International Journal of Cloud Computing and Services Science (IJ-CLOSER)
Vol.3, No.1, February 2014, pp. 1~25
ISSN: 2089-3337

Enhancing Cloud Resource Utilisation using Statistical Analysis

Sijin He*, Li Guo, Yike Guo***
* Departement of Computing, Imperial College London
** Departement of Computing, University of Central Lancashire

Article Info

Article history:

Received Nov 12th, 2013
Revised Dec 25th, 2013
Accepted Jan 26th, 2014

Keyword:

Algorithms
Cloud Computing
Heuristics
Resource Allocation
Resource Management

ABSTRACT

Resource provisioning based on virtual machine (VM) has been widely accepted and adopted in cloud computing environments. A key problem resulting from using static scheduling approaches for allocating VMs on different physical machines (PMs) is that resources tend to be not fully utilised. Although some existing cloud reconfiguration algorithms have been developed to address the problem, they normally result in high migration costs and low resource utilisation due to ignoring the multi-dimensional characteristics of VMs and PMs. In this paper we present and evaluate a new algorithm for improving resource utilisation for cloud providers. By using a multivariate probabilistic model, our algorithm selects suitable PMs for VM re-allocation which are then used to generate a reconfiguration plan. We also describe two heuristics metrics which can be used in the algorithm to capture the multi-dimensional characteristics of VMs and PMs. By combining these two heuristics metrics in our experiments, we observed that our approach improves the resource utilisation level by around 8% for cloud providers, such as IC Cloud, which accept user-defined VM configurations and 14% for providers, such as Amazon EC2, which only provide limited types of VM configurations.

Corresponding Author:

Yike Guo,
Departement of Computing,
Imperial College London,
South Kensington, London, SW7 2AZ, UK.
Email: yg@imperial.ac.uk

1. INTRODUCTION

Driven by the rapid growth of the demand for efficient and economical computational power, cloud computing [1], has led the world into a new era. By using virtual machines (VMs), cloud providers deliver virtual computational resources, such as computational power, storage space and network, in such a way that cloud users are able to consume them over the Internet as services. For example, many large cloud providers such as Amazon EC2 [2], GoGrid [3] have made great success, which encourage more traditional data centres to get involved into this business. However, unlike those giant providers, small and medium sized cloud infrastructure providers still find difficulties in answering some of the critical questions for their cloud resource management mechanisms. They, in particular, normally have limited resources and are short of proper knowledge for better resource scheduling in order to reduce their operating costs that is crucial to their business success. Therefore, properly scheduling cloud resource definitely helps small and medium sized cloud infrastructure providers to make great success [4] as the virtualisation technology coupled with cloud reconfiguration algorithms enables more efficient cloud resource utilisation in Internet Data Centres.

Resource scheduling has been widely studied and adopted in the cloud infrastructure [5, 6]. However, static resource scheduling results in inefficient resource utilisation in the cloud infrastructure. For example, VM placement is a static resource scheduling approach that allocates newly created VMs to physical machines (PMs) in the cloud infrastructure normally lead to PM resources not efficiently utilised. This is because there are always some resources left in the PMs, such as CPU and memory resources, etc.

The remaining resources normally do not meet VM resource requirements from the cloud users. In the other words, we consider the remaining resources as non-utilisable resources. Therefore, cloud reconfiguration algorithm is one of the dynamic resource scheduling approaches that has been developed to address this problem and try to increase the amount of utilisable resource in the Cloud infrastructure. However, the cloud reconfiguration algorithm normally results in high migration costs due to ignoring the multi-dimensional characteristics of VMs and PMs.

In this chapter, we firstly describe two types of cloud resource scheduling approaches and discuss their advantages and disadvantages. We then present and evaluate a new cloud reconfiguration algorithm for improving resource utilisation for the cloud infrastructure by considering the multi-dimensional characteristics of logical machines. By using a multivariate probabilistic model, the algorithm selects a subset of PMs for VM re-allocation that are then used to generate a reconfiguration plan. We then present two heuristics metrics that exhibit the multi-dimensional characteristics of VMs and PMs. By combining these two heuristics metrics in our experiments, we observed that our approach improves the utilisable resource level by around 8% for providers, such as IC Cloud, which accept user-defined VM configurations and 14% for providers, such as Amazon EC2, which only provide limited types of VM configurations.

2. CLOUD RESOURCE SCHEDULING APPROACHES

In order to drive the cloud providers to success, cloud resource scheduling is one of the most important and essential VM cloud resource management mechanisms that must be handled accurately and effectively. For example, it allocates a newly created VM to one of the PMs in the data centre, migrates VMs from one PM to another in order to avoid hotspot or improve resource utilisation in the cloud infrastructure. In general, there are two types of cloud resource scheduling approaches: static resource scheduling approach and dynamic resource scheduling approach.

Static Resource Scheduling Approach

The static resource scheduling approach is an approach that allocates cloud resources without resource re-allocation. For example, VM placement that allocates a newly created VM to one of the PMs in the cloud infrastructure, this operation does not involve any VM re-allocation once a VM is allocated.

Dynamic Resource Scheduling Approach

The dynamic resource scheduling approach is an approach that re-allocate cloud resources according to the current cloud resource utilisation. For example, VM migration is used to avoid hotspot or improve resource utilisation in the cloud infrastructure by re-allocating VMs to other PMs. Server consolidation is a way to reduce energy consumption in data centres in order to achieve the goal of green computing. This approach re-allocates all VMs in a PM to other PMs so as to safely shut down the PM.

2.1. Static Resource Scheduling Approach

As we mention earlier, VM placement is a static resource scheduling approach that allocates a newly VM to PMs in the cloud infrastructure using a resource scheduler without VM re-allocation. For example, the resource utilisation of PMs in the cloud infrastructure is determined at cloud user request submission time. When a request of VM placement is accepted, the resource scheduler monitors the current resource utilisation in PMs and determines the best location for a newly created VM to be allocated among those PMs as shown in Figure 1. Once the newly created VM was allocated, it would not be re-allocated.

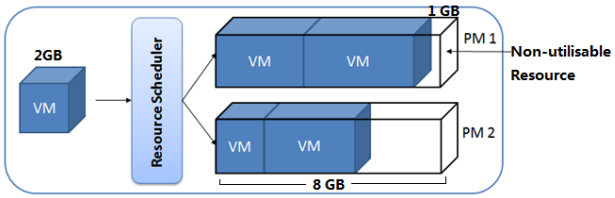

Figure 1. A typical example of VM placement

However, the static resource scheduling approach has the following drawbacks:

- The static resource scheduling approach tends to produce more remaining resources in PMs as VM resource requirements of cloud users are unpredictable, and the remaining resources in some of the PMs could not meet normal VM resource requirements from cloud users. This leads to new VMs could not be allocated to some of the PMs. Therefore, the resources

in some PMs become non-utilisable. For example as shown in Figure 1, there are two PMs with 8 GB memory respectively and hosting a few number of VMs. We can easily see that there are some remaining resources (1 GB memory) in PM 1. The remaining resource of PM 1 could not be used for hosting a new incoming VM with 2 GB memories, as the VM requires more memory resource (i.e. 1 GB more) so as to be allocated to the remaining resource in PM 1. Therefore, the new VM will be allocated to PM 2 where there is enough memory to meet the VM resource requirement. The remaining resources in PM 1, however, unlikely to meet VM resource requirements of cloud users. It will then be considered as wasted resource, or non-utilisable resource.

Figure 2. A typical example of memory balloon

- Memory balloon and CPU virtualisation are widely supported features of virtualisation technology. Memory balloon is a virtual memory management technique used to free unused memory. Having multiple VMs on a single PM requires virtual memory management techniques to control resource sharing and to prevent shortage. The use of CPU virtualisation is to allow two separate computers running on a single PM.
  These virtualisation technologies enable cloud users to configure their resource requirements, such as VM memory size [7] and CPU number, at operation time, which makes the resource scheduler work less efficient. For instance as shown in Figure 2, at a request time, a cloud user submits a VM resource requirement with 2 GB memory to a PM with 7 GB memory but he then changes the memory size to 4 GB later. If many users on a same PM are trying to execute the same operation, some of their requests may fail due to the unavailability of the resources in the PM, in this example, another VM also requires 4 GB memories, but the request will be failed due to unavailability of physical resources in the PM.

These two reasons make the static resource scheduling approach unfit, and instead, requiring dynamic re-allocation. Therefore, a dynamic VM re-allocation on the PMs is required for improving utilisable resource in the cloud infrastructure. This is referred as the dynamic resource scheduling approach.

2.2 Dynamic Resource Scheduling Approach

Due to the nature of elasticity of cloud resource provision, only applying the static resource scheduling approach cannot optimally improve the resource utilisation in the cloud infrastructure. Some cloud infrastructure providers are tackling this problem using resource reservation approach [8]. They require cloud users to setup a lower and an upper bound of the requested resource in the SLA in order to manage cloud resource scheduling properly. For example, a cloud user can set its minimum memory size to be 1 GB and the maximum memory size to be 4 GB. However, in the worst scenario case, i.e. in a limited resource environment, this would cause a huge amount of resource being wasted. This is because the cloud user may scale the resources down to the lower bound (i.e. 1 GB memory) for saving cost during the operation time. Even though the unused resources were released, the cloud infrastructure provider could not re-allocate the resource to other cloud users as the released resource are reserved so as to guarantee the cloud user may

require some extra resources up to the upper bound (i.e. 4 GB memory) of the resource stated in the SLA in the future. This is a common problem for those small and medium sized cloud infrastructure providers as they have limited cloud resource.

We can clearly see that, in order to efficiently utilise the PM resources without affecting cloud users' normal operations, runtime resource re-allocation is required. We call this runtime resource re-allocation as the dynamic resource scheduling approach.

There are two cases that occur in a cloud infrastructure provider's day-to-day operation where resource re-allocation needs to be carried out:

Use Case 1

We assume the cloud infrastructure provider only has two PMs (PM 1 and PM 2) and each has 20 GB memory, running five VMs in total as shown in Figure 3. PM 1 hosts 4 VMs (from VM 1 to VM 4), each with 2 GB memories. VM 5 with 12 GB memories is running on PM 2. If a cloud user submits a request for 16 GB memories, there is no sufficient memory for it on either PMs and therefore resource re-allocation needs to be executed.

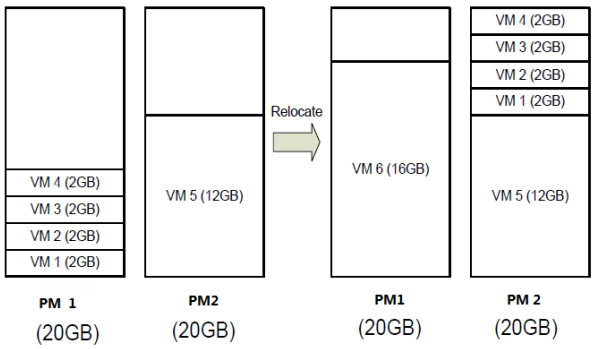

Figure 3. Resource Re-allocation for Two PMs

Use Case 2

Multiple VMs are located on the same PM trying to extend their resources to larger scales. As a result, the sum of the requested resources exceeds the capacity of the underlying PM. Resource, therefore, needs to be re-allocated. For example, we assume that there are just two PMs with 6 GB respectively in a data centre as shown in Figure 4. They both have memory capacity of 6 GB. Two VMs on PM 1 occupy 2 GB memory and three VMs on PM 2 occupy 4 GB memory. VM 1 and VM 2 on PM 2 are both required to scale up their memory to 3 GB. However, scaling up the memory for both VM 1 and VM 2 at the same time will cause the occupied memory reaching 7 GB that exceeds the 6 GB memory capacities on the PM 2. The only possible solution is to move either VM 1 or VM 2 to the PM 1 where there is enough free memory space (4 GB) for either VM 1 or VM 2 to scale up its memory.

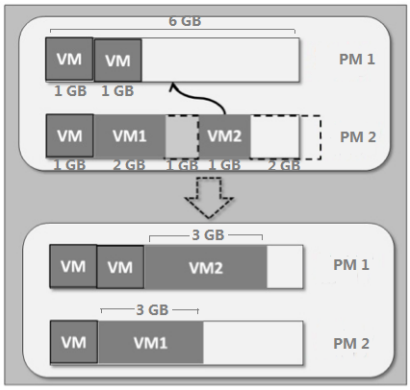

Figure 4. Resource Re-allocation for VM adjustment

Both use cases require VMs to be moved around in order to improve the utilisable resource in the cloud environment. Fortunately VM migration provides us a good foundation for such a dynamic resource management mechanism, i.e. static and live migrations, where static migration is a migration that requires a

VM shutdown before VM re-allocation, whilst live migration is a migration that enables on-the-fly VM re-allocation.

