Improving Resource Utilisation in the Cloud Environment using Multivariate Probabilistic Models

Sijin He, Li Guo, Moustafa Ghanem, Yike Guo*
Department of Computing
Imperial College London
London, UK
{sh107, liguo, mmg, y.guo}@imperial.ac.uk

***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.**

Keywords- Algorithms; Cloud Computing; Heuristics; Resource Allocation; Resource Management

I. 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. Being able to efficiently utilise cloud resource is a critical issue that directly affects the profits of cloud providers, especially some small and medium-sized cloud providers [2]. The virtualisation technology coupled with cloud reconfiguration algorithms enables more efficient cloud resource utilisation in Internet Data Centres.

For better resource utilisation, many cloud providers start with static allocation of VMs to physical machines (PMs) using a resource scheduler as shown in Figure 1. However, the resource utilisation in the cloud environment decreases as the number of VMs increases in the PMs. This is due to mainly the following two reasons. (1) There are always some unallocated resources left on the PMs. This is due to the unpredictable incoming VM configurations from cloud users. Without accurate historical data, it is difficult to allocate VM properly. (2) Memory balloon and CPU virtualisation are widely supported features of virtualisation technology. It enables cloud users to configure their VMs memory size and CPU number at operating time, which makes the scheduler work less efficient (See [2] for more detail). These two reasons make the static scheduling approach unfit, and instead, requiring dynamic re-allocation. Therefore, a dynamic VM re-allocation on the PMs is required for improving resource utilisation in the cloud environment.

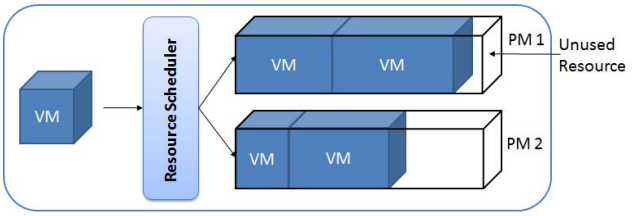

Figure 1. Static VM allocation

Cloud reconfiguration algorithms are based on VM re-allocation techniques that construct a suitable reconfiguration plan for achieving greater resource utilisation in the cloud environment. Existing cloud reconfiguration algorithms [3, 4, 5] aim to solve the problem of low PM resource utilisation to allow more VMs to be allocated in the cloud environment. They mainly comprise of two processes: Target Mapping Generation (TMG) and Migration Plan (MP), as shown in Figure 2. Such algorithms firstly compute a VM-to-PM target mapping in TMG which guarantees better resource utilisation than the existing VM-to-PM mapping, and then 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 environments into consideration for VM re-allocation which results in high migration costs. As the number of PMs increases in the cloud environment, 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.

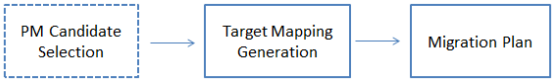

Figure 2 Cloud Reconfiguration Processes

In this paper, we introduce an algorithm for improving resource utilisation for cloud providers. It uses a multivariate probabilistic normal distribution model to select suitable PMs for VM re-allocation before a reconfiguration plan is generated, which leads to less number of VMs being re-

* Please direct your enquiries to the communication author Professor Yike Guo.

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.66

allocated (i.e. less migration costs). We call this procedure as PM Candidate Selection (PMCS). Two heuristic metrics are considered, i.e. imbalance and volume, which exhibit multi-dimensional characteristics of VMs and PMs for achieving better resource utilisation. By combining these two heuristic metrics in our experiments, we observed that our approach improves the resource utilisation level for cloud providers.

The rest of the paper is organised as follows: Section II reviews the related work of cloud reconfiguration algorithms and related problems. Section III describes mathematical background of our approach. Section IV presents our cloud reconfiguration algorithm. Experimental evaluations of our work are presented in Section V, and we conclude the paper and suggest some future work in Section VI.

II. RELATED WORK

A. Multi-dimensional Bin Packing Problem

The VM re-allocation approach is an instance of the well known Multi-Dimensional Bin Packing (MDBP) problem [6] which has been mostly studied by means of simulations in several works [3, 7, 8]. It is well-known that finding optimal solutions to MDBP is NP-hard. Hermenier et. al [3] 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 limited for the computation which could lead to the output being not as good as heuristic approaches.

