Adaptive Provisioning of Virtual Machines

Submitted in Partial Fulfillment of the requirements
for the degree of
Master of Technology
by

Dhaval Vishwanath Bonde
Roll no: 08305910
Under the guidance of
Prof. Umesh Bellur
and
Prof. Purushottam Kulkarni

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay
2011

Dissertation Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

The dissertation entitled **"Adaptive Provisioning of Virtual Machines"**, submitted by **Dhaval Vishwanath Bonde** (Roll No: **08305910**) is approved for the degree of **Master of Technology** in **Computer Science and Engineering** from **Indian Institute of Technology, Bombay**.

Prof. Umesh Bellur **Dept CSE, IIT Bombay** **Supervisor**	**Prof. Purushottam Kulkarni** **Dept CSE, IIT Bombay** **Supervisor**
Prof. Anirudha Sahoo **Dept CSE, IIT Bombay** **Internal Examiner**	**Dr. Dilys Thomas** **TRDDC** **External Examiner**

Prof. Sonar
SJMSOM, IIT Bombay
Chairperson

Place: IIT Bombay, Mumbai
Date: 28^{th} **June, 2011**

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature

Name Of student

Roll number

Date

Acknowledgements

I would like to thank my advisors Prof. Umesh Bellur and Prof. Purushottam Kulkarni for their constant support and guidance. They meticulously pointed out errors and omissions throughout the project. I would also like to thank Mr. Deepak Deshpande from IBM, for his valuable inputs during the monthly meetings.

Special thanks to Sujesha Sudevalayam for her constant help and support during the project. Her unmatched enthusiasm and willingness to help are worth appreciating. I thank Senthil Nathan for lending me his machines for experimental work. I would also like to thank all Synerg lab members Ramesh, Vaibhao, Prajakta, Kushal, Riju, Vijay, Sumedh and Krishnadhan for fun moments during the hectic hours in the lab. I would like to thank Purva, Manasi and Aditya for their memorable company and immense moral support during the entire journey of MTech.

Finally, I would like to thank the Department of Computer Science and Engineering at IIT Bombay for providing excellent infrastructure for the project work and a student friendly environment.

Above all, I would like to thank my parents Vishwanath and Shubhangi Bonde, my brother Vipul and my grandmother for their immense love and support. Without them, working on this project would have been impossible.

This project was partly funded under CAS(Center for Advanced Studies), grant no. 09IBM001 at IIT Bombay, from IBM.

Contents

List of Tables

List of Figures

Abstract

Virtualization has brought with it many opportunities for data center management, mainly in terms of higher capacity savings and higher operating efficiency. One main benefit of virtualization is multiplexing of many virtual machines on the same physical machine. For accruing this benefit, an important step is provisioning of virtual machines. This can be done by analyzing the resource usage traces of the corresponding application. One of the approaches that has been tried for provisioning is the consideration of a *peak buffer*, which is the characterization of the peaks of a resource utilization timeseries, to be considered for multiplexing with other peak buffers. The resource usage values above the 90th-percentile of the resource usage distribution have been used as peak buffers. However, simply choosing a 90th percentile value may not provide us with the complete benefits of the peak buffer, since it does not consider the way in which the resource utilization is distributed. Hence, in our work, we have proposed a method which adaptively decides this percentile value for the peak buffer, from the resource usage distribution. We present an algorithm for this and evaluate it with respect to a 90th percentile based strategy. Results show that our adaptive provisioning algorithm provides a better packing of VMs with a reasonably lower number of violations of the provisioned capacity.

Chapter 1

Introduction and Problem Definition

In traditional data centers, applications are tied to specific physical servers that are often over-provisioned in order to serve the complex resource requirements of enterprise services and in the hope of handling unexpected surges in resource demands. As a consequence, the level of resource utilization on any server is typically very low [3]. This incurs high operational costs and heavy investments, not to mention wasted power and floor space and a significant management overhead. Such a situation may be referred to as a 'server sprawl' [15], a situation in which there are many underutilized servers taking up more space and energy than can be justified by their workloads.

Cloud computing offers a means to decouple the application activities from the physical servers hosting them. This has enabled consolidation of multiple applications onto a lesser number of physical servers resulting in an increase in server utilization. Such decoupling of resources is facilitated by the concept of a 'virtual machine' which hosts an application. Physical resources are made available to the virtual machine by a host operating system running on each physical machine. The virtual machine runs over this host operating system which also provides facilities for creation, destruction and migration of virtual machines. Xen [12], KVM [9], VMware [17] are some of the products that are widely used for management of such virtual machines.