3. MULTI-DIMENSIONAL CHARACTERISTICS

Dynamic resource provisioning has been widely accepted and adopted in the cloud infrastructure. As we mention earlier, static resource scheduling leads to non-utilisable resources in PMs. Even though some existing cloud reconfiguration algorithms have been developed to address this problem and try to improve resource utilisation in the cloud infrastructure, they normally result in high migration costs and low resource utilisation in the cloud environment due to ignoring the multi-dimensional characteristics of VMs and PMs.

3.1. VM Configuration

In generally, a VM configuration is defined by cloud users who specify their required amount of resources, i.e. VM resource requirements, such as number of CPUs and memory sizes, etc.
Nowadays, there are two ways to define VM configurations offered by different cloud infrastructure providers. The VM configuration offerings can be classified as provider-defined configuration and user-defined configuration respectively.

Provider-defined configuration

For the provider-defined configuration, cloud infrastructure providers, such as Amazon EC2, GoGrid, predefine limited number of types of VM configurations for cloud users. For example, Amazon EC2 offers 10 types of VM configurations for cloud users to choose.

User-defined configuration

For the user-defined configuration, Cloud infrastructure providers, such as IC Cloud [9], allow their cloud users to define VM configurations that meet their needs.

After a VM configuration is decided by a cloud user, it will be sent to a cloud infrastructure provider for approval. If the requested resources are available in the cloud infrastructure, the request will be accepted and then the cloud infrastructure provider will create and allocate the requested VM to a suitable PM. We refer this as VM placement. This strategy ensures that all VMs in the cloud infrastructure do not compete with each other in the same PM in a way that they have sufficient CPU and memory resources available as requested by cloud users.

3.2. Vector Representation

In a data centre, there are a set of m number of PMs, $PM = \{PM_0, \dots, PM_i, \dots, PM_{m-1}\}$ with $m = |PM|$, each of which hosts n number of VMs, $PM_i = \{VM_0, \dots, VM_m\}$. In addition, there are d types of resources in a PM, for instance, CPU cycles, CPU cores, RAM size, network bandwidth and disk size, etc.
A VM-to-PM mapping is a mapping between VMs and their hosting PMs, i.e. the allocation of VMs to its hosting PM. In order to achieve better resource utilisation in the cloud infrastructure, a good VM-to-PM mapping is essential. The problem of creating a good VM-to-PM mapping is an instance of the MDBP problem [10, 11], in which the PMs represent the bins and VM the items to be packed. However, it is well known that finding optimal solutions to MDBP problem is NP-hard. Hermenier et. al. [12] model the MDBP problem as a Constraint Satisfaction Problem (CSP) which can determine a globally optimal solution but it is computationally expensive. Therefore, they impose a time limit for the computation that could lead to the output being not as good as heuristic approaches. Therefore, various heuristic approaches have been developed to provide good results in a reasonable amount of time. Polynomial time approximate solutions (PTAS) [10] is proposed to solve the problem with a low approximation ratio. The static re-allocation approach [13] is a simple heuristic for the MDBP problem and applies it to minimise the number of PMs required to host a given web traffic. Nathuji et. al. have applied a First-Fit Decreasing (FFD) heuristic for MDBP problem with variable bin sizes and costs and have introduced the notion of cost of VM live migration, but the information about the cost calculation is not provided. It is important to note that the problem of minimising migration costs during re-allocation is still an open research problem.

Representing the quantities of various resources as vectors is a traditional approach for VM resource management [14, 15]. As each VM is actually a combination of different types of resources, it can be represented as a d-dimensional vector where each dimension represents a single type of the required resources requested by cloud users. Once a VM resource requirement of a cloud user is accepted, a VM is generated with the user-defined resource requirement, we call this as VM configuration that can be represented as a capacity vector, $\vec{R}$, which are identified with d number of dimensions which are scalar components. It is the vector addition of normalised capacity vectors of each resource type to the total

capacity of its hosting PM. i.e. $\vec{R} = \overrightarrow{R_1} + \overrightarrow{R_2} + \cdots + \overrightarrow{R_d}$. For example, we only consider two resource types for a VM, i.e. CPU and memory resources. $\overrightarrow{R_{CPU}}$ represents the CPU of a VM and $\overrightarrow{R_{memory}}$ is the memory of the VM. The vector addition of both resources type is $\vec{R} = \overrightarrow{R_{CPU}} + \overrightarrow{R_{memory}}$.

Each PM, has a fixed total capacity vector $\vec{C}$ in d-dimensional space. Without loss of generality we assume that the values of $\vec{C}$ for all PMs have been normalised to 1. The resource utilisation vector $\vec{L}$ of PM is the vector additions of all capacities vectors of VMs that reside onPM, i.e. $\vec{L} = \sum_{i \in VMs} \overrightarrow{R^i}$ where VMs are a set of VMs are hosted in PM.

For simplicity, we consider three major resource types, i.e. CPU, memory and I/O as shown in Figure 5. (Note: we use the term, CPU, to represents virtual CPU throughout the context). All resources have been normalised and all the resource related information are expressed as vectors. The total capacity $\vec{C}$ of a PM is expressed as a vector from the origin of $(0, 0, 0)$ to point $(1, 1, 1)$. Resource utilisation vector $\vec{L}$ represents the current resource utilisation of a PM. The vector difference between the capacity vector $\vec{C}$ and the resource utilisation vector $\vec{L}$ represents the remaining capacity vector $\vec{F}$, which essentially captures how much capacity is left in the PM which could be used to allocate to an incoming VM.

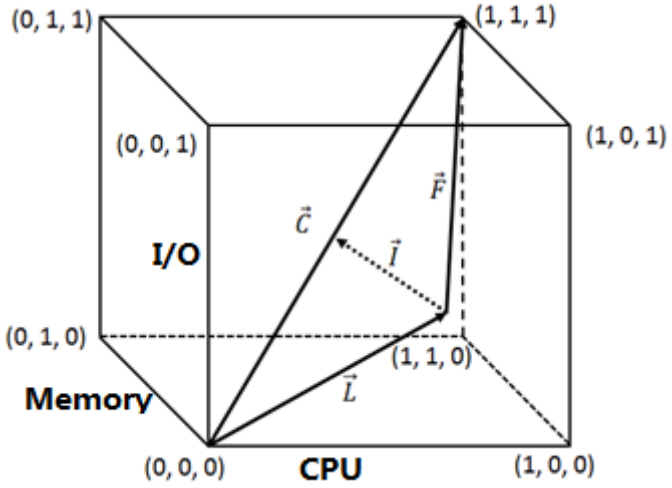

Figure 5. Vector representation of a PM

For example, a VM configuration in a PM or a PM configuration usually consists of CPU number and memory size. Suppose that the capacity vector of a PM is $(6, 6)$ representing 6 CPUs and 6 GB memory, it hosts two VMs, one with the capacity vector $(2, 2)$, another with capacity vector $(3, 3)$, the utilisation vector $\vec{L}$ of the PM is the addition of two VM capacity vectors, i.e. $(2, 2) + (3, 3) = (5, 5)$. Its remaining capacity vector is therefore $(6, 6) - (5, 5) = (1, 1)$.

Since vectors in PMs and VMs are multi-dimensions, they must exhibit some multi-dimensional characteristics. There have been similar related works considering multi-dimensional characteristics. For example, Ganapathi et al. [16] use the Euclidean distance between the applications in the space could describe the similarity among these applications. The advantage of this kind of approaches is that the prediction result could be highly accurate. But the accuracy usually relies on the abundant historical data and the computational complexity would be relatively higher. In the following two sections, we describes two types of heuristics that exhibit multi-dimensional characteristics, imbalance and volume, which are defined as heuristic metrics used for cloud reconfiguration algorithms.

3.3 Imbalance Heuristics

Imbalance heuristics is a measure that indicates the degree of imbalance of resource utilisation of a PM. We define the imbalance vector $\vec{I}$ of a PM as the vector difference between $\vec{L}$'s projection on $\vec{C}$and $\vec{L}$. $\vec{I}$ preserves directionality as well as the degree of imbalance. If $\vec{L}$ of a PM exactly aligns with $\vec{C}$, then we say that the PM is well utilised in a balanced manner along each resource axis. Note that the imbalance of a VM_i with respect to its hosting PM is defined in a similar manner, i.e. as the vector difference between $\overrightarrow{R^{VM_i}}$'s projection on $\vec{C}$ and $\overrightarrow{R^{VM_i}}$.

Assume that a resource utilisation vector $\vec{L}$ is represented by $c\vec{\imath} + m\vec{\jmath} + o\vec{k}$, where c, m and o are the normalised values of CPU, memory and I/O respectively. $\vec{C}$ is represented by $\vec{\imath} + \vec{\jmath} + \vec{k}$, where $\vec{\imath}$, $\vec{\jmath}$ and $\vec{k}$, are the unit vectors along CPU, memory and I/O respectively. Therefore, the projection vector [14] of the unit vector, $\vec{P}$, along the $\vec{C}$ is given in Equation 1.

$$\vec{P} = \left(\frac{1}{\sqrt{3}} \times c + \frac{1}{\sqrt{3}} \times m + \frac{1}{\sqrt{3}} \times o\right)\left(\frac{1}{\sqrt{3}} \times \vec{\imath} + \frac{1}{\sqrt{3}} \times \vec{\jmath} + \frac{1}{\sqrt{3}} \times \vec{k}\right) \qquad Equation\ 1$$

The term on the right hand side of Equation 1 is the unit vector along $\vec{C}$ and the term on the left hand side is the magnitude of projection of $\vec{L}$on $\vec{C}$ which can be simplified as shown in Equation 2.

$$\vec{P} = \left(\frac{c+m+o}{3}\right)\vec{\imath} + \left(\frac{c+m+o}{3}\right)\vec{\jmath} + \left(\frac{c+m+o}{3}\right)\vec{k} \qquad Equation\ 2$$

The $\vec{I}$ is given by the vector difference between $\vec{L}$ and the projection vector, e.g. $\vec{I} = \vec{L} - \vec{P}$, as shown in Equation 3.

$$\vec{I} = \left(c - \frac{c+m+o}{3}\right)\vec{\imath} + \left(m - \frac{c+m+o}{3}\right)\vec{\jmath} + \left(o - \frac{c+m+o}{3}\right)\vec{k} \qquad Equation\ 3$$

The magnitude $|\vec{I}|$ of $\vec{I}$ which measure the degree of imbalance of a PM is calculated as shown in Equation 4.

$$|\vec{I}| = \sqrt{\left(c - \frac{c+m+o}{3}\right)^2 + \left(m - \frac{c+m+o}{3}\right)^2 + \left(o - \frac{c+m+o}{3}\right)^2} \qquad Equation\ 4$$

We can generalise Equation 4 into multi-dimensions d as shown in Equation 5.

$$|\vec{I}| = \sqrt{\sum_{i \in dim}\left(\overrightarrow{L_\iota} - \frac{\sum_{\delta \in dim}\overrightarrow{L_\delta}}{d}\right)^2} \qquad Equation\ 5$$

where $\overrightarrow{L_\iota}$ is the resource utilisation vector of the dimension i of a PM and dim represents a set of d different number of resource types.

The average magnitude of imbalance of among m PMs in the cloud infrastructure is $\frac{\sum_{i=1}^{m}|\vec{I_\iota}|}{m}$. For example, given a VM called VM A with a capacity vector $(2, 4)$ (i.e. 2 CPUs and 4 GB memory) and its hosting PM has a capacity vector $(6, 6)$ (i.e. 6 CPUs and 6 GB memory). Suppose there is only VM A hosted in the PM, we can easily calculate the imbalance of the PM by using Equation 3. Therefore, the imbalance of the PM is $(-1/6, 1/6)$ and the magnitude of the imbalance of the PM is 0.236. The magnitude of the imbalance indicates the degree of imbalance of the PM is small.

3.4 Volume Heuristics

In this section, we introduce a scalar metric called volume heuristics. The volume heuristic is used to measure the ratio of a VM with respects to its hosing PM. In the other word, the volume of a VM with respects to its hosting PM is used to measure the average of all resources occupied by a d-dimensional VM with respects to its hosting PM. Suppose that we have a VM with capacity vector $\vec{R}$ hosting in a PM with capacity vector $\vec{T}$, we define the volume of a VM with respect to its hosting PM, V, as a sum of every single resource $|\overrightarrow{R_\iota}|$ with respects to its corresponding resource $|\overrightarrow{T_\iota}|$, where $i \in d$, divided by the number of dimensions of the VM, d, as shown in Equation 6.

$$V = \frac{\sum_{i \in dim}(\frac{|\overrightarrow{R_i}|}{|\overrightarrow{T_i}|})}{d} \qquad Equation\ 6$$

where dim represents a set of d different number of resource types.

