On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach

Mayank Mishra and Anirudha Sahoo
Department of Computer Science and Engineering
Indian Institute of Technology Bombay, Mumbai, India 400076
Email: {mayank,sahoo}@cse.iitb.ac.in

***Abstract*—In this paper, we present the methodologies used in existing literature for Virtual Machine (VM) placement, load balancing and server consolidation in a data center environment. While the methodologies may seem fine on the surface, certain drawbacks and anomalies can be uncovered when they are analyzed deeper. We point out those anomalies and drawbacks in the existing literature and explain what are the root causes of such anomalies. Then we propose a novel methodology based on vector arithmetic which not only addresses those anomalies but also leads to some interesting theories and algorithms to tackle the above mentioned three functionalities required in managing resources of data centers. We believe that with a strong mathematical base, our methodology has the potential to become the foundation of future models and algorithms in this research area.**

I. Introduction

An important enabler for cloud computing [3] is the Virtualization technology [11]. The ability to virtualize the Physical Machine(PM) gives immense benefits in terms of reliability, efficiency, scalability and makes computing available as an utility service. Virtualization, coupled with migration capability, enables the data centers to consolidate their computing needs and use lesser number of PMs. Virtualization has enabled data centers to be more elastic and scalable.

VM placement and migration techniques are required for load balancing, server consolidation and hotspot mitigation. Thus, VM placement is an important aspect of data center resource management. The general approach for handling these problems is to have a mathematical representation or a metric of resource utilizations by different VMs and PMs and maintain the balance using this metric. This metric is typically a function of resource utilization of individual resource types. Depending on the sophistication of the metric, it can be used for purposes ranging from simple VM placement to measuring degree of imbalance in resource utilizations in PM.

The problem of VM placement and migration consists of two distinct parts. The first part is to correctly estimate the VM resource requirements. This is a crucial and difficult step since a VM keeps changing its resource requirement dynamically. There are many proposed approaches for estimating the resource requirements of VMs [12]. One common approach is to base this estimation on history of resource utilization of the VM. Once the resource requirements of VMs are properly estimated, the next part is to use a VM placement strategy to achieve efficient resource utilization of PMs. In this paper, we assume that resource requirement of the VMs is known and hence we concentrate on the second part of the problem.

There are few research work reported in the literature for VM placement. Those methods might look fine at a glance, but a deeper scrutiny can expose various anomalies and drawbacks which might affect the performance of the system. Most of them devise a metric, which is a function of resource utilizations of individual resource types. They use this metric for placement and migration of VMs as well as for load balancing and consolidation of servers. In this paper, we present various methodologies used in the literature for VM placement, server load balancing and server consolidation and point out the drawbacks and anomalies in those methodologies and discuss the root cause of such anomalies. Then we propose a novel methodology based on vector arithmetic which not only addresses those anomalies but also leads to some interesting theories and algorithms to tackle the above mentioned three functionalities required in managing resources of data centers. We believe that with a strong mathematical base, our methodology has the potential to become the foundation of future models and algorithms in this research area.

II. Related Work

There have been many methodologies reported in the literature for VM placement and migration. Most of them use a metric to make VM placement/migration related decisions. These metrics vary greatly in terms of their usefulness. Some approaches use metric which is a weighted sum of the resources [4] while some others use a more complex mathematical function of resources [13]. The estimation of the resource requirement for VMs is a field of research in itself. Resource estimation requires analysis of logs of the past resource usage of the applications which are intended to be hosted on VMs. As there is overhead involved in running applications in virtualized environment, the resource requirement of an application would be different when it runs on a VM than when it runs on a PM. An overhead estimation model for application is also required for finding accurate resource needs on virtual platforms. Various approaches have been proposed in the literature for this purpose. In [12], authors discuss a microbenchmark based overhead estimation approach for application's resource requirement in different virtualization architectures. Paper [6] discusses various models for application performance prediction in virtualized environment. Methodology used in [9] employs forecasting techniques to estimate the resource requirements of VMs along with the time and cost of migrations to fulfill changes in resource requirement. The estimation of application resource requirement

also depends upon the underlying virtualization framework. An application can have different overheads based on the type of virtualization (Paravirtulization [5], [**?**], hardware virtualization [2]) used in the system. Discussion on different types of virtualizations and a comparison among them can be found in [11], [7].

In this paper, we focus on problems related to VM Placement, Load Balancing and Consolidation. In the next section, we present a detailed description of some of the well known approaches used to address these problems.

III. Methodologies used in Existing Literature

In the existing literature, VM placement problem has been stated to be similar to multi-dimensional *bin packing problem* [13]. In this section, we first present a short comparison between *bin packing* [1] and VM placement problem so that the exact difference between them is clear. Then we present some methodologies reported in existing literature for VM placement, server load balancing[13] [10] and server consolidation[8]. We point out the metrics used in these methodologies. The choice of metric is very crucial for the kind of operation it is intended for. Some metrics are suitable for VM placement, but can be ineffective for server load balancing and vice versa. We will show that some of metrics and methodologies used in the current literature may look fine but a deeper analysis would reveal the subtle problems inherent in them. We bring out problems these methodologies exhibit and point out the root cause of the problem. Lessons learnt from the pitfalls of discussed methodologies enable us to propose a novel scheme for VM placement, server load balancing and server consolidation.

