1, pp.231–243.

Song, W., Xiao, Z., Chen, Q. and Luo, H. (2019) 'Adaptive resource provisioning for the cloud using online bin packing', in *IEEE Transactions on Computers*, Vol. 63, No. 11, pp.145–157.

Sotiridis, S., Bessis, N. and Buyya, R. (2018) 'Self managed virtual machine scheduling in cloud systems', *Information Sciences*, Vol. 23, No. 3, pp.381–400, Elsevier.

Vasic, N. and Kostic, D. (2016) 'Energy-aware traffic engineering', in *Proceedings of the 1st ACM International Conference on Energy-Efficient Computing and Networking (e-Energy 2010)*, Passau, Germany, pp.169–178.

Xiao, Z., Song, W. and Chen, Q. (2015) 'Dynamic resource allocation using virtual machines for cloud computing environment', in *IEEE Transaction on Parallel and Distributed Systems*, Vol. 24, No. 6, pp.89–102.

Xiong, Y., Huang, S., She, M.J. and Jiang, K. (2019) 'A Johnson's-rule-based genetic algorithm for two-stage-task scheduling problem in data-centers of cloud computing', *IEEE Transactions on Cloud Computing*, Vol. 7, No. 3, pp.597–610.

Xu, Z., Liang, W., Xu, W., Jia, M. and Guo, S. (2016) 'Efficient algorithms for capacitated cloudlet placements', *IEEE Trans. Parallel Distrib. Syst.*, October, Vol. 27, No. 10, pp.2866–2880.

Yao, D. and Yu, C. (2019) 'Using crowd sourcing to provide QoS for mobile cloud computing', *IEEE Transactions on Cloud Computing*, Vol. 7, No. 2, pp.123–139.

Yin, Z., Chen, H. and Hu, F. (2019) 'An advance decision model enabling two way initiative offloading in edge computing', *FGCS*, Vol. 90, No. 5, pp.39–48, Elsevier.

Zhang, X., Hu, Z., Zheng, M. and Liuyang, J. (2019) 'A novel cloud model based data placement strategy for data-intensive application in clouds', *FGCC*, Vol. 70, No. 11, pp.445–456, Elsevier.