Suppose that a VM has a VM configuration of 2 CPU and 2 GB memory which can be represented as a capacity vector $(2, 2)$ and its hosting PM has 6 CPU and 6 Gb memory which can be represented as a capacity vector $(6, 6)$, the volume of the VM with respect to its hosting PM is $\frac{\frac{2}{6}+\frac{2}{6}}{2} = \frac{1}{3}$. The volume of the VM represents the VM occupies one third of the total resource in its hosting PM.

The volume of a VM can also be calculated with respects to the total resource in its Cloud infrastructure. For example, a VM with a capacity vector $(2, 2)$, the capacity vector of its hosting environment is $(10, 10)$, the volume of the VM with respects to its hosting cloud environment is $\frac{\frac{2}{10}+\frac{2}{10}}{2} = \frac{1}{5}$.

3.5 Multivariate Normal Distribution of VM Configurations

If a cloud infrastructure provider provides user-defined configurations to cloud users, as number of different VM configurations increases, i.e. more and more cloud users submit their VM configurations to the cloud infrastructure providers; it is reasonable to assume that the VM configurations are in a normal distribution. Due to the nature of multiple resources in a VM configuration, it is necessary to consider that VM configurations form a multivariate normal distribution [17]. By collecting a large number of d-dimensional VM capacity vectors from cloud users, a d-dimensional VM configuration multivariate normal distribution is formed, i.e. $X \sim N_d(\mu, \Sigma)$, where $\vec{\mu}$ is a d-dimensional mean VM capacity vector as defined in Equation 8 and Σ is a $d \times d$ covariance matrix as defined in Equation 9.

In Figure 6, the Probability Density Function (PDF) of a VM configuration multivariate normal distribution is presented which was built based on 1000 VM configurations submitted by cloud users in IC Cloud [9]. The PDF of the VM capacity vector $\vec{R}$ can be calculated as shown in Equation 7. Let $\vec{R}$ be x, the PDF of $\vec{R}$ is given by

$$p(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \qquad \text{Equation 7}$$

Greater the PDF of x indicates the more frequently the VM configuration $\vec{R}$ likely to be submitted by a cloud user.

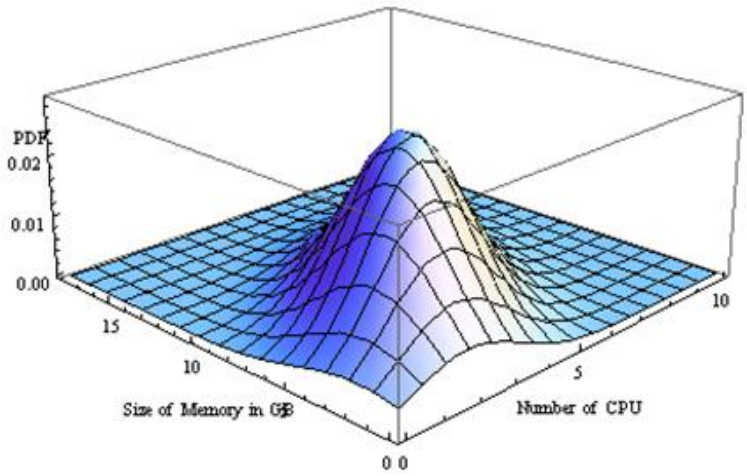

Figure 6. PDF of VM configuration of user-defined configurations

Mean Capacity Vector

A d dimensional mean capacity vector $\vec{\mu}$ is a vector that constitutes of d number of expected capacity of single resource types as shown in Equation 8.

$$\vec{\mu} = (E[X_1], E[X_2], \dots, E[X_i], \dots, E[X_d]) \qquad \text{Equation 8}$$

where X_i is a single resource types in the d dimensional space.

$d \times d$ Covariance matrix

A $\boldsymbol{d} \times \boldsymbol{d}$ covariance matrix is defined in Equation 9.

$$\Sigma = \left[Cov\left[X_i, X_j\right]\right]; i = 1,2, \dots, d; j = 1,2, \dots, d \qquad \text{Equation 9}$$

3.6 Grouping of logical machines

The resource vectors are very useful for making re-allocation decisions. One of our goals is to make the resource utilisation of PMs as balanced as possible after VM re-allocation, i.e. the $\vec{L}$ of a PM should be as closely aligned to the $\vec{C}$ as possible. Sometimes, a PM may have more resource utilisation along the memory axis compared to the CPU axis that causes the imbalance of PM. For example, ideally we aim to find a VM that is less utilised in memory compared to CPU so as to achieve resource balance in PMs after VM re-allocation. It is called finding complementary logical machines.

In order to keep PMs in the cloud infrastructure in a balanced manner after VM re-allocation, logical machines, such as PMs or VMs, can be classified into groups according to the resource utilisation of each resource type [14]. We divide logical machines into $d!$ (the factorial of d) number of groups according to number of resource types. For example, we consider 3 types of resource types (CPU, memory and I/O) which can be projected on to a plane as shown in Figure 7. We now have 3! number of groups, i.e. 6 groups.

Group COM

For group COM, CPU is the most utilised resource, memory is least utilised resource and I/O is in between the other two. Group COM and group MOC are complementary to each other in terms of resource utilisation.

Group CMO

For group CMO, CPU is the most utilised resource, I/O is least utilised resource and memory is in between the other two. Group CMO and group OMC are complementary to each other in terms of resource utilisation.

Group MCO

For group MCO, memory is the most utilised resource, I/O is least utilised resource and CPU is in between the other two. Group MCO and group OCM are complementary to each other in terms of resource utilisation.

Group MOC

For group MOC, memory is most utilised resource, CPU is least utilised resource and I/O is in between the two. Group MOC and group COM are complementary to each other in terms of resource utilisation.

Group OMC

For group OMC, I/O is the most utilised resource, CPU is least utilised resource and memory is in between the other two. Group OMC and group CMO are complementary to each other in terms of resource utilisation.

Group OCM

For group OCM, I/O is the most utilised resource, memory is least utilised resource and CPU is in between the other two. Group OCM and group MCO are complementary to each other in terms of resource

utilisation.

Grouping is an efficient way of identifying complementary logical machines. For instance, if we have a PM whose $Memory > CPU > I/O$, the PM will fall into group MCO. Its complementary VM will be a VM in exactly opposite group, i.e. group OCM. Therefore, re-allocating the complementary VM to the PM in the opposite group can achieve more balanced PM after VM re-allocation. Note that if no PM is found in one of the complementary groups, this will result in lower resource utilisation level, as complementary VMs may not be found.

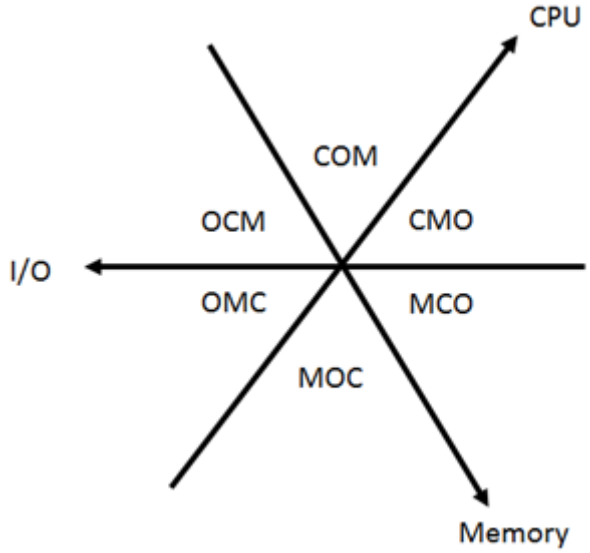

Figure 7. Grouping by resource utilisation

3.7 Resource Utilisation Level

The objective of the cloud reconfiguration algorithm is to maximise the utilisable resources among PMs. It is essential to measure the utilisable resource of a VM-to-PM mapping before and after VM re-allocation. In the other word, resource utilisation level is used to evaluate the level of utilisable resource of a VM-to-PM mapping. Since there is more than one resource type in a PM, utilisable resource of only a single type of resource is not sufficient to indicate the overall utilisable resource of the whole PM. Thus, we define a metric called resource utilisation level to measure the utilisable resource of a PM in terms of the mean VM capacity vector $\vec{\mu}$. The reason we use the mean VM capacity vector $\vec{\mu}$ is because it represents the most frequent configurations defined by the cloud users which can be obtained from the VM Multivariate Normal Distribution as shown in Equation 8.

Given a scalar component of a single resource type i of a d dimensional PM capacity vector, $|\overrightarrow{A_i}|$, where $i \in d$, and a scalar component of the same single resource type i of a d dimensional mean VM capacity vector $|\overrightarrow{\mu_i}|$, we define the Mean Vector Ratio of a single resource type i for a PM (MVR_i) as the ratio of $|\overrightarrow{A_i}|$ to $\overrightarrow{\mu_i}$ as shown in Equation 10.

$$MVR_i = \frac{|\overrightarrow{A_i}|}{|\overrightarrow{\mu_i}|} \qquad \text{Equation 10}$$

For example, we only consider capacity vector with CPU number and memory size, i.e. it can be represented as a vector (CPU number, memory size). Given the mean VM capacity vector $\vec{\mu}$ to be $(2, 1)$ and the PM capacity vector $\vec{A}$ to be $(2, 3)$, the Mean Vector Ratio of the CPU number is, therefore, $\frac{2}{2} = 1$, and the MVR of the memory size is $\frac{3}{1} = 3$.

We then define two functions, $f(\overrightarrow{A^{PM}})$ and $g(\overrightarrow{A^{PM}})$.

$f(\vec{A})$: This function returns the maximum number of mean VM capacity rounding down to the nearest integer that can be hosted by the PM capacity vector $\vec{A}$. Given a PM capacity vector $\vec{A}$, the maximum

number of the mean VM capacity vector $\vec{\mu}$ can be hosted by $\vec{A}$ rounding down to the nearest integer is shown in Equation 11. For example, given a mean VM capacity vector $(1.5, 1)$, let the PM capacity vector $\vec{A}$ be $(2, 3)$, $f(\vec{A})$ is $\min(\left\lfloor \frac{2}{1.5} \right\rfloor, \left\lfloor \frac{3}{1} \right\rfloor)$ which is equal to 1. Therefore, the given PM capacity vector $\vec{A}$ can only host the maximum number of mean VM capacity is 1.

$$f(\vec{A}) = \min_{i \in dim}(\lfloor MVR(\vec{A}_i) \rfloor) \qquad \text{Equation 11}$$

where dim represents a set of d different number of resource types.

$g(\vec{A})$: This function is similar to the previous function $f(\vec{A})$. However, instead of returning a round-down integer, it returns the maximum number of mean VM capacity without rounding, i.e. in decimal. Given a PM capacity vector $\vec{A}$, the maximum number of the mean VM capacity $\vec{\mu}$ can be hosted by $\vec{A}$ without rounding is shown in Equation 12. For example, given a mean VM capacity vector $(1.5, 1)$, let the PM capacity vector $\vec{A}$ to be $(2, 3)$, $f(\vec{A})$ is $\min(\left\lfloor \frac{2}{1.5} \right\rfloor, \left\lfloor \frac{3}{1} \right\rfloor)$ which is equal to 1.33. Therefore, the given PM capacity vector $\vec{A}$ hosting the maximum number of mean VM capacity is 1.33.

$$g(\vec{A}) = \min_{i \in dim}(MVR(\vec{A}_i)) \qquad \text{Equation 12}$$

where dim represents a set of d different number of resource types.

Given a VM-to-PM mapping α, there is a set of PMs M in the mapping α which consists of m number of PMs in the mapping α. Each PM_i, $i \in M$, has its capacity $\vec{C}^{PM_i}$, utilised resources $\vec{L}^{PM_i}$ and remaining capacity $\vec{F}^{PM_i}$. The resource utilisation level for the mapping, denoted as $\psi(\alpha)$, is defined in Equation 13.

$$\psi(\alpha) = \frac{g(\sum_{i \in M} \vec{L}^{PM_i}) + \sum_{i \in M} f(\vec{F}^{PM_i})}{g(\sum_{i \in M} \vec{C}^{PM_i})} \qquad \text{Equation 13}$$

We all know that the total capacity of all PMs and the total utilised resources remain constant before and after VM re-allocation in the Cloud infrastructure. Therefore, $g\left(\sum_{i \in M} \vec{L}^{PM_i}\right)$ and $g(\sum_{i \in M} \vec{C}^{PM_i})$ remain the unchanged. However, the remaining capacity of each PM may vary after VM re-allocation, $f(\vec{F}^{PM_i})$ returns the maximum number of the mean VM capacity $\vec{\mu}$ hosted by the remaining capacity of PM_i, any unallocated resource will be considered as the non-utilised resource. This is because it is unlikely to meet the VM resource requirements from cloud users.