Due to the NP-hard nature of the 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) [9] is proposed to solve the problem with a low approximation ratio. Kimbrel et al. [7] 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 exisitng approaches used still ignore the multi-dimensional characteristics of VMs and PMs [10].

B. Cloud Reconfiguration Algorithms

Researching on cloud reconfiguration algorithms is an active research area of Cloud computing. One problem addressed is trying to minimise number of PMs in the cloud environment. This is commonly referred as VM consolidation. The static re-allocation approach [11] is a simple heuristic for the MDBP problem and applies it to minimise the number of PMs required to host a given web traffic. A resource management algorithm for grids [8] attempts to minimise the number of migrations of VMs while minimising the number of PMs used. A similar objective is also pursued in [12] and [13]. These approaches, however, have focused on how to calculate a new configuration, and have neglected the migration overhead. An algorithm [14] is proposed to pack VMs according to their CPU needs while minimising the number of migrations. They have formulated the problem of dynamic placement of applications in virtualised heterogeneous systems as continuous optimisation: at each time frame the placement of VMs is optimised to minimise power consumption and maximise performance. The authors 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. Some interesting approaches have been conducted using genetic algorithm [4] and Ant Colony Optimisation [5].

Researchers have also applied a variety of methods to achieve greater resource utilisation. Prediction techniques and queuing theory results [15] are employed to allocate resources efficiently within a single PM serving a web workload. It proposes an optimisation algorithm for resource economy [16] that allocates PM resources depending on the expected financial gain for the hosting centre. Similar work is presented in [17] which is on the top of an economic model applies the feedback control loop to drive resource allocation. Resource overlooking is advocated [18] as a means of increasing the revenue generated by available resources in a shared hosting platforms. The authors focus on providing a fair schedule while minimising job migrations. Sandpiper [19] is a reconfiguration engine, based on an FFD heuristic, to relocate VMs from overloaded to under-utilised nodes. When a migration between two nodes is not directly feasible, the system identifies a set of VMs to swap in order to free a sufficient amount of resources on the destination node. Then the sequence of migrations is executed. This approach is able to solve simple replacement issues but requires some space for temporarily hosting VMs on either the source or the destination node. Han et al. [20] employ queuing model based techniques to guide the VM allocation in application scaling.

III. MATHEMATICAL BACKGROUND

As mentioned earlier, the problem of VM-to-PM mapping is an instance of the MDBP problem, in which the PMs represent the bins and VM the items to be packed. An IDC environment consists of a set of PMs, $\mathrm{PM} \coloneqq \{PM_0, \dots, PM_v, \dots, PM_{m-1}\}$ with $m = |PM|$, each of which hosts one or more VMs, $PM_i \coloneqq \{VM_0, \dots, VM_m\}$. The types of available resources (i.e. CPU cycles, CPU cores, RAM size, network bandwidth and disk size, etc.) in PMs are defined by the set D with $d = |D|$. The allocation of VMs to PMs can be considered as a VM-to-PM mapping M.

Representing the quantities of various resources as vectors is a traditional approach for VM resource management [21, 22]. 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 from cloud-users. We model the configuration of a VM, say VM_i, as the capacity of the VM represented by a d-dimensional capacity vector $\vec{R}_i$, which is the vector addition of normalised capacity vectors of each resource type.

Each PM, say PM_k, has a fixed total capacity vector $\vec{C}_k$ in d-dimensional space. Without loss of generality we assume that the values of $\vec{C}_k$ for all PMs have been normalised to 1. The resource utilisation vector $\vec{L}_k$ of PM_k is the vector additions of all capacities vectors of VMs that reside on PM_k, i.e. $\vec{L}_k = \sum_{i \in VM\ in\ PM_k} \vec{R}_i$. Note that the capacity vectors of VMs are also normalised with respect to the total capacity of their hosting PM.

For simplicity, we consider three major resource types, i.e. CPU, memory and I/O as shown in Figure 3. (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 $\binom{0}{0}$ to point $\binom{1}{1}$. 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.

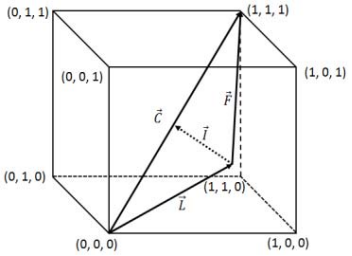

Figure 3. Vector Representation of a PM

Since vectors in PMs and VMs are multi-dimensional, they must exhibit some multi-dimensional characteristics. Two typical characteristics (imbalance and volume) are defined which are used as heuristic metrics in our cloud reconfiguration algorithm.