A. Comparison Between VM Placement and Bin Packing Problem

If a VM is considered to be a three dimensional object (assuming cpu, memory and I/O as the three dimensions) then the problem of placing the VMs over the PMs looks similar to the three dimensional bin packing problem. But they are not exactly the same. In 3D bin packing problem, a set of three dimensional objects (generally cuboids) are required to be placed inside three dimensional containers (also cuboids). The aim is to pack as many objects in the containers as possible, so that the number of containers required is minimized. While packing objects into a given container, two objects can be placed side by side or one on top of the other. But if we consider VMs as the objects, then placing VMs side by side or one on top of the other is not a valid operation. This is because once a resource is utilized or occupied by a VM, it can not be reused by any other VM. This is the main difference between three dimensional bin packing and VM placement problem. We depict this difference in Figure 1. For simplicity, we show resources in two dimensions in a PM. A certain amount of resources has been used up in the PM (shown as lightly shaded rectangle). When a new VM (shown as a dark shaded rectangle) is to be placed in this PM, the only allowed position for this rectangle is the one shown with a '✓' mark. In case of bin packing problem, the positions marked 'X' are also allowed, whereas they are not allowed in VM placement.

The problem of placing the VMs over the PMs is actually similar to Vector Packing Problem which is also a NP-Hard problem. A paper which describes the approximation solutions for the Vector packing problem is [**?**].

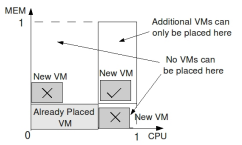

Fig. 1. VM Placement in Two Dimension Resource Space

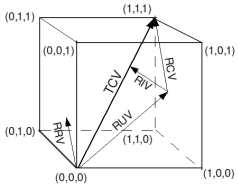

Fig. 2. Depiction of Various Vectors used

B. Terminologies Used

In this section we would like to introduce few terminologies which we will use in this paper to help us explain the existing as well our methodologies better. Please refer to Figure 2 where the quantities are illustrated. We consider three major resources available with the PM namely CPU, Memory and I/O. These resources form the three dimensions of an *abstract* object. We normalize the resources along each axis. Hence, the total available resource can be represented as a unit cube as shown in the Figure 2. We refer to this cube as *Normalized Resource Cube (NRC)*. All the resource related information is expressed as a vector within the NRC. Thus, The total capacity of PM is expressed as a vector from the origin of the cube (0, 0, 0) to point (1, 1, 1). This vector is identified as Total Capacity Vector (TCV). Resource Utilization Vector (RUV) represents the current utilization of resources of a PM. RUV is the vector addition of normalized utilization vectors of each resource type. Note that the normalization of utilization of individual resource type happens with respect to the total capacity of that resource type. The vector difference between TCV and RUV represents the Remaining Capacity Vector (RCV), which essentially captures how much capacity is left in the PM. The resource requirement of a VM is represented by Resource Requirement Vector (RRV) which is the vector addition of normalized resource requirement vectors of each resource type. Note that the resource requirement is normalized with respect to the total capacity of the target PM. If RUV of a PM exactly aligns with the TCV, then we say that the PM is utilized in a balanced manner along each resource axis. So, to measure the degree of imbalance of resource utilization of a PM, we define the Resource Imbalance Vector (RIV) of PM, which is the vector difference between RUV's projection on TCV and RUV. RIV of a VM is defined in a similar way: it is the vector difference between RRV's projection on TCV and the RRV.

C. SandPiper

SandPiper is a XEN based automated provisioning system for monitoring and detecting hotspots [13]. Thus, it detects when a VM is underprovisioned and either allots more resources locally (if possible) or migrates the VM to a new PM which is capable of supporting the VM. In this discussion, we will focus mainly on how this migration is done. More specifically, how Sandpiper decides the destination PM.

Sandpiper takes VM migration decision based on a metric, which it refers to as *volume*. As we describe later, this is not actually volume. So to avoid confusion, we term it as *sand_volume*. Sandpiper defines this as

$$sand\_volume = \frac{1}{1-cpu} * \frac{1}{1-net} * \frac{1}{1-mem}$$

where *cpu*, *net* and *mem* are the corresponding normalized utilizations of the resources. If we consider PM as a 3-D object with the above three resources as its three dimensions, then the total capacity of the PM corresponds to a unit cube. Thus, the total normalized utilization (of all the three resources) corresponds to *occupied volume*, i.e., this much of volume (or combined resources) has been used up. We refer to the volume which is available as *exploitable volume*. Thus, exploitable volume is given by

$$exploitable\_volume = (1-cpu) * (1-net) * (1-mem)$$

which is the remaining capacity volume of the PM. Therefore, the metric used by Sandpiper is actually not the total utilization volume of the PM, but it is actually the inverse of the exploitable volume of the PM. We will use the term *total utilization volume* and *volume* interchangeably in this paper and we define it as $(cpu * net * mem)$.

When a hotspot is detected the PMs are ordered in decreasing order of their *sand_volumes*. Within each server, VMs are considered in decreasing order of their *sand_volume to size ratio (VSR)*, where size is the memory footprint of the VM. The VM migration algorithm then proceeds by considering the VM with highest VSR from the PM having highest *sand_volume* and determines whether it can be migrated to the *target PM* with least *sand_volume*. The move is only possible if each of individual resource requirement of VM can be fulfilled by the new PM. If sufficient resources are not available on that PM then the PM with second least *sand_volume* is considered and so on. If none of the PMs is able to satisfy the resource requirement of VM then the VM with next highest VSR is considered.