4. CLOUD RECONFIGURATION ALGORITHM

Cloud reconfiguration algorithms are based on VM re-allocation techniques for constructing a suitable reconfiguration plan in order to achieve greater resource utilisation in the Cloud infrastructure. This section details the assumptions of this algorithm and provides a detailed algorithm description of the VM re-allocation problem as the MDBP problem [10, 11]. Due to the NP-hard nature of the MDBP problem and the need to compute the solutions in a reasonable amount of time, approximation approaches have been developed to provide good results. Polynomial time approximate solutions (PTAS) [10] is proposed to solve the problem with a low approximation ratio. Kimbrel et al. [18] proposes an algorithm for a restricted job allocation problem with minimum migration constraints, but their approach does not allow for multiple jobs being assigned to a single machine. It is important to note that most existing approaches used still ignore the multi-dimensional characteristics of VMs and PMs [19].

In general, the existing cloud reconfiguration algorithms comprise of two stages: Target Mapping Generation (TMG) and Migration Plan (MP).

Target Mapping Generation

The cloud reconfiguration algorithms firstly compute a VM-to-PM target mapping which guarantees better resource utilisation than the existing VM-to-PM mapping. Note that a VM-to-PM mapping is the arrangement of VMs to its hosting PM.

Migration Plan

The cloud reconfiguration algorithms construct a migration plan from the existing mapping to the target mapping with the minimum migration costs. However, existing algorithms take all PMs in the Cloud infrastructure into consideration for VM re-allocation that results in high migration costs. As the number of PMs increases in the Cloud infrastructure, more VMs that have been allocated to the PMs need to be considered for VM re-allocation. This causes an increase in the total migration costs in MP.

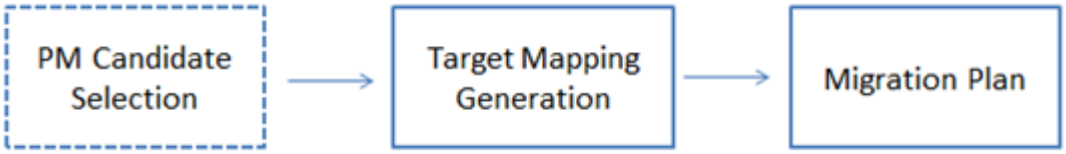

Figure 8. Cloud Reconfiguration Stages

Selecting suitable PMs in the Cloud infrastructure before re-allocation is essential to reduce the total migration costs. Therefore, we propose a cloud reconfiguration algorithm that firstly selects a list of suitable PMs before the traditional two stages. We call this stage as PM Candidate Selection (PMCS) as shown in Figure 8.

PM Candidate Selection

The cloud reconfiguration algorithm firstly selects a set of suitable PMs for VM re-allocation, this would potentially reduce number of VMs required to be re-allocated and hence reduce the total migration costs.

4.1 Assumptions

The algorithm requires a history of VM configurations from cloud users, a multivariate normal distribution is built based on the history. VM capacities can be seen as static at the point when the algorithm proceeds. We also assume that a batch of VMs has been allocated to the Cloud infrastructure and may need to be re-allocated. The algorithm schedules VMs according to resource utilisation level of VM-to-PM mappings.The algorithm is then executed at regular time intervals, say every hour, for calculating the resource utilisation level of the current mapping in the Cloud infrastructure and computing for a new mapping. If the resource utilisation level of the new mapping is greater than the current mapping, the algorithm will be triggered for VM re-allocation, otherwise, the algorithm will be aborted and wait for the next time interval.

4.2 PM Candidate Selection

At this stage, a set of suitable PMs is selected from all PMs in the cloud infrastructure for VM re-allocation so as to reduce migration costs. Since a new incoming VM will be allocated to one of the PMs, it is essential to choose a PM according to its remaining capacity. However, not every PM has enough remaining resources to host a new incoming VM. This stage is to choose PMs whose remaining capacity is not in a similar size to the mean VM capacity, as the mean VM capacity is the most frequent VM configuration requested by cloud users. The remaining capacity of a PM similar to the mean VM capacity indicates that it is more likely to host a new incoming VM whose configuration is similar to the mean VM capacity. This results in more utilisable resource in the PM and is not required for re-allocation. Therefore, we choose a PM whose remaining capacity does not frequently occur in the multivariate normal distribution. The remaining capacity $\vec{F}$ of a PM is considered as an input to Equation 7 to obtain the corresponding PDF $p(\vec{F})$.

As we mention earlier, we choose PMs whose remaining capacity is not in a similar size to the mean VM capacity. Therefore, for choosing a set of suitable PMs that is not in a similar size to the mean VM capacity, a range of PDF is required to be defined for selecting suitable set of PMs. It is common convention of using $p(\vec{\mu} \pm 1.96\sigma)$ where σ is the standard deviation of the distribution. The range covers 95% of area under a normal distribution, i.e. 95% confidence level. In our study, we set the range as $p(\vec{\mu} \pm 1.96\sigma)$. We choose PMs whose remaining capacities are not in this range, i.e. 5% of the area, where the VM configurations rarely occur in the distribution. Note that the confidence level can be adjusted by cloud infrastructure providers according to their demands for utilisable resources in the cloud infrastructure. For example, if we choose a small confidence level, more PMs will be selected, more possible VM-to-PM mappings, which lead to higher migration costs, but more utilisable resources.

4.3 Target Mapping Generation

We propose two types of heuristic approaches used for generating a target mapping from the original mapping; one uses the imbalance heuristic, and the other uses the volume heuristic. After performing PMCS, we have a list of selected PMs. For imbalance heuristic, we group the list of suitable PMs into $d!$ number of PM groups according to their resource utilisations. After grouping, there are $\frac{d!}{2}$ sets of PMs and each set is used as input for TMG. For volume heuristics, the whole list of selected PMs is taken as an input.

Imbalance Heuristics

Before using imbalance heuristics, PMs into different groups according to their resource utilisation of each resource type can be firstly classified before generating a new target mapping. By considering three resource types, i.e. number of CPU, memory size and I/O, we have 3 specific resource sets of PMs, each set is for each resource type respectively. For example, I/O set is a set that contains PMs with moderate amount I/O resources. This set can be split into two complementary groups, COM group ($CPU > I/O > Memory$) and MOC group ($Memory > I/O > CPU$) respectively.

VMs hosted in a specific resource set of PMs are classified into two complementary groups according to their VM configurations. For example, given two complementary groups in I/O set, one is COM group (CPU-intensive) and the other is MOC group (Memory-intensive), a CPU-intensive VM will be placed to a CPU-intensive group called Pos whilst a Memory-intensive VM will be placed to a Memory-intensive group Neg. If a VM is neither CPU nor Memory-intensive, it can be placed to either group.

Two complementary groups in a specific resource set are input to Algorithm 1 for constructing a new VM-to-PM mapping in turn. VMs in group Pos and group Neg are sorted by the magnitude of the imbalances of their VM capacities with respects to their hosing PMs in descending order. (line 3 and 4), i.e. a VM with greater magnitude of the imbalance will be allocated to a PM first. At the beginning of the algorithm, $PMList$ is a list that contains all selected PMs without any VM being allocated in the specific resource set (line 5), where PMs in $PMList$ are sorted by the magnitude of the imbalances of the PMs' current resource utilisation in descending order in each iteration (line 7). The algorithm allocates VMs from group Pos and Neg to the PMs in $PMList$. In this algorithm, every VM is allocated to a PM that is complementary to the VM (line 8 to 18) so as to result in better degree of imbalance of PMs after allocation. The algorithm outputs $PMList$ with new VM allocation, i.e. a partial VM-to-PM mapping for a specific resource set. Repeat the algorithm for other specific resource sets. The target mapping is an aggregation of all partial VM-to-PM mappings generated by the algorithm and the PMs that have not been selected in PMCS.

Algorithm 1 Imbalance Heuristics

Require Pos, Neg

1: $Pos = \{VM_1, \ldots, VM_i, \ldots VM_n\}$
2: $Neg = \{VM_{n+1}, \ldots, VM_j, \ldots, VM_m\}$
3: $Pos = SortByImbalance(Pos)$
4: $Neg = SortByImbalance(Neg)$
5: $PMList = \{PM_1, \ldots, PM_k, \ldots, PM_o\}$
6: **while** $\neg\, Pos.empty() \wedge \neg\, Neg.empty()$ **do**
7: **PMList=SortByImbalance(PMList)**
8: **for** PM_k **in** $PMList$ **do**
9: **if** PM_k complementary to VM_i in Pos **then**
10: $PM_k.put(VM_i)$
11: $Pos.remove(VM_i)$
12: **break**
13: **else**
14: $PM_k.put(VM_j)$
15: $Neg.remove(VM_j)$
16: **break**
17: **end if**
18: **end for**
19: **end while**
20: **return** $PMList$

Volume Heuristic

By taking the whole set of selected PMs into account, a new VM group called VMList is created where all VMs belonging to the set of selected PMs are placed to VMList. It is used as an input to Algorithm 2 for constructing a VM-to-PM mapping. The VMs in VMList are sorted by volume of VMs with respects to

the selected PMs in descending order (line 2). Similar to the imbalance approach, at the begining of the algorithm, PMList is a set that contains all selected PMs without any VM allocation (line 3) and they are sorted by the volume of remaining capacity of the selected PMs in descending order in each iteration (line 5). Our allocation approach is similar to best-fit decreasing algorithm, i.e. the algorithm starts allocating VMs with the largest volume to PMs with the minimum volume of remaining capacity in PMList first (line 4 to 9). The reason we use best-fit decreasing is because it is one of the simplest heuristic algorithms for solving the bin-packing problem [10, 11]. The algorithm finally outputs PMList. The target mapping is an aggregation of PMs in PMList and PMs that have not been selected in PMCS.

In order to evaluate whether a target mapping is reasonable for VM re-allocation, we apply Equation 13 to obtain the resource utilisation level of the orginal and target mapping. If the resource utilisation level of the target mapping is greater than the original VM-to-PM mapping, migration plan will then be executed.

Algorithm 2 Volume Heuristic
Require $VMList$

```
1:  VMList = {VM_1, ..., VM_i, ..., VM_n}
2:  VMList = SortByVolume(VMList)
3:  PMList = {PM_1, ..., PM_k}
4:  while ¬ VMList.empty() do
5:      PMList = SortByRemainingVolume(PMList)
6:      for PM_i in PMList do
7:          PM_i.put(VM_j)
8:          VMList.remove(VM_j)
9:          break
10:     end for
11: end while
12: return PMList
```

4.4 Migration Plan

The migration plan must consider the costs associated with performing the migration of VMs. These VMs are logical servers and may be serving real time requests. Therefore, any delay resulting from the migration needs to be considered as a cost. Use of a cost function also helps in designing an algorithm which does not lead to frequent migration of VMs. Solutions aiming to minimise the number of moves from current mapping to target mapping is still an open research problem. For example, VM migrations between PMs can be performed directly, or via intermediate PMs (pivot), depending on resource utilisation of the target PM. Moving through intermediate PMs generates greater migration costs due to more migration operations. Therefore, the constraint programming [12] is used to guarantees the minimum moves of migrations. The detail of generating a migration plan with the lowest migration cost can be found in [4, 20, 21].

5 EXPERIMENTAL EVALUATION

The experimental evaluation is designed to illustrate the effectiveness of our approach for VM re-allocation based on user-defined and provider-defined VM configurations and demonstrate the importance of multi-dimensional characteristics. In order to evaluate the effectiveness of our approach, we developed a cloud simulation toolkit called ICCS (Imperial College Cloud Simulation) for simulating cloud resource management which is similar to CloudSim [22]. Three resource types (i.e., CPU, memory and I/O) are considered for VM re-allocation. In this experimental setting, we assume that all PMs in the Cloud infrastructure are homogeneous. Each PM has 36 CPUs, 36 GB memories and 36 GB I/O.

We conducted two sets of experiments, one is based on the user-defined VM configurations and the other is based on the provider-defined VM configurations. In the experiments, we mainly evaluate the resource utilisation level of target mappings and compare with other approaches.

5.1 Experiments with User-defined VM configurations

This experiment is based on the user-defined VM configurations. Before we conduct the experiment, we collected 1000 VM configurations from IC Cloud [9] for building a multivariate normal distribution. The 3-dimensional mean VM capacity vector and the covariance matrix were then calculated from the distribution.