A. Imbalance

Imbalance 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 $\vec{R}_i$'s projection on $\vec{C}$ and $\vec{R}_i$.

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 $\vec{I}$ [21] is given by

$$\vec{I} = \left(c - \frac{c+m+o}{3}\right)\vec{\imath} + \left(m - \frac{c+m+o}{3}\right)\vec{\jmath} + \left(o - \frac{c+m+o}{3}\right)\vec{k} \quad (1)$$

The magnitude $|\vec{I}|$ of $\vec{I}$ which measure the degree of imbalance is calculated as

$$|\vec{I}| = \sqrt[2]{\left(c - \frac{c+m+o}{3}\right)^2 + \left(m - \frac{c+m+o}{3}\right)^2 + \left(o - \frac{c+m+o}{3}\right)^2} \quad (2)$$

We can generalise Equation 2 into multi-dimensions D as

$$|\vec{I}| = \sqrt[|D|]{\sum_{d \in D}\left(\vec{L}_d - \frac{\sum_{\delta \in D}\vec{L}_\delta}{|D|}\right)^{|D|}} \quad (3)$$

where $\vec{L}_d$ is the resource utilisation of the dimension d of a PM. The average magnitude of imbalance of m PMs in the entire cloud environment is $\frac{\sum_{i=1}^{m}|I_i|}{m}$.

B. Volume

Volume is another multi-dimensional measure that indicates the size of resource utilisation of a PM, i.e. the capacity of a VM. We define volume V as a product of every single resource in a resource vector.

$$V(\vec{A}) = \prod_{i \in D} \vec{A}_i \quad (4)$$

Suppose that a VM with a resource utilisation vector $\vec{L}$ is represented by $\begin{pmatrix} c \\ m \\ o \end{pmatrix}$, the volume of the resource utilisation vector is defined as $c \times m \times o$.

C. VM Configuration Multivariate Normal Distribution

In generally, a VM configuration is defined by specifying the number of CPUs required and the associated memory requirements, etc. This can be expressed as the capacity vector $\vec{R}$. For example, a capacity vector $\binom{2}{2}$ of a VM represents as a VM configuration with 2 CPU and 2 GB memory.

Nowadays, there are two key approaches for defining VM configurations based on taking provider-defined and user-defined views. In the provider-defined view, cloud providers, such as Amazon EC2 [23], GoGrid [24], predefine limited number of types of VM configurations for cloud users. In the user-defined view, cloud providers, such as IC Cloud [25], allow their cloud users to define VM configurations. After a VM configuration is defined, it will be sent to the cloud providers for approval. If the requested resources are available in the cloud environment, the request will be accepted and then the cloud provider will create and allocate the requested VM to a suitable PM. This strategy ensures that all VMs in the cloud environment will have sufficient CPU and memory resources as requested by cloud users. This leads to a large number of different VM configurations created in the cloud environment. Therefore, 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 [26]. 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 and Σ is a $d \times d$ covariance matrix.

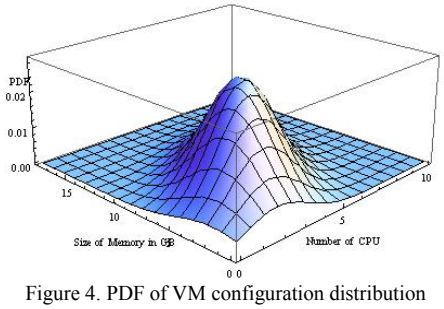

Figure 4. PDF of VM configuration distribution

In Figure 4, 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 [25]. The PDF of the VM capacity vector $\vec{R}$ can be calculated as shown in Equation 5. Let x be $\vec{R}$, the PDF of $\vec{R}$ is given by

$$p(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \quad (5)$$

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

D. Resource Utilisation Level

The objective of the cloud reconfiguration algorithm is to maximise the utilisable resources among PMs. It is essential to measure the resource utilisation of a VM-to-PM mapping before and after VM re-allocation. Since there is more than one resource type in a PM, resource utilisation of only a single type of resource is not sufficient to indicate the overall resource utilisation of the whole PM. Thus, we define a metric to measure the resource utilisation 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.