The above approach of migrating VM is a worst fit algorithm with a greedy approach. Intuitively, this algorithm might seem fine. But if one thinks little bit more about the approach it will be clear that this approach may not be as appropriate. We illustrate the flaw in this approach by a simple example. For simplicity, without loss of generality, we reduce the problem to two dimensions (two resources). Consider two PMs with total utilization volume (area) as shown in Figure 3A and 3C. The *sand_volume* of the two PMs are

$$\textit{sand\_volume(PM1)} = \frac{1}{0.33} * \frac{1}{0.67} = 4.52$$
$$\textit{sand\_volume(PM2)} = \frac{1}{0.43} * \frac{1}{0.43} = 5.40$$

Let us say the VM to be migrated (from the highest sand_volume PM) has dimensions as $cpu = 0.3$ and $mem = 0.2$. As per Sandpiper algorithm, PM1 will be picked as target PM, since its sand_volume is lower and it satisfies the individual dimensional requirements of the VM. Once this VM is migrated to PM1, the resource utilization would be as shown in Figure 3B. From this figure, it is clear that the space left on PM1 (after placing the VM) is very small. On the other hand if the VM was placed on PM2 (Figure 3D), then the space left is more than that when it is placed on PM1. Moreover, the relative resource utilization among individual resources in PM2 is more balanced than PM1. It is obvious, that the VM placement should happen based on exploitable volume of the PMs and that the shape of the exploitable volume should also be taken into account while deciding the target PM to migrate a VM to.

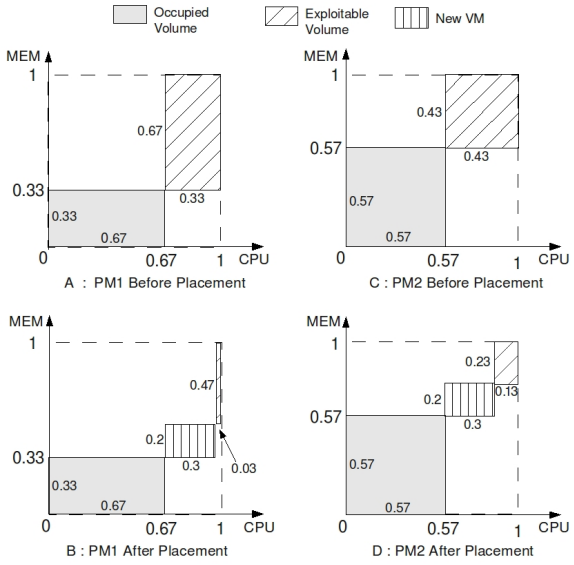

Fig. 3. Illustration of Anomaly in SandPiper VM Placement

The root cause of Sandpiper choosing the wrong PM is that it converts the three dimensional resource information of PMs into a single dimension metric (which is sand_volume) and then uses this single dimension metric for worst fit in a three dimensional scenario. In this process, the information about the shape of the resource utilization is lost. If the resource utilization was actually single dimensional, then comparing target PMs with respect to the dimension would be correct. In another words, if two PMs have same exploitable volume (in single dimension case), then they both are equally suitable as target PM. But when resources are in multiple dimensions, even if two PMs have same exploitable volume, one may be better than the other based on the *shape* of its exploitable volume. So, a good VM placement algorithm should not only consider exploitable volume of resources, it also should take the shape of the exploitable volume into account.

D. Vector Dot

Another novel scheme for VM placement has been reported in [10], which the authors refer to as *VectorDot* method. In this method, normalized resource utilization of PMs and normalized resource requirement of the VMs are expressed as vectors. Basically, VectorDot uses *Dot Product* of RUV of PM and RRV of VM to choose the target PM for placement of VM. The PM, whose RUV gives the lowest *Dot Product* with RRV of VM, is chosen. The main idea behind this method is to place VM on a target PM for which the resource requirement vector of the VM is *complementary* to the resource utilization vector of the PM. This ensures relatively balanced use of different resources. For example a VM which has small CPU requirement and high MEM requirement should be put on a PM which has large CPU utilization but small MEM utilization. In terms of RUV and RRV, it means that for a given VM (to be placed), among all the target PMs, the PM, whose RUV makes largest angle with the RRV of the VM, is the most appropriate target PM. VectorDot uses dot product of RUV and RRV to find the most appropriate target PM. VectorDot uses a metric they termed as *extended dot product*,

which makes use of dot product of RUV and RRV to find the most appropriate target PM.

The PM selection method used by VectorDot may seem all right. But a closer look at it would reveal that the method can lead to undesirable situation, which we explain by giving a simple numerical example. Here again, the example is in a two dimensional space, to keep it simple. The example is depicted in Figure 4. The figure shows the normalized total capacity vector (TCV) for two PMs, RUVs of the two PMs and RRV of a VM. Of the two PMs one has to be chosen based on VectorDot method for placement of the VM. The RUVs of the two PMs and RRV of VM are represented by vectors

$$\overrightarrow{RUV}(PM1) = 0.5\hat{i} + 0.4\hat{j}$$
$$\overrightarrow{RUV}(PM2) = 0.3\hat{i} + 0.4\hat{j}$$
$$\overrightarrow{RRV}(VM) = 0.1\hat{i} + 0.2\hat{j}$$

where $\hat{i}$ and $\hat{j}$ denote the unit vector along CPU and MEM axis respectively. So, the dot product between $\overrightarrow{RRV}(VM)$ and $\overrightarrow{RUV}(PM1)$ is 0.13, whereas the dot product between $\overrightarrow{RRV}(VM)$ and $\overrightarrow{RUV}(PM2)$ is 0.11. So, VectorDot will choose PM2 as target PM. However, it is clear from Fig 4 that PM1 is a better choice for VM. This is because the VM has complementary resource requirement with respect to PM1, i.e., VM needs more memory than CPU and PM1 has used up less memory and more CPU. In case of PM2, it has used up more memory and less CPU and hence clearly the VM does not have complementary resource requirement with respect to PM2. Since the VectorDot approach takes dot product of RRV of VM and RUV of PM, the length of PM's RUV plays a role in deciding the target PM. This factor is responsible for the flaw in VectorDot approach. Instead, it should have picked the PM, whose RUV subtends larger angle with the VM's RRV. So, an easy way to get this information is to find the dot product between the VM's RRV and the unit vector along PM's RUV. Let us illustrate this using the same example.