We firstly generate 150 original VM-to-PM mappings. In the first iteration, we start with 1 hosting PM to be available for VM allocation. We repeat the following procedure for 150 times with increasing the total number of hosting PMs by 1 in each iteration. We then allocate a VM one by one to a hosting PM using

First Fit algorithm that is one of the most common approaches for VM placement to PMs [9] in cloud computing industry. Each allocated VM was randomly generated from the multivariate normal distribution. The allocation of VM to the PM stops when the hosting PM does not have sufficient resources to host the next incoming VM. If there are more resources available for other hosting PMs for VM allocation, the allocation of VM to the next available hosting PM will be performed. If there are no more resources available in PMs, the VM allocation will be stopped and an original mapping is generated. In this way, 150 original VM-to-PM mappings can be generated. For each original VM-to-PM mapping, the resource utilisation level of the original mapping can be calculated using Equation 13.

The cloud reconfiguration algorithm is then applied to construct a new target VM-to-PM mapping based on its corresponding original VM-to-PM mapping. The resource utilisation level of the target mapping can then be calculated. Therefore, the resource utilisation level for the original VM-to-PM mapping and the new target VM-to-PM mappings are known, the following sections compare and discuss the experiments conducted for the cloud reconfiguration algorithm.

PM Candidate Selection for User-defined Configurations

In this experiment, we evaluate the PMCS of the cloud reconfiguration algorithm for the user-defined configurations. We conducted two types of experiments, one is with PMCS and the other is without PMCS for both imbalance and volume heuristics.

Without performing PMCS, the average resource utilisation level for both imbalance and volume heuristics are slightly better than the ones with PMCS by 0.42% and 1.02% respectively as shown in Figure 9 and Figure 10. This is due to local optimal is achieved within each group in PMCS and the ones without PMCS take all PMs into consideration which makes VM re-allocation becomes more flexible. However, the rate of decrease in resource utilisation level is very small.

In addition to that, the number of PMs considered for VM re-allocation decreases by 9.17% on average by performing PMCS as number of PMs increases as shown in Figure 11. We conclude that reducing number of PMs used for VM re-allocation using PMCS does not have a significant impact on the resource utilisation level of the target VM-to-PM mappings; instead, it results in less migration costs in the migration plan as number of PMs considered for VM re-allocation decreases whilst the resource utilisation level stays relatively constant. This leads to smaller number of migrations need to be performed.

We notice that using PMCS results in an insignificant decrease of resource utilisation level of the mappings, at the same time, it reduces a certain number of migrations. Therefore, we conclude that PMCS is helpful for cloud reconfiguration algorithm with user-defined configurations.

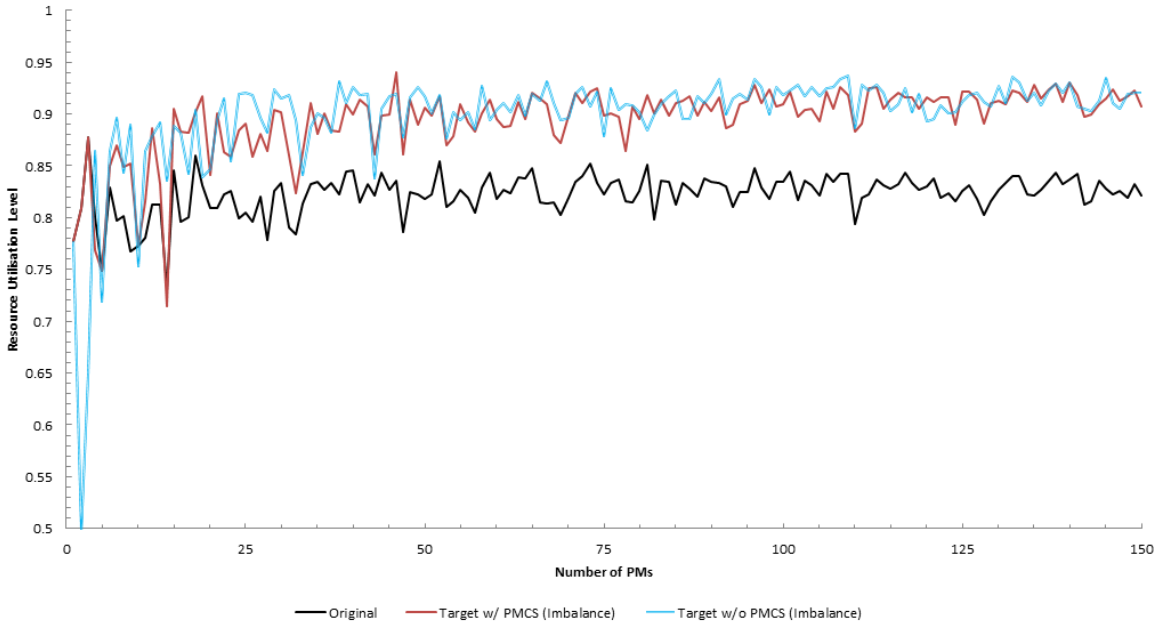

Figure 9. Resource utilisation with and without PMCS using imbalance heuristics with user-defined configuration

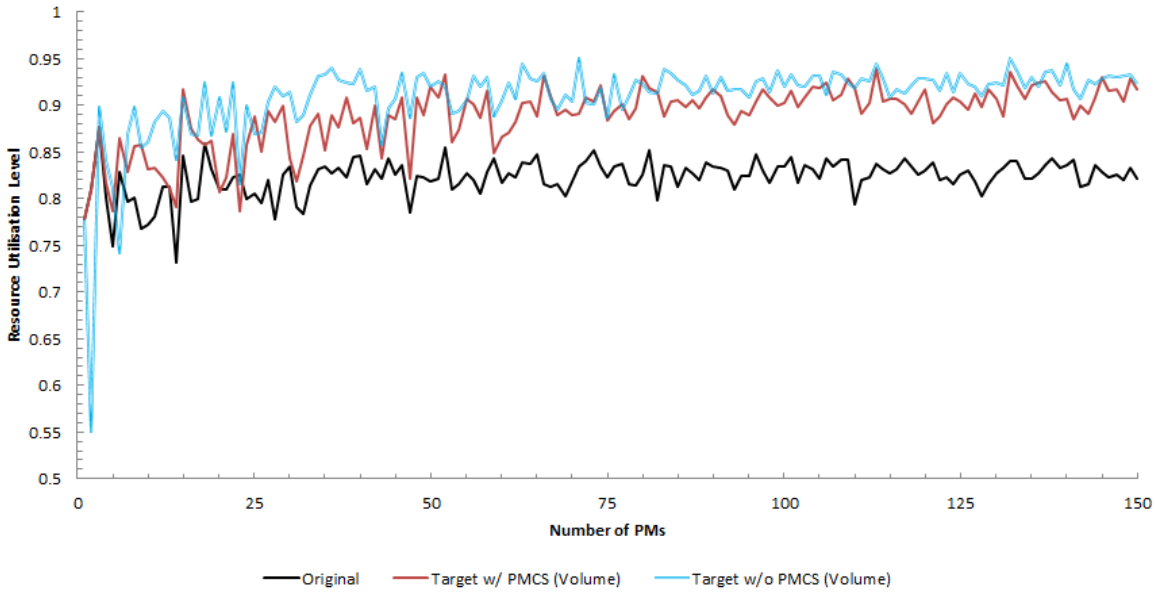

Figure 10. Resource utilisation with and without PMCS using volume heuristics with user-defined configuration

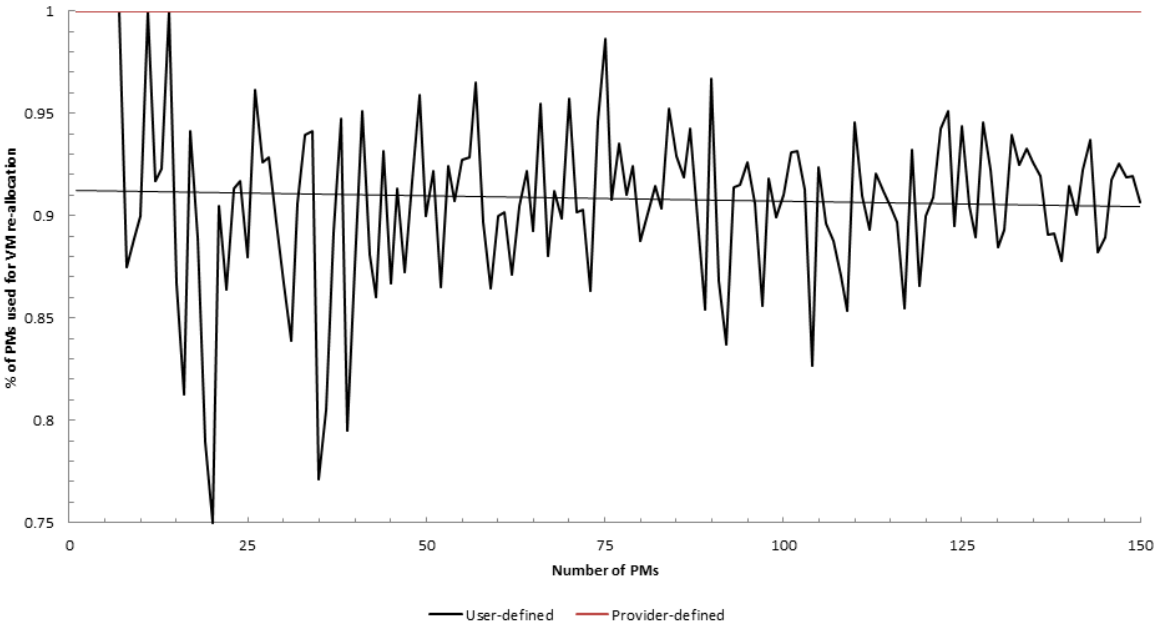

Figure 11. Number of PMs used for re-allocation

Grouping for Volume Heuristics

In this experiment, we specially added the feature "grouping" to the imbalance heuristics of the cloud reconfiguration algorithm. In this experiment, we evaluate whether grouping of the cloud reconfiguration algorithm for volume heuristics is suitable.

In Figure 12, it shows that volume heuristic without grouping performs slightly better than the one with grouping, but it is not significant, this is because volume heuristic does not take advantage of grouping as grouping only works well for finding complementary VMs to PMs. Therefore, we conclude that grouping

is not an essential feature for volume heuristics.

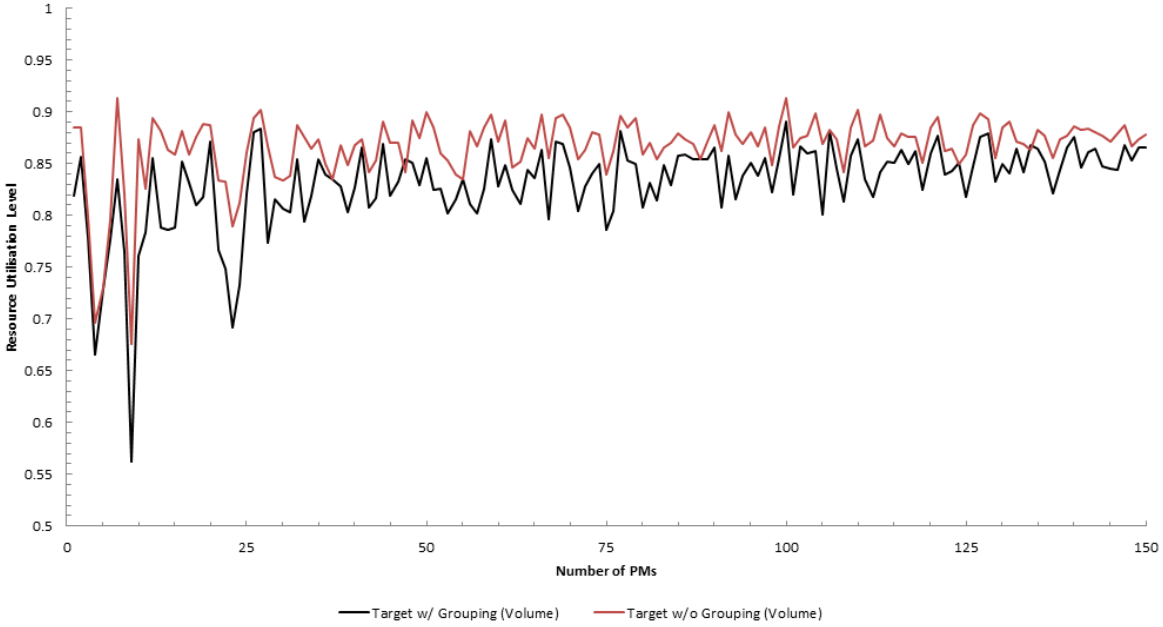

Figure 12. Resource utilisation with and without grouping for volume heuristics with user-defined configurations

Difference between Imbalance and Volume Heuristics for User-defined Configurations

In this experiment, we evaluate the difference in the resource utilisation level between imbalance and volume heuristics for user-defined configurations. Figure 13 shows the fluctuation of the resource utilisation levels using imbalance and volume heuristics with PMCS respectively and the difference between them is only \$0.57\%\$. It is easy to notice that the resource utilisation levels increase as the number of PMs increases. Therefore, there is no significant difference in the resource utilisation level.