We firstly define a Mean Vector Ratio (MVR) which represents the ratio of a resource vector to the mean VM capacity vector $\vec{\mu}$ for a single resource type as shown in Equation 6.

$$MVR(\vec{A}_d^{PM_i}) = \vec{A}_d^{PM_i} / \vec{\mu}_d \quad (6)$$

We then define two functions, $f(\vec{A}^{PM_i})$ and $g(\vec{A}^{PM_i})$. $f(\vec{A}^{PM_i})$: Given a resource vector $\vec{A}^{PM_i}$, the number of the mean VM capacity vector $\vec{\mu}$ can be hosted in $\vec{A}^{PM_i}$, regardless of any remaining proportion of $\vec{\mu}$ as shown in Equation 7. For example, given a VM capacity vector $\binom{1.5}{1}$, let $\vec{A}^{PM_i}$ be $\binom{2}{3}$, $f(\vec{A}^{PM_i})$ is $\min\{\lfloor 2/1.5 \rfloor, \lfloor 3/1 \rfloor\}$ which is equal to 1. Therefore, $\vec{A}^{PM_i}$'s capacity is equal to a size of 1 mean VM capacity.

$$f(\vec{A}^{PM_i}) = \min_{d \in D}\{\lfloor MVR(\vec{A}_d^{PM_i}) \rfloor\} \quad (7)$$

$g(\vec{A}^{PM_i})$: Given a resource vector $\vec{A}^{PM_i}$, the number of the mean VM capacity $\vec{\mu}$ can be hosted in $\vec{A}^{PM_i}$ taking all remaining proportion of $\vec{\mu}$ into account as shown in Equation 8.

$$g(\vec{A}^{PM_i}) = \min_{d \in D}\{MVR(\vec{A}_d^{PM_i})\} \quad (8)$$

Given a VM-to-PM mapping M, there are m PMs in the mapping. Each PM_i 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 (M) , is shown in Equation 9.

$$\psi(\mathrm{M}) = \frac{g(\sum_{i \in m} \vec{L}^{PM_i}) + \sum_{i \in m} f(\vec{F}^{PM_i})}{g(\sum_{i \in m} \vec{C}^{PM_i})} \quad (9)$$

$g(\sum_{i \in m} \vec{L}^{PM_i})$ and $g(\sum_{i \in m} \vec{C}^{PM_i})$ are constant resources as they are not changed before and after VM re-allocation. $f(\vec{F}^{PM_i})$ calculates the number of the mean VM capacity $\vec{\mu}$ the remaining capacity of PM_i can host, any unallocated resource will be considered as a waste. The value of $f(\vec{F}^{PM_i})$ can vary depending on different VM-to-PM allocations.

IV. Cloud Re-allocation Algorithm

This section details the assumptions of this algorithm and provides a detailed algorithm description of the VM re-allocation problem as the MDBP problem. As we mentioned earlier, there are three stages in the algorithm: PM Candidate Selection (PMCS), Target Mapping Generation (TMG) and Migration Plan (MP). PMCS is a process that selects suitable PMs for VM re-allocation, this would potentially reduce number of PMs required to be re-allocated and hence reduce the total migration costs. TMG produces a new VM-to-PM target mapping for improving resource utilisation in the cloud environment by using heuristic approaches. Finally, MP generates a VM migration plan from the current mapping to the target mapping with minimum migration costs.

A. Assumptions

The algorithm requires a history of VM configurations from cloud users, a multivariate normal distribution is built based on the history. The 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 environement and may need to be re-allocated. The algorithm schedules VMs according to this information.

The algorithm is then executed at regular time intervals (e.g. every hour), for calculating the resource utilisation level of the current mapping in the cloud environment and computing for a new target mapping. If the resource utilisation level of the target mapping is greater than the current mapping, the algorithm will proceed, otherwise, it will be aborted.

B. PM Candidate Selection

In this procedure, a set of suitable PMs is selected from all PMs in the cloud environment for VM re-allocation. Each PM may have some remaining resources which may host one or more VMs. The remaining capacity $\vec{F}$ of every PM is input to Equation 5 to obtain the corresponding PDF $p(\vec{F})$.