The unit vectors of RUVs of PM1 and PM2 are respectively

$$\widehat{RUV}(PM1) = \frac{0.5\hat{i} + 0.4\hat{j}}{\sqrt{0.5^2 + 0.4^2}} = 0.78\hat{i} + 0.62\hat{j}$$
$$\widehat{RUV}(PM2) = \frac{0.3\hat{i} + 0.4\hat{j}}{\sqrt{0.3^2 + 0.4^2}} = 0.6\hat{i} + 0.8\hat{j}$$

Now the dot products will be

$$\overrightarrow{RRV}(VM) \cdot \widehat{RUV}(PM1) = 0.1 * 0.78 + 0.2 * 0.62 = 0.202$$

$$\overrightarrow{RRV}(VM) \cdot \widehat{RUV}(PM2) = 0.1 * 0.6 + 0.2 * 0.8 = 0.22$$

And the PM with smaller dot product, i.e., PM1 will be chosen as the target PM. However, the unit vector approach does not solve the problem completely as we discuss below.

There is one more subtle problem with the approach used in VectorDot. The authors claim that they used a best fit scheme which considers placing the VM on that PM which gives the lowest dot product (between RRV and RUV). But the method used by VectorDot may not always be a best fit scheme. For example, consider the example shown in Figure 5. The Figure shows two PMs which have different RUVs. But the magnitude of the two vectors are same. Since the angle between VM's RRV and PM2's RUV is more than that between VM's RRV and PM1's RUV, VectorDot would choose PM2 as target PM (because the earlier one will have lesser dot product). But

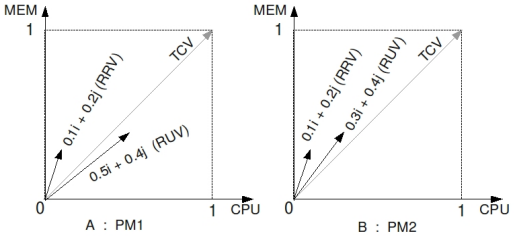

Fig. 4. Illustration of VectorDot Method

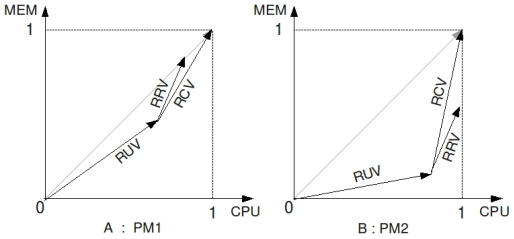

Fig. 5. Mitigating Problem in VectorDot by Using RCV

after placing VM on PM2 it can be seen that it turns out to be a best fit only in terms of CPU. The Memory is still quite underutilized. If the VM is placed on PM1 then it could have resulted in a better utilization of Memory resource. It can be verified that the unit vector method mentioned above would also have the same problem as VectorDot, i.e., it will choose PM2 as the target PM. This flaw is introduced because VectorDot mechanism did not consider RCV while attempting to achieve best fit. The flaw can be corrected by comparing the angle between between VM's RRV and RCVs of the PMs and choosing the PM whose RCV subtends the smallest angle with VM's RRV. The intuitive reason behind the above remedial method is that the closer the VM's RRV to the RCV of a PM, the more balanced its final RUV (after the VM is placed on the PM under consideration) would be.

E. Virtualized Server Load

The work reported in [4] uses a metric called *Virtualized Server Load (VSL)* for load balancing and placement. This metric is defined as follows.

$$VSL_{host} = \sum_{resource} w_{resource} \times \frac{\sum_{v \in VM_{host}} v_{resourceUsage}}{Host_{resourceCapacity}}$$

where *resource*$\in \{CPU, Memory, Disk\}$ and $w_{resource}$ is the weight associated with each *resource*, VM_{host} is the set of VMs hosted on *Host*, $v_{resourceUsage}$ gives the utilization of *resource* by VM and $Host_{resourceCapacity}$ is the total capacity of *Host* in terms of *resource*. This metric is used to represent the utilization of a server and is primarily employed for load balancing. The VSL value is found for all the servers in the infrastructure and the mean and standard deviation is calculated. The mean of the VSL values of PMs in a set L is denoted by by μ_L and standard deviation on the same load set is denoted by σ_L. The ratio of the two (shown below) is taken as an indicator of the load balance factor of the set of L PMs.

$$C_L = \frac{\mu_L}{\sigma_L}$$

In the event of overload or load imbalance the VMs which cause C_L to decrease are marked for migration. It can be seen that this approach tries to make VSL values of all the PMs as close to each other as possible. It can be seen that

VSL essentially uses normalized utilization of resource of a VM (normalized with respect to PM's utilization) for load balancing. Hence it suffers from the similar problems as that exhibited by *SandPiper*.

IV. Our Methodology

In the previous section, we saw shortcomings of some of the existing methods used in VM placement. Before we present our novel methodology, we would like to outline the properties that metrics and parameters used in VM placement should have, so that the anomalies pointed out in the previous sections can be avoided.