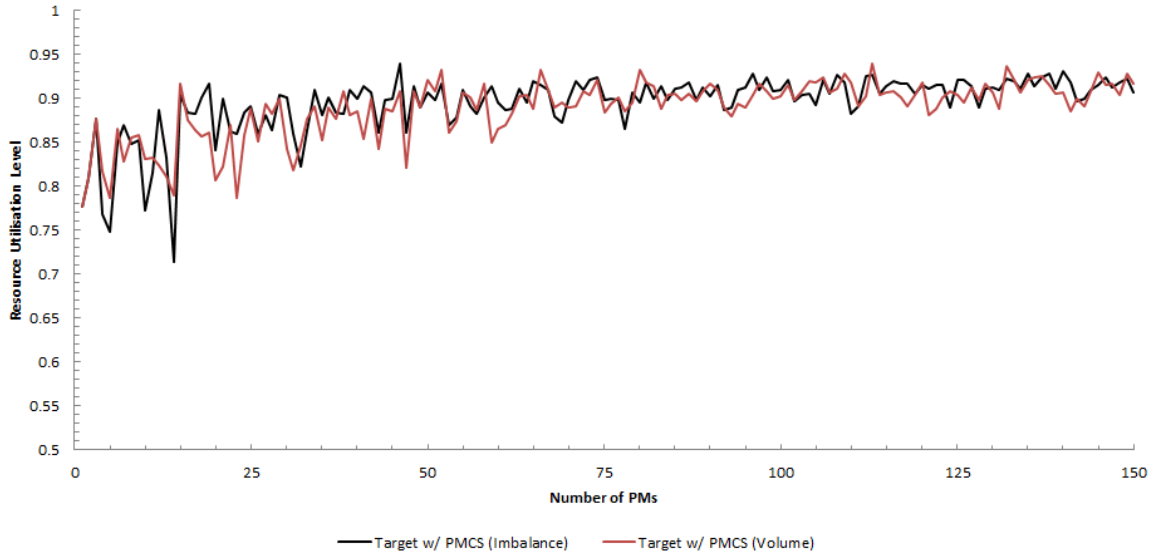

Figure 13. Resource utilisation with user-defined configuration using imbalance and volume

Combination of Imbalance and Volume Heuristics for User-defined Configurations

To achieve better resource utilisation level, we combine the two heuristics (imbalance and volume) by taking the one with higher resource utilisation level of the same mapping. This results in an increase of resource utilisation level from the original mapping by 7.71%. In Figure 14, it shows a combination of two multi-dimensional heuristics (imbalance and volume) outperforms the combination of each single resource type (i.e. heuristic approaches that consider CPU, memory and I/O individually) by 17.89% on average. Note

that the combination of each single resource type is obtained by taking the one with higher resource utilisation level of the same mapping. Therefore, multi-dimensional heuristics play an important part in VM re-allocation.

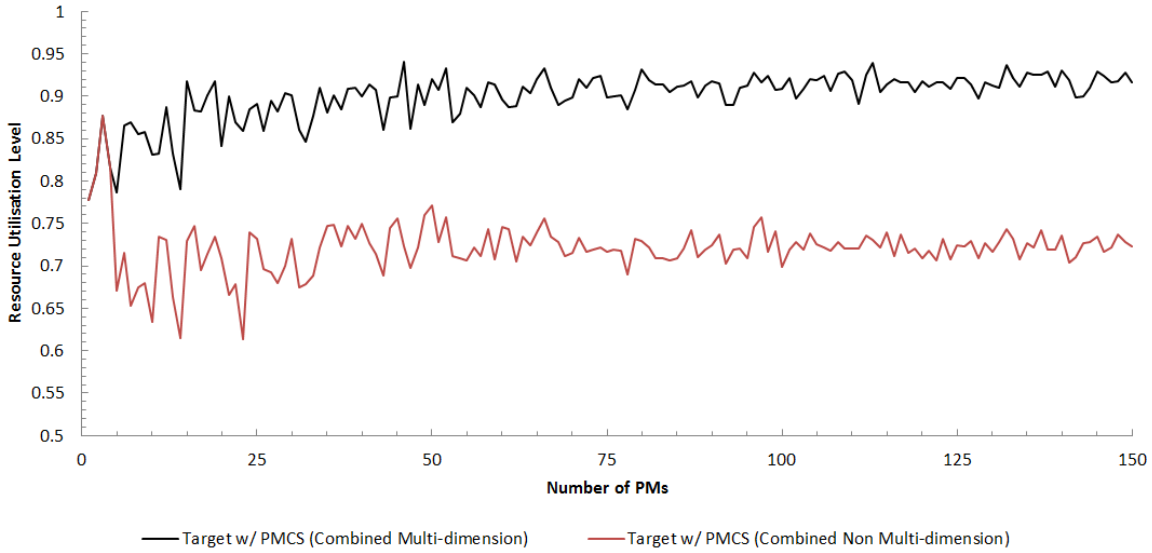

Figure 14. Resource utilisation with user-defined configuration using combined multi-dimensions and non multi-dimensions

5.2 Experiments with Provider-defined VM configurations

This experiment is based on the provider-defined VM configurations. As the number of types of VM configurations given by a cloud provider is fixed, Amazon EC2 offers 10 types of VM configurations. We also generated a total of 150 original VM-to-PM mappings also generated for this experiment. The similar experimental methodology is also applied to this experiment, but the set of VM for initial allocation was generated from the 10 types of the Amazon EC2 VM configurations.

We firstly generate 150 original VM-to-PM mappings. In the first iteration, we start with 1 hosting PM to be available for VM allocation. We repeat the following procedure for 150 times with increasing the total number of hosting PMs by 1 in each iteration. We then allocate a VM one by one to a hosting PM using First Fit algorithm that is one of the most common approaches for VM placement to PMs in cloud computing industry. Each allocated VM was randomly chosen from the 10 types of the Amazon EC2 VM configurations. The allocation of VM to the PM stops when the hosting PM does not have sufficient resources to host the next incoming VM. If there are more resources available for other hosting PMs for VM allocation, the allocation of VM to the next available hosting PM will be performed. If there are no more resources available in PMs, the VM allocation will be stopped and an original mapping is generated. In this way, 150 original VM-to-PM mappings can be generated. For each original VM-to-PM mapping, the resource utilisation level of the original mapping can be calculated using Equation 13.

The cloud reconfiguration algorithm is then applied to construct a new target VM-to-PM mapping based on its corresponding original VM-to-PM mapping. The resource utilisation level of the target mapping can then be calculated. Therefore, the resource utilisation level for the original VM-to-PM mapping and the new target VM-to-PM mappings are known, the following sections compare and discuss the experiments conducted for the cloud reconfiguration algorithm.

Difference between Imbalance and Volume Heuristics for Provider-defined Configurations

Figure 15 shows the fluctuation of the resource utilisation levels using imbalance and volume heuristics with PMCS respectively, and the difference between them is 10.54%. This is due to the fact that the number of provider-defined VM configurations is limited and could not form a normal distribution. Therefore, imbalance heuristics performed less efficient than the volume heuristics.

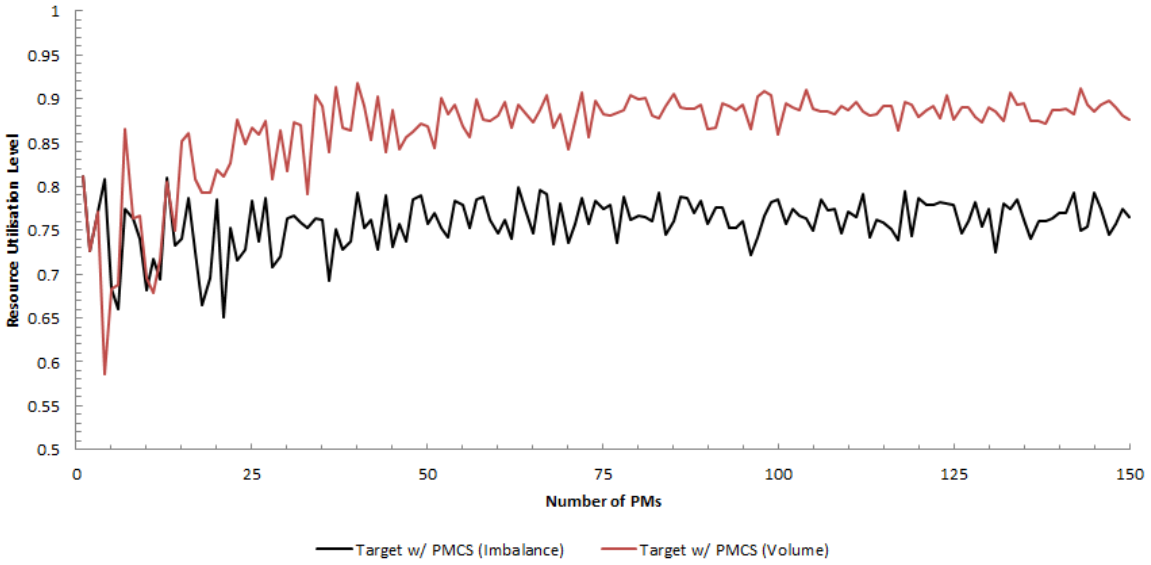

Figure 15. Resource utilisation with imbalance and volume heuristics with provider-defined configuration

PMCS for Provider-defined Configurations

In this experiment, we evaluate the PMCS of the cloud reconfiguration algorithm for the provider-defined configurations. We conducted two types of experiments, one is with PMCS and the other is without PMCS for both imbalance and volume heuristics.

As shown in Figure 11, the cloud reconfiguration algorithm with PMCS selects all PMs as suitable PMs for VM re-allocation from the provider-defined configurations. This is due to the fact that the number of configurations is small and could not exhibit normal distribution. It is difficult to find a remaining capacity of a PM that is similar to the mean VM capacity vector. Therefore, PMCS does not help infrastructure providers to reduce their migration costs if they offer provider-defined configurations.

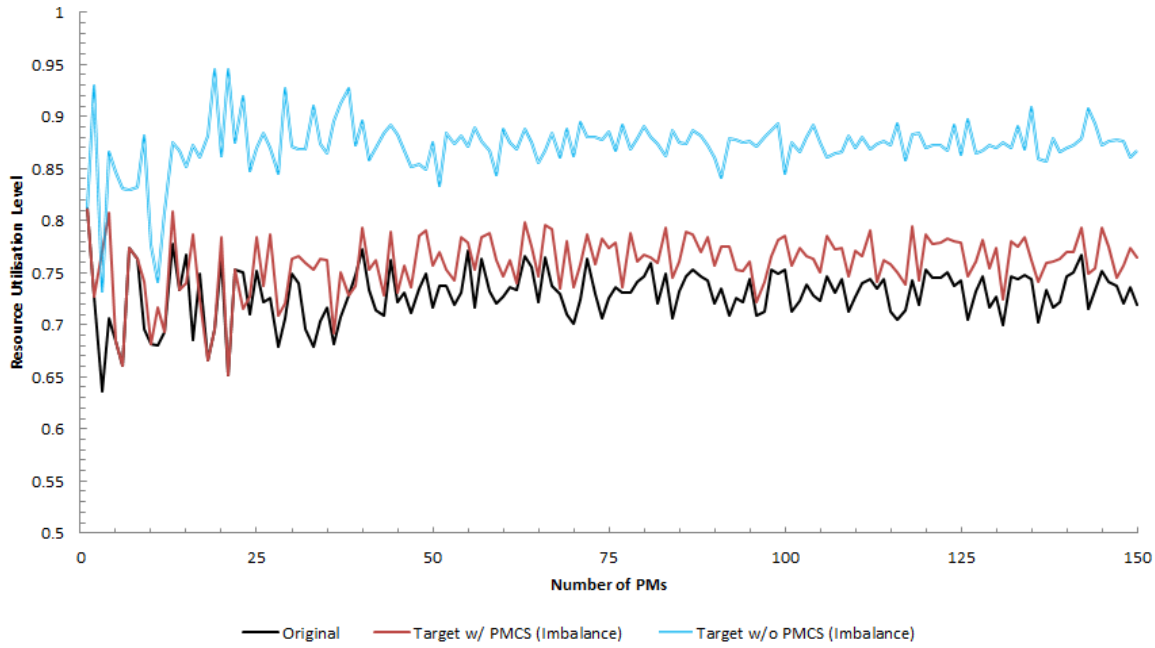

Figure 16. Resource utilisation with and without PMCS with provider-defined configuration using imbalance heuristics