Since a new incoming VM will be allocated to the remaining capacity of a PM, it is essential to choose a PM according to its remaining capacity. Our approach is to choose PMs whose remaining capacity is not in a similar size of the mean VM capacity. This is because the mean VM capacity is the most frequent VM configuration defined 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 better resource utilisation 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. For choosing a set of suitable PMs, a range of PDF is required to be defined which is used for selecting suitable 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. In our study, we set the range as $p(\vec{\mu} \pm 1.96\sigma)$. We choose PMs whose remaining capacities are not in the range, i.e. 5% of the area, where the VM configurations rarely occur in the distribution. Furthermore, the range can be adjusted by cloud providers according to their demands for resource utilisation level in the cloud environments as better resource utilisation level can be achieved by selecting more number of PMs which can be adjusted by narrowing the range.

1) Grouping

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. In order to keep PM in a balanced manner after VM re-allocation, PMs are classified into groups according to the resource utilisation of each resource type.

Sometimes, a PM may have more resource utilisation along the memory axis compared to the CPU axis which causes the imbalance of PM. Ideally we aim to find a VM which is less utilised in memory compared to CPU so as to achieve resource balance in PMs after VM re-allocation. Therefore, we classify PMs into different groups according to their resource utilisation of each resource type [21] before generating a new target mapping.

We divide PMs into $d!$ 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 5. We now have 3! groups, i.e. 6 groups. For example, for group COM, CPU is the most utilised resource, memory is least utilised and I/O is in between the other two. For group MOC, memory is most utilised, CPU is least and network is in between the two. Thus, group COM and group MOC are complementary to each other in terms of resource utilisation. Grouping is an efficient way of identifying complementary PMs. For instance, if we have a PM whose $Memory > CPU > I/O$, the PM will fall into group MCO. Its complementary PM will be a PM in exactly opposite group, i.e. group OCM. The same methodology also applies to VMs.

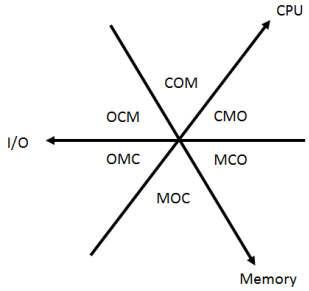

Figure 5. Grouping by resource utilisation

C. Target Mapping Generation

In this paper, 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 suitable PMs. For imbalance heuristic, we group the list of suitable PMs into $d!$ PM groups according to their resource utilisations. After grouping, there are $d!/2$ sets of PMs and each set is used as input for TMG. 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. For volume heuristic, the whole list of suitable PMs is taken as an input.

1) Imbalance Heuristic

By considering three resource types, we have 3 sets of PMs for TMG. For each set of PMs, there are two complementary PM groups. We then classify VMs in the PM groups into two new VM groups according to their resource capacities. For example, given two complementary PM groups, one is COM group (CPU-intensive) and the other is MOC (Memory-intensive) group, a CPU-intensive VM in a PM group will be placed to a CPU-intensive VM group called $\boldsymbol{Pos}$ whilst a Memory-intensive VM will be placed to a Memory-intensive VM group $\boldsymbol{Neg}$. If a VM is neither CPU nor Memory-intensive, it can be placed to either VM groups.

Two complementary VM groups out of $d!/2$ groups are then input to Algorithm 1 for constructing a new VM-to-PM mapping in turn. VMs in the VM groups are sorted by the magnitude of the imbalances of their VM capacities in descending order. (line 3 and 4), i.e. a VM with greater magnitude of the imbalance will be allocated to a PM first. $\boldsymbol{PMList_k}$ is a list that contains all suitable PMs with no VM allocation at the beginning of the algorithm (line 5),

where $k \in d!/2$. PMs in $\boldsymbol{PMList_k}$ 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 in the two VM groups to PMs in $\boldsymbol{PMList_k}$. Every VM always tries to find a PM which is complementary to the VM (line 8 to 18) which results in better degree of imbalance of PMs after allocation. The algorithm outputs $\boldsymbol{PMList_k}$ with new VM allocation. The algorithm executes $d!/2$ times until all new VM allocation in different $\boldsymbol{PMList_k}$ are generated. The target mapping is an aggregation of all PMs in $\boldsymbol{PMList_k}$ and the PMs that have not been chosen for TGM.