A. Properties Required for Metrics Used in VM placement

1) *Shape*: The metric should retain the shape of the *abstract* multi-dimensional object representing VM or PM.
2) *Remaining Capacity*: The metric should carry the information about remaining capacity or exploitable capacity (of all resource types) which will be used while placing the objects (VMs).
3) *Remaining Capacity of Individual Dimension*: The metric should also retain the remaining capacity of each individual resource type.
4) *Combined Utilization*: Combined utilization (of all resource types) information of PM should be carried by the metric.
5) *Utilization of Individual Dimension*: Information about utilization of each resource type (dimension) should also be available.

A metric having all the properties mentioned above can be used for VM placement, server consolidation and load balancing. But it is very difficult for a single metric to have all the above properties. Hence, it is better to have different metrics to address different problems. Since we use a vector based method, we carry all the information required to come up with different metrics to address above said problems.

B. Operations Required for Resource Management in Data Centers

The broad operation of *VM Placement* can be categorized into following more specific operations:

- Static VM placement : In this problem, a set of VMs with their respective resource requirement is given. A set of PMs with their individual capacity is given. The objective is to place the VMs in the PMs such that the total number of PMs required is minimized.
- Dynamic VM placement: Here, a set of PMs with their current resource utilization is given. As and when new VMs arrive, the goal is to place a new VM in a PM such that a given objective is satisfied (e.g., such that PMs are load balanced)
- PM Load balancing : This operation involves identifying overloaded PMs and then migrating some VMs (in those PMs) to other PMs in the system so that the overloading of PMs is mitigated.
- PM Consolidation: The idea here is to identify PMs running at low utilization and then migrating VMs resident on those PMs to other PMs such that those low utilization PMs can be taken off-line.

None of the existing approaches we discussed in the previous section provide methodologies to support all of the above mentioned operations related to VM placement. In the subsequent sections, we would present the theory behind our methodology. Then we will present algorithms to perform each of these operations related to VM placement.

C. Representation of Resources

We represent all the quantities related to various resources as vectors. All these vectors has been defined and explained in Section III-B. Note that if an administrator does not want to allow all the capacity of the PM to be utilized, then allowable fraction of the capacity (of each resource type) will define the NRC. Since we represent all the quantities of resource as vectors the shape of the resources are preserved while making various decisions related to VM placement.

D. Planar Resource Hexagon

We have resource vectors TCV, RUV, RCV and RRV in the 3-D space which we will use for making different VM placement decisions. One of our prime goals while placing VM is to make the resource utilization of PMs as balanced (along each resource dimension) as possible, i.e., the RUV of a PM should be as closely aligned to the TCV as possible. This would require that we have a way of finding *complementary VM* for a PM. For example, a VM which has more requirement along MEM compared to CPU (imbalanced in MEM), should be placed on PM which is less utilized in MEM compared to CPU. To achieve this goal of finding a good match for a given VM, we have devised a novel approach which we present next.

We begin with the NRC and project it on a plane perpendicular to the principal diagonal of the cube as shown in Figure 6. It is easy to see that this would result in a regular hexagon on the said projection plane. We refer to this as Planar Resource Hexagon (PRH). Corners (0, 0, 0) and (1, 1, 1) of the NRC map to the center of the PRH. Other six corners maps to the six vertices of the hexagon as marked in the figure. Then we take the projection of tip of the resource vectors TCV, RUV (of PM) and RRV (of VM) onto the projection plane. The projection points of these three vectors would be inside the regular hexagon. For simplicity, the projection point of RUV of PM (RRV of VM) will be referred to as position of PM (VM) in the PRH. Note that the projection of tip of TCV is the center of the PRH.

The PRH is divided into six equal parts in the form six equilateral triangles. All the six triangles are congruent to each other. These six triangles make it very convenient to classify resource vectors RUV and RRV in terms of their individual resource dimension. We have named these six triangles as per the position of projection of tip of the vector. For example in Figure 7, $\triangle CM$ defines the region where the projection of tip of RUV (of a PM) whose $CPU > MEM > IO$, fall in the PRH. Likewise, $\triangle CI$ represents the region where the projection of tip of RUV (of a PM) whose $CPU > IO > MEM$ would fall on the PRH. Other triangles can be identified by similar inequalities. When there is a tie between two or more resources then it should be broken using some order of priority among the resources as per user's preference. We will refer to these triangles as *resource triangles (RT)*. These triangles can be exploited to figure out relative imbalance of a resource vector along each resource axis and to identify a complementary resource vector which can balance out the individual resources. This is very helpful while choosing

suitable target PM for a VM. For this purpose of choosing target PM of a given VM, we define *complementary resource triangle (CRT)* of a RT. For resource triangle CI, CPU is the most utilized resource, MEM is least utilized and IO is in between the other two. For triangle MI, MEM is most utilized, CPU is least and IO is in between the two. Thus, $\triangle CI$ and $\triangle MI$ are complementary of each other in terms of resource usage. It is easy to verify that the opposite triangles in the PRH are complementary of each other. We index these six triangles from 0 to 6 as shown in the Figure 7.