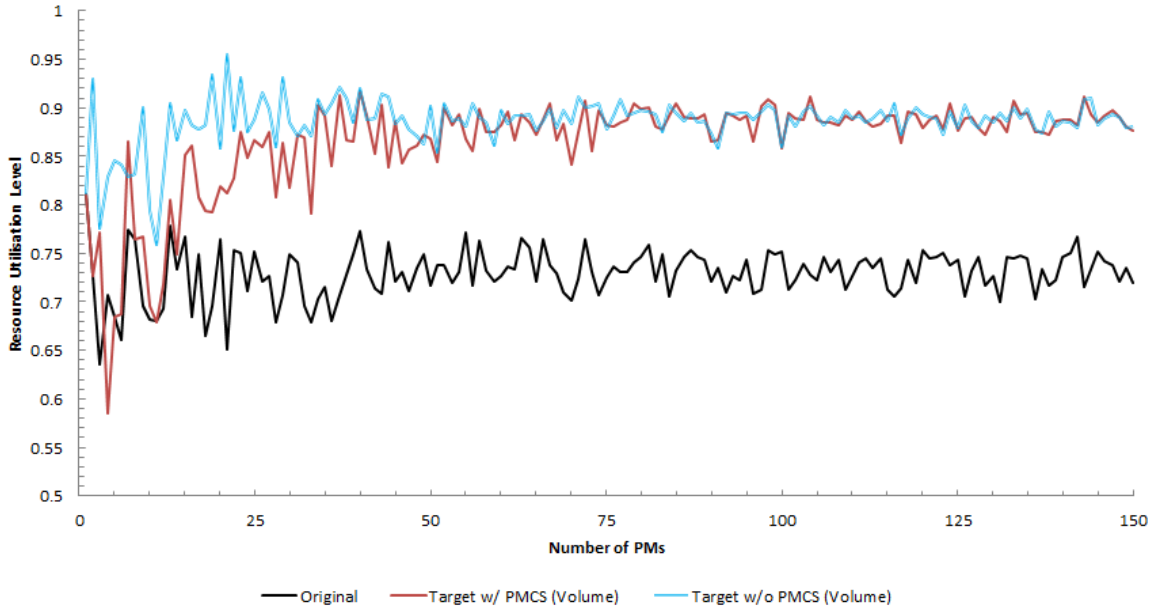

Figure 17. Resource utilisation with and without PMCS with provider-defined configuration using volume heuristics

In addition, without performing PMCS, the average resource utilisations for imbalance heuristics is better than the one with PMCS by 11.30% as shown in Figure 16 and the average resource utilisations for volume heuristics is better than the one with PMCS by 2.17% as shown in Figure 17. In overall, the imbalance and volume heuristics with PMCS do not perform well as it is difficult to find a VM that is complementary to a PM given that the number of types of VM configurations is limited and normal distribution for the provider-defined configurations can not be built. Therefore, PMCS is not suitable for provider-defined configurations.

Combination of Imbalance and Volume Heuristics for Provider-defined Configurations

In this experiment, we evaluate the difference in resource utilisation level between the combination of multi-dimensional heuristics (imbalance and volume heuristics) and the combination of each single resource type for provider-defined configurations. In Figure 18, by combining two multi-dimensional heuristics, i.e. imbalance and volume heuristics, the resource utilisation level increases by 13.71% in comparison with the original mapping. Note that the combination of the two heuristics is a method that takes one of the heuristics with higher resource utilisation level of the same mapping.

Further, it performs better than the combination of each single resource type by only 0.49% on average. Note that the combination of each single resource type is obtained by taking the one with higher resource utilisation level of the same mapping. This shows that there is no significant difference on improving resource utilisation level using these two approaches as the number of types of VM configurations is limited, and the normal distribution cannot be formed. Therefore, we conclude that multi-dimensional heuristics perform a slightly better than the non-dimensional characteristics, but it is not significant.

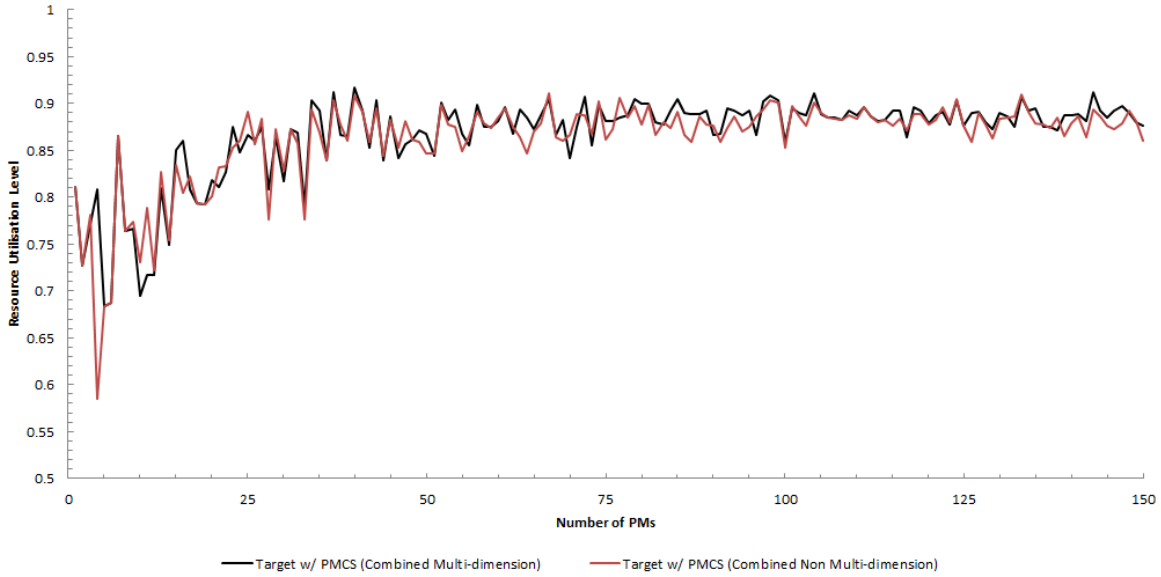

Figure 18. Resource utilisation with provider-defined configuration

5.3 Experiment with the Combination of User-defined and Provider-defined Configurations

As we mentioned earlier, Cloud infrastructure providers normally provide either user-defined or provider-defined configurations to cloud users. In this experiment, we assume that a Cloud infrastructure provider can offer both user-defined and provider-defined configurations to cloud users.

We firstly deployed the same cloud simulator called ICCS to randomly generate two set of user-defined and provider-defined VMs (a total of 46193 VMs) which have been plotted as a multivariate normal distribution as shown in Figure 19. The user-defined configurations were randomly generated from an IC Cloud VM distribution which was built by 1000 VM configurations. The 3-dimensional mean VM capacity vector and the covariance matrix were then calculated from the distribution. The provider-defined VMs were randomly generated from 10 types of VM configurations of Amazon EC2.

In comparison with a user-defined VM configuration distribution (Figure 3.6), we can easily see that the covariance of the combined VM configuration distribution is much greater than the user-defined VM configuration, this is due to the fact that there are significant numbers of provider-defined configurations which are not similar to the user-defined configurations. Thus, they affect the VM distribution directly.

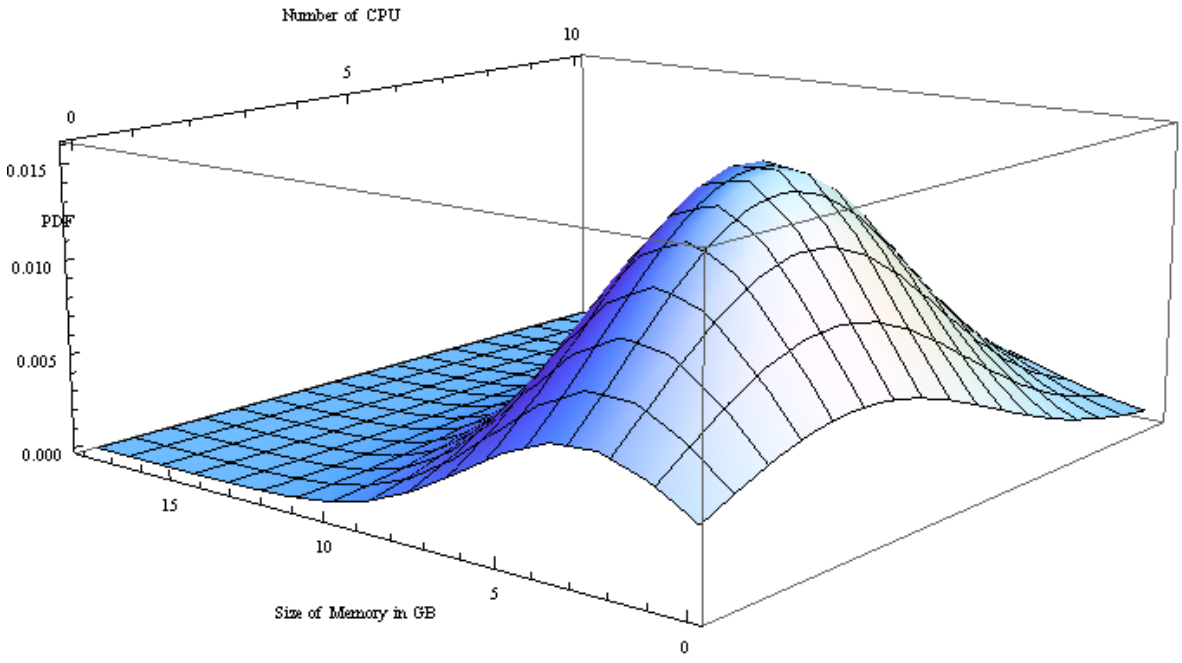

Figure 19. PDF of VM configuration distribution of user-defined and provider-defined configurations

For this experiment, we also generated a total of 150 original VM-to-PM mappings for re-allocation using imbalance heuristics. The similar experimental methodology was also applied to this experiment, but the set of VM for initial allocation is generated from the IC Cloud user-defined configuration distribution and the 10 types of the Amazon EC2 VM configurations, i.e. we firstly generate 150 original VM-to-PM mappings. In the first iteration, we start with 1 hosting PM to be available for VM allocation. We repeat the following procedure for 150 times with increasing the total number of hosting PMs by 1 in each iteration. We then allocate a VM one by one to a hosting PM using First Fit algorithm that is one of the most common approaches for VM placement to PMs in cloud computing industry. Each allocated VM was randomly chosen from the 10 types of the Amazon EC2 VM configurations and IC Cloud user-defined configuration distribution. The allocation of VM to the PM stops when the hosting PM does not have sufficient resources to host the next incoming VM. If there are more resources available for other hosting PMs for VM allocation, the allocation of VM to the next available hosting PM will be performed. If there are no more resources available in PMs, the VM allocation will be stopped and an original mapping is generated. In this way, 150 original VM-to-PM mappings can be generated. For each original VM-to-PM mapping, the resource utilisation level of the original mapping can be calculated using Equation 13.

The cloud reconfiguration algorithm is then applied to construct a new target VM-to-PM mapping based on its corresponding original VM-to-PM mapping. The resource utilisation level of the target mapping can then be calculated. Therefore, the resource utilisation level for the original VM-to-PM mapping and the new target VM-to-PM mappings are known, the following sections compare and discuss the experiments conducted for the cloud reconfiguration algorithm.