Algorithm 1: Imbalance Heuristic
Input: $\boldsymbol{Pos}$, $\boldsymbol{Neg}$

1. $\boldsymbol{Pos} = \{VM_1, \dots, VM_i, \dots VM_n\}$
2. $\boldsymbol{Neg} = \{VM_{n+1}, \dots, VM_j, \dots, VM_m\}$
3. $\boldsymbol{Pos} = SortByImbalance(\boldsymbol{Pos})$
4. $\boldsymbol{Neg} = SortByImbalance(\boldsymbol{Neg})$
5. $\boldsymbol{PMList_k} = \{PM_1, \dots, PM_k\}$
6. **while** $(!\boldsymbol{Pos}.empty() \,\&\, !\boldsymbol{Neg}.empty())$
7. $\quad \boldsymbol{PMList_k} = SortByImbalance(\boldsymbol{PMList_k})$
8. $\quad$ **for** $(PM_k \; in \; \boldsymbol{PMList_k})$
9. $\quad\quad$ **if** $(PM_k$ **complementary to** VM_i **in** $\boldsymbol{Pos})$
10. $\quad\quad\quad PM_k.put(VM_i)$
11. $\quad\quad\quad \boldsymbol{Pos}.remove(VM_i)$
12. $\quad\quad\quad$ **break**
13. $\quad\quad$ **else**
14. $\quad\quad\quad PM_k.put(VM_j)$
15. $\quad\quad\quad \boldsymbol{Neg}.remove(VM_j)$
16. $\quad\quad\quad$ **break**
17. **return** $\boldsymbol{PMList_k}$

2) Volume Heuristic

By taking the whole set of suitable PMs into account, a new VM group called $\boldsymbol{VMList}$ is created where all VMs belonging to the set of suitable PMs are placed to $\boldsymbol{VMList}$. It is used as an input to Algorithm 2 for constructing a VM-to-PM mapping. The VMs in $\boldsymbol{VMList}$ are sorted by volume of VMs in descending order (line 2). Similar to the imbalance approach, $\boldsymbol{PMList}$ is a set that contains all suitable PMs with no VM allocation at the beginning of the algorithm (line 3) and they are sorted by the volume of remaining capacity of 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 $\boldsymbol{PMList}$ first (line 4 to 9). The algorithm finally outputs $\boldsymbol{PMList}$. The target mapping is an aggregation of PMs in $\boldsymbol{PMList}$ and the PMs that have not been chosen for TGM.

Algorithm 2: Volume Heuristic
Input: $\boldsymbol{VMList}$

1. $\boldsymbol{VMList} = \{VM_1, \dots, VM_i, \dots VM_n\}$
2. $\boldsymbol{VMList} = SortByVolume(\boldsymbol{VMList})$
3. $\boldsymbol{PMList} = \{PM_1, \dots, PM_k\}$
4. **while** $(!\boldsymbol{VMList}.empty())$
5. $\quad \boldsymbol{PMList} = SortByRemainingVolume(\boldsymbol{PMList})$
6. $\quad$ **for** $(PM_i \; in \; \boldsymbol{PMList})$
7. $\quad\quad PM_i.put(VM_j)$
8. $\quad\quad \boldsymbol{VMList}.remove(VM_j)$
9. $\quad\quad$ **break**
10. **return** $\boldsymbol{PMList}$

The resource utilisation level of a target mapping can be calculated using Equation 9. If the resource utilisation level of the target mapping is greater than the original VM-to-PM mapping, migration plan will then be executed.

D. 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. The solution to the constraint programming [3] guarantees the minimum moves. The detail of generating a migration plan with the lowest migration cost is not covered in this paper. More migration plan algorithms can be found in [2, 4, 5].

V. 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 [27]. 3 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 environment are homogeneous. Each PM has 36 CPUs, 36 GB memory 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.

A. 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 [25] for building a multivariate normal distribution. The 3-dimensional mean VM capacity vector and the covariance matrix were then calculated from the distribution. 150 target VM-to-PM mappings are generated for this experiment. In the first iteration, we set the number of PMs available for VM allocation to be 1, we then begin with allocating a set of VMs with random configurations based on the normal distribution to the PM using First-fit algorithm as it is a one of the most practical approaches for initial VM allocation [25]. If the PM does not have sufficient resources to host a VM, the VM allocation will stop or the VM will be allocated to the next available PM with sufficient resource. After that, the resource utilisation level of this mapping (original mapping) is calculated using Equation 9. Our cloud reconfiguration algorithm is then applied to construct a new

target VM-to-PM mapping. The resource utilisation level of the target mapping is then calculated. We repeat this process by incrementing 1 PM in each iteration until the number of target mappings reaches 150.