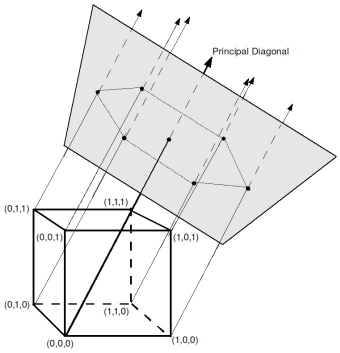

Fig. 6. Obtaining Planar Resource Hexagon from NRC

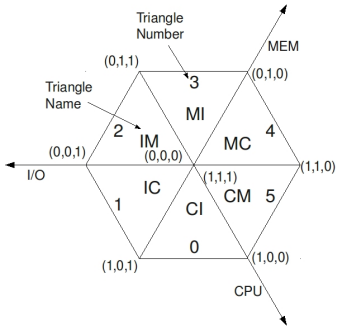

Fig. 7. The Planar Resource Hexagon (PRH)

E. Categorization of Overall Utilization

The Planar Resource Hexagon captures the directionality of a resource vector. The point representing RUV of a PM or RRV of a VM in the PRH carries information about the relative utilization of different resources. But it does not have the information about the overall utilization. This information is important in classifying utilization level of a PM. This classification, in turn, is required while making load balancing and consolidation decisions. We categorize overall utilization of a PM as LOW, $MEDIUM$ or $HIGH$. We define a PM to have overall $HIGH$ utilization if any of its individual resources has high utilization. Essentially, these three levels of utilization define three cubes in the NRC cube as shown in Figure 8. A simple algorithm to find the overall utilization of a PM is given in Algorithm 1. Note that even though the algorithm uses a max function, it is actually not required to look at individual resources to compute this function. Knowing the triangle where a PM lies, the individual resource which has the maximum utilization can be determined instantly. For example, if this point lies in triangle CM then CPU has the highest utilization. The threshold for deciding the utilization level can be left to the user.

Algorithm 1 GetOverallPMUtilization()

```
1: if max(CPU, MEM, IO) ≥ HIGH then
2:     Utilization level is HIGH;
3: else if max(CPU, MEM, IO) ≥ MEDIUM then
4:     Utilization level is MEDIUM;
5: else
6:     Utilization level is LOW;
7: end if
```

It may not be clear why we are suggesting to look at the individual resource that has maximum utilization to decide the overall utilization of a PM, instead of just looking at the magnitude of PM's RUV. The magnitude of PM's RUV is not a good representation of the overall utilization. For example, consider two PMs, PM1 and PM2 whose RUVs are of same length. Let us say PM1 is more tilted towards the CPU axis whereas PM2 is tilted towards the TCV. In such a scenario, PM1 would have lesser amount of CPU to offer than PM2. Thus even if other resources are less utilized for PM1 (than PM2), the CPU becomes the bottleneck for PM1. Hence, utilization of CPU should be used to decide overall utilization of PM1.

It should be clear that we are using two levels of categorization for a PM. First we try to find the triangle in the PRH in which a PM lies and then we find its overall utilization level. Thus, there are a total of eighteen (six triangles and three utilization levels) categories to which a PM may belong.

F. Ordering VMs or PMs Inside a Resource Triangle

Inside a given resource triangle, we order the resource vectors as per its distance from the principal diagonal of NRC. This distance is essentially a measure of imbalance of the resource vector with respect to the TCV. We represent this imbalance factor as vector RIV. RIV preserves directionality (imbalance towards which resource axis or axes) as well as the amount of imbalance. It is quite easy to compute this vector for a given resource vector. We outline the major steps below. Assume that a resource vector (say RUV) is represented by $c\hat{i} + m\hat{j} + io\hat{k}$, where c, m and io are the normalized value of CPU, MEM and IO used by the resource vector respectively. $\hat{i}$, $\hat{j}$ and $\hat{k}$ are the unit vectors along CPU, MEM and IO axes respectively. The projection vector of this vector along the principal diagonal of NRC is given by

$$(\frac{1}{\sqrt{3}} * c + \frac{1}{\sqrt{3}} * m + \frac{1}{\sqrt{3}} * io)(\frac{1}{\sqrt{3}}\hat{i} + \frac{1}{\sqrt{3}}\hat{j} + \frac{1}{\sqrt{3}}\hat{k})$$

The term in the right side parenthesis is the unit vector along TCV and the term in the left side parenthesis is the magnitude of projection of RUV on TCV. The above expression is simplified to

$$(\frac{c+m+io}{3}\hat{i} + \frac{c+m+io}{3}\hat{j} + \frac{c+m+io}{3}\hat{k})$$

So the RIV is given by the vector difference between the RUV itself and the projection vector shown above, which is

$$(c - \frac{c+m+io}{3})\hat{i} + (m - \frac{c+m+io}{3})\hat{j} + (io - \frac{c+m+io}{3})\hat{k}$$

Having introduced all the concepts required for our methodology now we will show how the three functionalities, namely VM placement, load balancing and server consolidation, required to manage resources of PMs can be designed.

G. Static VM Placement

In this operation, a set of VMs with their respective resource requirements is given. A set of PMs, on which the those VMs are to be placed, is given. The goal is to place the VMs in PMs such that the total number of PMs required to place all the VMs is minimized. Since this is a NP-hard problem, we provide a heuristic. Here, we assume that this static placement happens when the system is coming up. However, if a static placement is to happen at any other time, our algorithm can be easily modified for that situation. The algorithm is given in Algorithm 2. Essentially, the algorithm starts out with the VM which is least imbalanced. This VM would be put on the first

PM. The new RUV of the PM is computed and its new position (new resource triangle) in the PRH is found (Line 22). Then VMs in the complementary resource triangle is considered (Line 23) and the VM which best complements the PM is chosen. This is determined by the addition of their RIVs and choosing the one with the smallest value (Line 21). So, the algorithm tries to place the VMs such a way that after the placement the PMs are as close to the center of the PRH as possible.