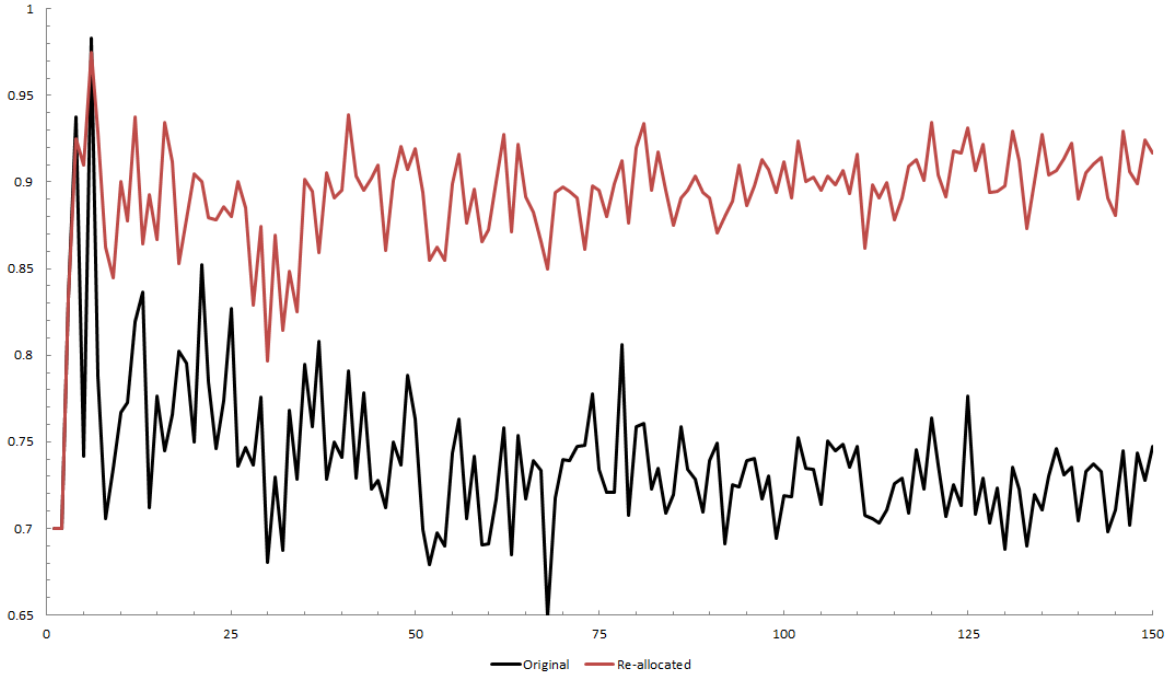

Figure 20. Resource utilisation with the combination of user-defined and provider-defined configurations

In Figure 20, it shows the fluctuation of the resource utilisation levels using the combination of user-defined and provider-defined configuration. The difference of the resource utilisation between the original mapping and the re-allocated mapping is 15.2%.

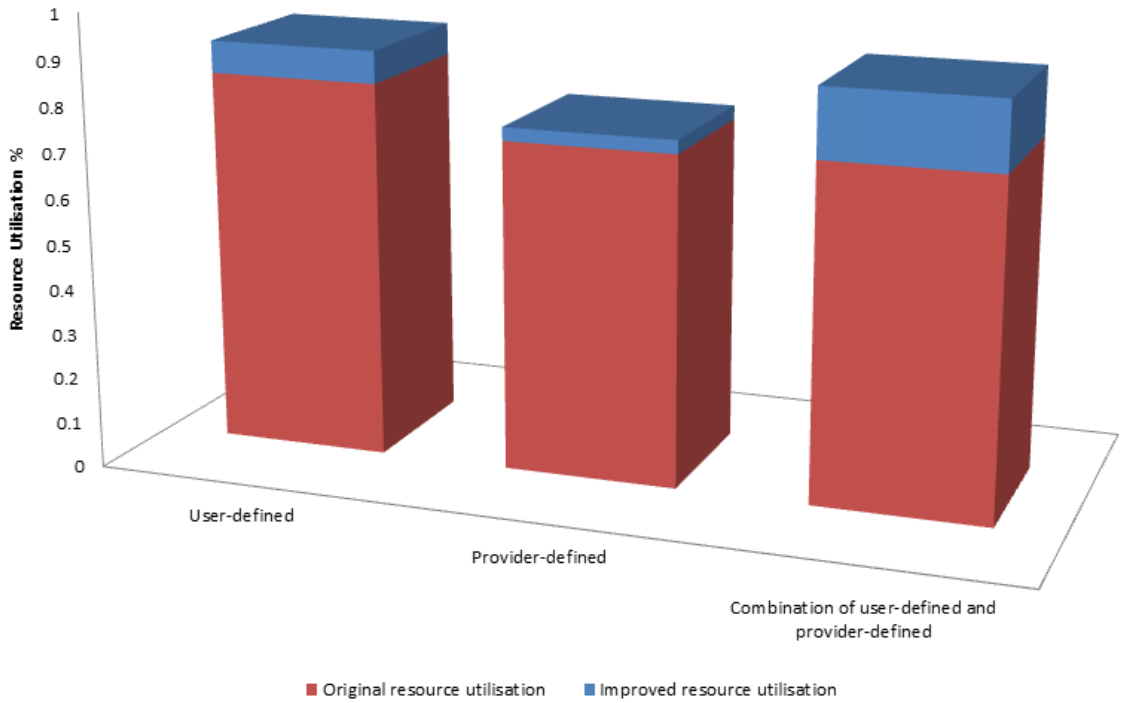

Figure 21. PDF of VM configuration distribution of user-defined and provider-defined configurations

In Figure 21, it shows the comparisons of resource utilisation between user-defined, provider-defined, and the combination of user-defined and provider-defined configurations. We can easily see that the original mapping of user defined is significantly greater than the other two mappings (approximately 10% greater), this is due to the fact that VMs of user-defined configurations can easily find their complementary VMs which makes greater resource utilisation in the Cloud infrastructure. In terms of improved resource utilisation after re-allocation, the combination of the user-defined and provider-defined configurations improved more than the other two. This is because VMs of the combination more easily find their complementary VMs after re-allocation than the other two. Since the resource utilisation using the user-defined configuration is relatively big, the improvement after the re-allocation is small. The resource utilisation using the provider-defined configuration improved the least as it is difficult to find complementary VMs after re-allocation, i.e. limited number of different size of VMs can be chosen from. In general, we conclude that the multi-dimensional heuristics are also suitable for the combination of user-defined and provider-defined configurations.

6 CONCLUSION

Our work in this paper introduces an algorithm for improving resource utilisation for the cloud infrastructure providers. By using a probabilistic multivariate model, the algorithm selects suitable PMs for VM re-allocation before a reconfiguration plan is generated. Our evaluation indicates that there is only a minor decrease in resource utilisation levels that results from reducing number of PMs for re-allocation. Therefore, the approach leads to a lower number of VMs being re-allocated, i.e. less migration costs, as number of PMs considered for VM re-allocation decreases. We presented two heuristics to be used in the algorithms, imbalance heuristics and volume heuristics. For the user-defined configuration, the imbalance heuristics performs slightly better than the volume heuristics. For the provider-defined configuration, the volume heuristic performs significantly better than the imbalance heuristic. This is because it is difficult to find a VM that is complementary to a PM given that the number of types of VM configurations is limited. Furthermore, our evaluation also shows that the multi-dimensional heuristic performs better than non-multidimensional heuristic for the user-defined configuration. For the provider-defined configuration, there is no significant difference on improving resource utilisation level using multi-dimensional or non-multidimensional approaches. We conclude that multi-dimensional heuristics are suitable for cloud infrastructures that offer user-defined configurations and the combination of user-defined and provider-defined configurations.

Our current evaluation is based on simulations of VM allocations. In future work, we will deploy the proposed algorithms in the IC Cloud environment to evaluate its effectiveness. Future work will also include

investigating the relationship between confidence level for selecting suitable PMs and migration costs as they have direct effect on each other and investigate the suitable time interval for the algorithm execution. Moreover, we will also investigate the historical data collected from IC Cloud with the proposed algorithm to help the future capacity planning when new PMs are required.

7 ACKNOWLEDGEMENT

This research is partially supported by the Innovative R&D Team Support Program of Guangdong Province (NO. 201001D0104726115), China.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, *et al.*, "Above the clouds: A berkeley view of cloud computing," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[2] E. Amazon, "Amazon elastic compute cloud (Amazon EC2)," *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[3] G. Infrastructure, "GoGrid Cloud Hosting," ed, 2009.

[4] S. He, L. Guo, and Y. Guo, "Real Time Elastic Cloud Management for Limited Resources," 2011.

[5] W. Jiang, J. Zhang, J. Li, and H. Hu, "A Resource Scheduling Strategy in Cloud Computing Based on Multi-agent Genetic Algorithm," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 11, pp. 6563-6569, 2013.

[6] X. Qu and W. Yingjun, "The Research on Software Resource Re-sharing for Small and Medium-sized Enterprise Cloud Manufacturing System," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, pp. 711-717, 2014.

[7] J. Shahabuddin, A. Chrungoo, V. Gupta, S. Juneja, S. Kapoor, and A. Kumar, "Stream-packing: Resource allocation in web server farms with a qos guarantee," *High Performance Computing—HiPC 2001*, pp. 182-191, 2001.

[8] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *Internet Computing, IEEE*, vol. 13, pp. 14-22, 2009.

[9] L. Guo, Y. Guo, and X. Tian, "IC cloud: a design space for composable cloud computing," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 394-401.

[10] C. Chekuri and S. Khanna, "On multi-dimensional packing problems," 1999, pp. 185-194.

[11] S. Kashyap and S. Khuller, "Algorithms for non-uniform size data placement on parallel disks," *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science*, pp. 265-276, 2003.

[12] F. Hermenier, X. Lorca, J. M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," 2009, pp. 41-50.

[13] J. S. Shahabuddin, K. Balaji, S. Kapoor, S. Juneja, V. Gupta, and A. Chrungoo, "System for optimal resource allocation and planning for hosting computing services," ed: Google Patents, 2005.

[14] M. Mishra and A. Sahoo, "On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach," 2011, pp. 275-282.

[15] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," 2006, pp. 373-381.

[16] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, 2010, pp. 87-92.

[17] R. A. Johnson and D. W. Wichern, *Applied multivariate statistical analysis* vol. 5: Prentice hall Upper Saddle River, NJ, 2002.

[18] T. J. Kimbrel, R. Krauthgamer, B. M. Schieber, M. I. Sviridenko, J. S. Thathachar, and M. Minkoff, "Dynamic resource allocation using known future benefits," ed: Google Patents, 2006.

[19] T. Setzer and A. Stage, "Decision support for virtual machine reassignments in enterprise data centers," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, 2010, pp. 88-94.

[20] L. He, D. Zou, Z. Zhang, K. Yang, H. Jin, and S. A. Jarvis, "Optimizing Resource Consumptions in Clouds," 2011, pp. 42-49.

[21] E. Feller, L. Rilling, and C. Morin, "Energy-Aware Ant Colony Based Workload Placement in Clouds," 2011.

[22] R. N. Calheiros, R. Ranjan, C. A. F. De Rose, and R. Buyya, "Cloudsim: a novel framework for modeling and simulation of cloud computing infrastructures and services," *Arxiv preprint arXiv:0903.2525*, 2009.

BIOGRAPHY OF AUTHORS

Dr. Sijin He received a BSc in Mathematics (1st Class Honour) from University College London in 2007 and a MSc in Computing Science from Imperial College London in 2008. In August 2013, he received PhD in Computer Science at Imperial College London on cloud computing. His main research interests are cloud resource management and cloud security. He is now working on cloud computing and big data at Discovery Science Group (DSG). He was also chief architect for a UK Ministry of Defence project, Rogue Virtual Machine Identification in DaISy Clouds, providing a cloud monitoring framework for intrusion detection in the cloud environment. He is also a key software engineer of a 23M eTRIKS project.

Dr. Li Guo is Lecturer at the School of Computing, Engineering and Computer Science at the University of Central Lancashire. He has PhD from the University of Edinburgh (2007) and has worked as research associate at Imperial College London. His research interests are in cloud computing, distributed sensor informatics, big data analysis and intelligent multi-agent systems, and has more than 40 peer-reviewed publications in these areas. He has contributed to many EPSRC and EU research projects and was the chief architect of the Imperial College Cloud (IC Cloud) system currently in use in a wide variety of Digital Economy projects, and was also chief architect for EU FP6 project-GridEcon providing computational facilities for data analysis services from a variety of sources and devices.

Prof. Yike Guo has been working in the area of data intensive analytical computing since 1995. During last 15 years, he has been leading the data mining group to carry out many research projects, including UK e-science projects such as: Discovery Net on Grid based data analysis for scientific discovery; MESSAGE on wireless mobile sensor network for environment monitoring; BAIR on system biology for diabetes study; iHealth on modern informatics infrastructure for healthcare decision making; UBIOPRED on large informatics platform for translational medicine research; Digital City Exchange on sensor information-based urban dynamics modelling. He was the Principal Investigator of the Discovery Science Platform grant from UK EPSRC where he is leading the team to build the IC Cloud system for large scale collaborative scientific research. He is now the Principal Investigator of the eTRIKS project, a 23M Euro project in building a cloud-based translational informatics platform for global medical research.