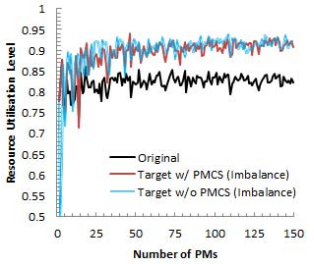

Figure 6(a) w/ &w/o PMCS

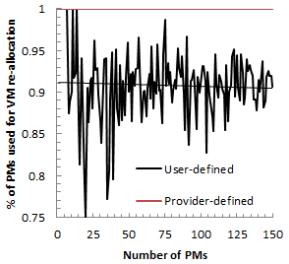

Figure 6(b) PMs used for re-allocation

Without performing PMCS, the average resource utilisations 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 6(a) and 7(a). 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 number of PMs considered for VM re-allocation decreases by 9.17% on average by performing PMCS as number of PMs increases in the cloud environment as shown in Figure 6(b). We conclude that reducing number of PMs used for VM re-allocation does not have a significant impact on the resource utilisation levels of the target mappings; instead, it results in less migration costs in the migration plan as number of PMs considered for VM re-allocation decreases.

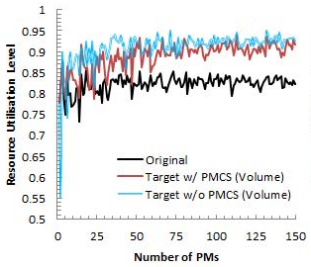

Figure 7(a) w/ &w/o PMCS

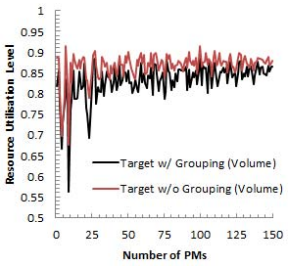

Figure 7(b) w/ &w/o grouping

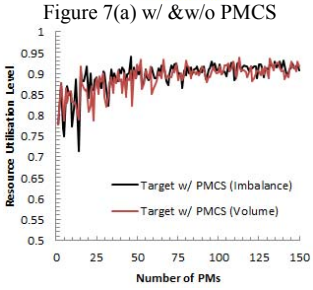

Figure 8(a) User-defined

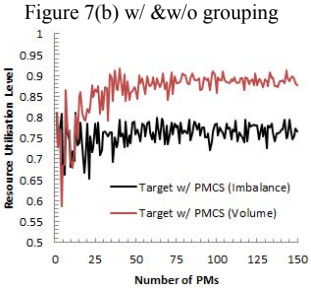

Figure 8(b) Provider-defined

We also notice that grouping helps imbalance heuristic to achieve better resource utilisation level. However, volume heuristic without grouping performs slightly better than the one with grouping as shown in Figure 7(b). This is because volume heuristic does not take advantage of grouping as grouping only works well for finding complementary VMs to PMs.

Figure 8(a) shows the fluctuation of the resource utilisation levels using imbalance and volume heuristics with PMCS 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. To achieve better resource utilisation level, we combine the two heuristics by taking the one with higher resource utilisation level. This results in an increase of resource utilisation level from the original mapping by 7.71%. In Figure 10(a), 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.

B. 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, we collected 10 types of VM configurations from Amazon EC2. 150 target VM-to-PM mappings are also generated for this experiment. The same experimental methodology is also applied to this experiment, but the set of VM for initial allocation is generated from the 10 types of the Amazon EC2 VM configurations.

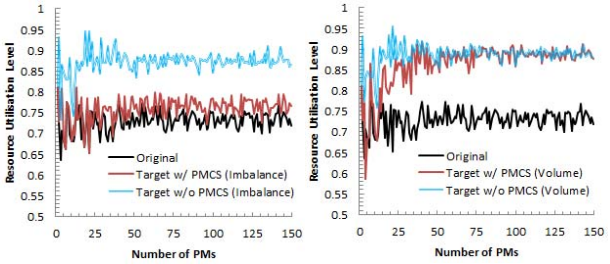

Figure 9. Resource Utilisation w/ &w/o PMCS (Provider-defined)

As shown in Figure 6(b), approaches with PMCS in the provider-defined view do not perform well as it selects all PMs as suitable PMs for VM re-allocation. Figure 8(b) shows the fluctuation of the resource utilisation levels using imbalance and volume heuristics with PMCS and the difference between them is 10.54%. In addition, without performing PMCS, the average resource utilisations for both imbalance and volume heuristics are better than the ones with PMCS by 11.30% and 2.17% respectively as shown in Figure 9. In overall, the imbalance heuristic does not perform well as it is difficult to find a VM which is complementary to a PM given that the number of types of VM configurations is limited.