Algorithm 2 StaticVMPlacement

```
1: T ← triangle in which the VM with least (in magnitude) RIV lies;
2: Introduce a new PM;
3: while All VMs are not placed do
4:     NumPotentialVMs ← 0; NumVMs ← 0;
5:     for all unplaced VMs in T do
6:         NumVMs ← NumVMs + 1;
7:         /*check individual resources to see if the PM can host this VM*/
8:         if PM can host this VM then
9:             NumPotentialVMs ← NumPotentialVMs + 1;
10:            Compute the vector addition of RIV of PM and RIV of VM;
11:            M ← magnitude of this addition vector;
12:        end if
13:    end for
14:    if NumVMs == 0 then
15:        /*Choose the neighbor of triangle T(configurable left of right)*/
16:        T ← neighbor(T); continue;
17:    end if
18:    if (NumVMs > 0)AND(NumPotentialVMs == 0) then
19:        Introduce a new PM; continue;
20:    end if
21:    Place the VM with least M on the current PM and mark the VM as
       placed;
22:    Compute the new RUV of this PM (after the VM is placed);
23:    T ← CRT of the resource triangle to which this PM belongs;
24: end while
```

H. Dynamic VM Placement

In case of dynamic VM placement there is one new VM which needs to be placed in a PM. The target PM for this new VM can be chosen based on one of the two goals:

- Load Balancing : Placing the new VM in such a manner that it helps in load balancing. For this purpose, the algorithm starts out with the PM which is least loaded and has complementary resource usage with respect to the VM.
- Server Consolidation : Placing the new VM such that it helps in server consolidation. In this case, the algorithm starts out with the PM with high load and has complementary resource usage with respect to the VM.

The algorithm for dynamic VM placement is shown in Algorithm 4. The *goal* value in this algorithm can be $LOAD\_BALANCE$ or $CONSOLIDATE$. This algorithm starts out with getting potential PMs for the VM by calling Algorithm 3. Then it just chooses the PM which is most complementary to the VM. Note that the best complementary PM is one whose RIV is of same magnitude as the VM's RIV and is in the opposite direction. The main subroutine called in this algorithm is *GetPotentialPMs()*. This algorithm (Algorithm 3) uses a heuristics to identify potential PMs for the VM. To understand this heuristics please refer to Figure 9. The figure shows the PRH and shows the triangle where a potential PM is being considered (say triangle 0). The first level neighbor of triangle 0 (where the PM belongs) are triangles 1 and 5. The second level neighbors are triangles 2 and 4. The traversal of the algorithm is along the pyramid shown to the right of the PRH in the figure. For load balancing,

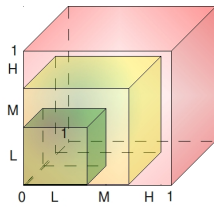

Fig. 8. Resource Utilization Levels in Form of Sub-cubes

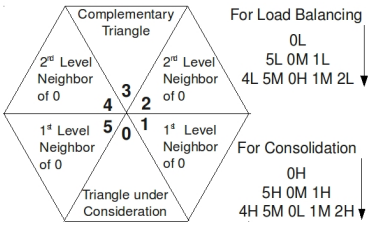

Fig. 9. Finding Potential PMs

in this example, it would first pick all the PMs in triangle 0 which have LOW overall utilization (marked as $0L$ in the figure) (Line 6 in Algorithm 3). If no PM was found, then it would pick PMs in triangle 0 with $MEDIUM$ utilization and PMs in triangles 5 and 1 with LOW utilization (marked as $5L$ $0M$ $1L$ in the figure) (Line 9). If still no PM was found, then it would pick PMs in triangle 0 with $HIGH$ utilization, PMs in triangle 5 and 1 with $MEDIUM$ utilization and PMs in triangle 4 and 2 with LOW utilization (marked as $4L$ $5M$ $0H$ $1M$ $2L$) (Line 13). This order of traversal of resource triangles tries to have a compromise between overall utilization of the PMs and the imbalance in RUV.

Algorithm 3 GetPotentialPMs(VM, goal)

```
1: /* goal can be either LOAD_BALANCE or CONSOLIDATE */
2: T ← CRT of the triangle in which the VM lies;
3: PotentialPMlist← ∅;
4: if goal = LOAD_BALANCE then
5:     {/* start from LOW util to HIGH util as shown in Figure 9*/}
6:     PotentialPMlist ← {PMs in T which have LOW utilization};
7:     Remove PMs from PotentialPMlist which can not support the VM;
8:     if PotentialPMlist= ∅ then
9:         PotentialPMlist←{PMs in T which have MEDIUM
           utilization}∪{PMs in T's two first order neighbors which
           have LOW utilization};
10:        Remove PMs from PotentialPMlist which can not support the VM;
11:    end if
12:    if PotentialPMlist= ∅ then
13:        PotentialPMlist←{PMs in T which have HIGH
           utilization}∪{PMs in T's two first order neighbors which
           have MEDIUM utilization}∪{PMs in T's two second order
           neighbors which have LOW utilization};
14:        Remove PMs from PotentialPMlist which can not support the VM;
15:    end if
16: else if goal = CONSOLIDATE then
17:    {start from HIGH util to LOW}
18:    PotentialPMlist←{PMs in T which have HIGH utilization};
19:    Remove PMs from PotentialPMlist which can not support the VM;
20:    if PotentialPMlist= ∅ then
21:        PotentialPMlist←{PMs in T which have MEDIUM
           utilization}∪{PMs in T's first order neighbors which have
           HIGH utilization};
22:        Remove PMs from PotentialPMlist which can not support the VM;
23:    end if
24:    if PotentialPMlist= ∅ then
25:        PotentialPMlist←{PMs in T which have LOW utilization}∪{PMs
           in T's first order neighbors which have MEDIUM
           utilization}∪{PMs in T's second order neighbors which
           have HIGH utilization};
26:        Remove PMs from PotentialPMlist which can not support the VM;
27:    end if
28: end if
29: if PotentialPMlist is Empty then
30:    put all the PMs which were not checked above into PotentialPMlist;
31:    Remove PMs from PotentialPMlist which can not support the VM;
32: end if
33: return PotentialPMlist;
```

I. Load Balancing

Load Balancing is required when there is a VM whose resource requirements are not being fulfilled by the PM on which it is hosted, thus leading to overload of the PM. The

Algorithm 4 DynamicPlaceVM(VM, goal)

```
1: PotentialPMlist ← GetPotentialPMs(VM, goal);
2: if PotentialPMlist empty then
3:     Introduce new PM;
4: end if
5: for all PM in PotentialPMlist do
6:     Compute the vector addition of RIVs of PM and VM;
7:     M ← magnitude of the above addition vector;
8: end for
9: Mark PM with the lowest M as the host for the VM;
```

resource crunch can happen for any resource. The algorithm is presented in Algorithm 5. The algorithm uses *cost* which is used to arrange VMs in VMlist. This cost can be defined by the user. For example, the cost can be the memory foot print of the VM, since it signifies the cost of migrating the VM. Another cost can be the VMs utilization of the resource which is causing overload on the PM.

J. Consolidation

When a PM runs on low utilization, the VMs on the server can be migrated to other PMs so that this PM can be taken off line. A simple algorithm to achieve this is presented in Algorithm 8.

Algorithm 5 LoadBalance(PM)

```
1: Order all VMs hosted on PM as per some cost into VMlist;
2: for all VM in VMlist do
3:     DynamicPlaceVM(VM, LOAD_BALANCE);
4:     if PM not Overloaded anymore then
5:         BREAK;
6:     end if
7: end for
```

Algorithm 6 LoadBalanceALLPMs()

```
1: for all Triangles T in Planar Resource Hexagon do
2:     for all PM in T with HIGH utilization do
3:         if PM is Overloaded then
4:             LoadBalance(PM);
5:         end if
6:     end for
7: end for
```

Algorithm 7 Consolidate(PM)

```
1: Arrange all VMs hosted on PM in VMlist;
2: for all VM in VMlist do
3:     DynamicPlaceVM(VM, CONSOLIDATE);
4:     if No more VMs on PM then
5:         Remove PM;
6:     end if
7: end for
```

V. Discussion

This paper clearly points out shortcomings of some of the work reported in the existing literature. We feel that the subtle drawbacks and anomalies are not very obvious in those studies. Hence, we think it will help the research community realize the shortcoming and prevent them from going in the wrong direction while dealing with VM placement, server load balancing and server consolidation problem.

We would like the readers to appreciate our methodology in terms of its correctness vis-a-vis the other literature in this field. It is easy to verify that the anomalies shown in the examples given for SandPiper and VectorDot do not exist in our methodology. We acknowledge that some of the algorithms presented here which uses our methodology are heuristics and hence may not be the most efficient. The intention of presenting those algorithms is to depict a way of using our methodology in solving the above three problems rather than showing their efficiency.

Algorithm 8 ConsolidateALLPMs()

```
1: for all Triangles T in Planar Resource Hexagon do
2:     for all PM in T with LOW utilization do
3:         Consolidate(PM);
4:     end for
5: end for
```

VI. Conclusion and Future Work

We have presented some of the methodologies used in existing literature in making VM placement and migration, server consolidation and load balancing decision. We have pointed out the drawbacks and anomalies present in those methodologies and where those anomalies originate from. We have proposed a novel methodology based on vector arithmetic to address these drawbacks and build the theory which can be used not only to make the decisions robust but also to make the process of choosing PMs easier and more appropriate. We feel that our methodology is the right step towards devising more efficient algorithms free of anomalies.

We intend to evaluate the performance of our methodology and compare it with the other methodologies in the literature. We are also going to evaluate the efficiency of various heuristics proposed in this paper (e.g., *GetPotentialPMs*). In the current version, we divide PRH into six triangles to define position of a PM or VM. We would like to increase the granularity of position of PMs or VMs by dividing the PRH into more number of triangles. However, more granularity leads to more balanced servers after VM placement, but comes at the cost of more computation to choose target PM for a VM.

References

[1] Bin packing problem. Wikipedia.

[2] Vmware. www.vmware.com.

[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[4] E. Arzuaga and D.R. Kaeli. Quantifying load imbalance on virtualized enterprise servers. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 235–242. ACM, 2010.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM, 2003.

[6] F. Benevenuto, C. Fernandes, M. Santos, V. Almeida, J. Almeida, G. Janakiraman, and J. Santos. Performance models for virtualized applications. In *Frontiers of High Performance Computing and Networking–ISPA 2006 Workshops*, pages 427–439. Springer, 2006.

[7] J.P. Casazza, M. Greenfield, and K. Shi. Redefining server performance characterization for virtualization benchmarking. *Intel Technology Journal*, 10(3):243–251, 2006.

[8] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 373–381. IEEE, 2006.

[9] A. Kochut and K. Beaty. On strategies for dynamic resource management in virtualized server environments. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on*, pages 193–200. IEEE, 2008.

[10] A. Singh, M. Korupolu, and D. Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12. IEEE Press, 2008.

[11] J.E. Smith and R. Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005.

[12] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Predicting Application Resource Requirements in Virtual Environments. 2008.

[13] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. Networked Systems Design and Implementation*, 2007.