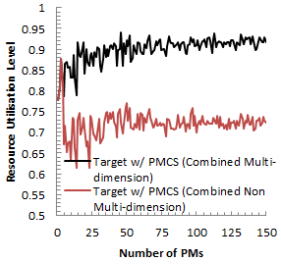

Figure 10(a) User-defined

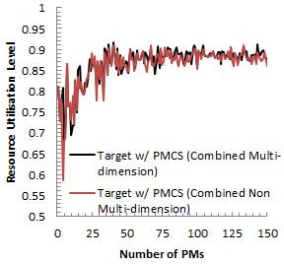

Figure 10(b) Provider-defined

In Figure 10(b), by combining two multi-dimensional heuristics, the resource utilisation level increases by 13.71% in comparison with the original mapping. Further, it performs better than the combination of each single resource

type by only 0.49% on average. This shows that there is no significant difference on improving resource utilisation level using these two approaches given that the number of types of VM configurations is limited.

VI. Conclusion and Future Work

Our work in this paper introduces an algorithm for improving resource utilisation for cloud 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 algorithm, imbalance and volume. In the user-defined view, the imbalance heuristic performs slightly better than the volume heuristic. In the provider-defined view, the volume heuristic performs significantly better than the imbalance heuristic. This is because it is difficult to find a VM which 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 in the user-defined view. In the provider-defined view, there is no significant difference on improving resource utilisation level using multi-dimensional or non-multidimensional approaches.

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 predefined range 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.

References

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

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

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

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

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

[6] 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.

[7] T. Kimbrel, B. Schieber, and M. Sviridenko, "Minimizing migrations in fair multiprocessor scheduling of persistent tasks," *Journal of Scheduling,* vol. 9, pp. 365-379, 2006.

[8] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic Application Container: A Lightweight Approach for Cloud Resource Provisioning," *2012 26th IEEE International Conference on Advanced Information Networking and Applications,* pp. 15-22, 2012.

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

[10] T. Setzer and A. Stage, "Decision support for virtual machine reassignments in enterprise data centers," 2010, pp. 88-94.

[11] 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.

[12] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic application placement under service and memory constraints," *Experimental and Efficient Algorithms,* pp. 391-402, 2005.

[13] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," 2007, pp. 119-128.

[14] R. Nathuji, K. Schwan, A. Somani, and Y. Joshi, "Vpm tokens: virtual machine-aware power budgeting in datacenters," *Cluster computing,* vol. 12, pp. 189-203, 2009.

[15] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," 2003, pp. 381-398.

[16] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," 2001, vol. 35, pp. 103-116.

[17] R. Levy, J. Nagarajarao, G. Pacifici, A. Spreitzer, A. Tantawi, and A. Youssef, "Performance management for cluster based web services," 2003, pp. 247-261.

[18] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," *ACM SIGOPS Operating Systems Review,* vol. 36, pp. 239-254, 2002.

[19] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," 2007, vol. 7, pp. 11–13.

[20] R. Han, L. Guo, Y. Guo, and S. He, "A Deployment Platform for Dynamically Scaling Applications in The Cloud," *2011 Third IEEE International Conference on Coud Computing Technology and Science,* pp. 506-510, 2011.

[21] 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.

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

[23] E. C. Amazon, "Amazon elastic compute cloud," *Retrieved Feb,* vol. 10, 2009.

[24] S. P. D. Hosting, "gogrid Cloud Hosting. Available on the WWW, 2009."

[25] L. Guo, Y. Guo, and X. Tian, "IC Cloud: A Design Space for Composable Cloud Computing," 2010, pp. 394-401.

[26] R. A. Johnson and D. W. Wichern, *Applied multivariate statistical analysis*, vol. 4: Prentice Hall Upper Saddle River, NJ, 2002.

[27] 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.