Due to the ability to host multiple applications (virtual machines) onto same physical servers, new challenges have cropped up. These challenges include figuring out placement plans - mapping of VMs to PMs, balancing load amongst all physical machines and characterizing application workloads for VM sizing (that is, determining the optimum capacity to be provisioned for a VM).

Because of the benefits in terms of cost and efficient resource usage to be accrued due to virtualization, many organizations are considering moving their data centers from the physical environment to a virtual environment. Moving an application running in a physical non-virtualized environment to a virtualized one involves three broad stages: 1) *Application Profiling* in which the application is profiled in its physical environment to get its resource utilization, 2) *VM Provisioning* in which the resource utilizations are used to generate provisioning values for virtual machines possibly after consideration of factors like workload correlations, affinities, etc. and 3) *Virtual Machine Placement* which takes these values and tries to figure out a grouping of the VMs onto the physical servers so that the number of physical machines is minimized. Our work mainly focuses on the provisioning and placement aspect of this process.

While provisioning, a very important constraint that needs to be considered is the correlation between workloads which facilitates provisioning based on off-peak values of workloads. Traditionally, virtual machines are provisioned by taking the peak resource utilization of the application, and using First Fit Decreasing(FFD) or some form of Best Fit algorithm to come up with a placement plan. Chapter 3 discusses such strategies with peak based provisioning which are agnostic to inter-workload correlation, and gives a comparative account of how these strategies fair in relation to one another.

Although provisioning based on peak usage would be a robust strategy, [16] observes that the provisioned sizes of virtual machines could be unnecessarily high and such a provisioning may even cause the application to be provisioned for all of the capacity available. Hence, provisioning at values much lower than the peak may be desirable. However, the trade-off here is that such provisioning at off-peak values would mean potential conflicts between workload values not provisioned for. Consequently, it also observes that correlation between workloads needs to be considered for placement. Based on these observations, it introduces two methods called Correlation Based Placement(CBP) and Peak Clustering based Placement(PCP), which use the 90th percentile value of workloads for provisioning of virtual machines. However, we argue that instead of a 90-percentile value, a value which is more adaptive to the workload distribution would be desirable for provisioning. Based on this, we introduce a new algorithm which analyzes the Cumulative Distribution Function(CDF) of the workload to come up with the provisioned value. The detailed algorithm is explained in chapter 4.

In the next section, we describe our motivation to come up with such an adaptive provisioning strategy, followed by our problem definition.

1.1 Motivation

As we have discussed, the algorithms proposed in [16] use a provisioning based on the 90th percentile of the workload timeseries. However, we argue that an adaptive strategy with respect to the workload distribution would be more suitable for provisioning.

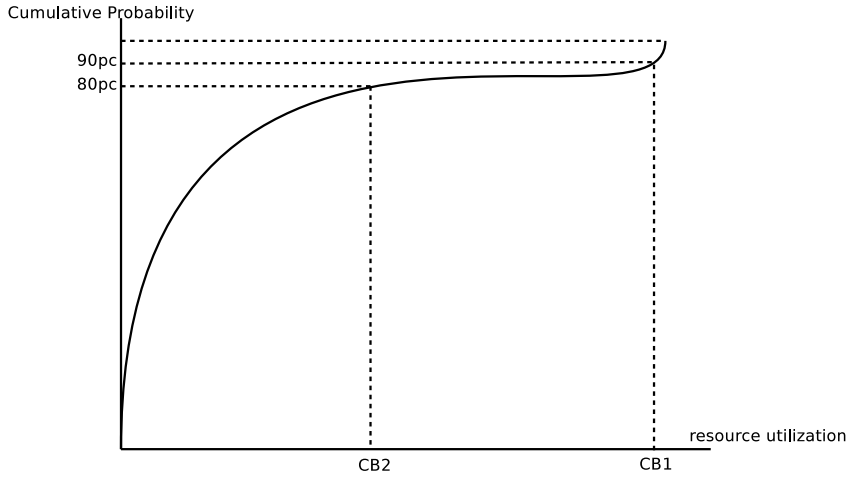

Figure 1.1: Drawbacks of a flat 90-percentile based provisioning

Let us understand this using an example. Consider the CDF in figure 1.1. Assume it represents the CDF of the VM's usage of a particular resource, say CPU, and this resource is being considered for provisioning of the VM. In this CDF, if we provision the size at a 90 percentile value, we will end up with the value of CB1. Whereas if we were to provision at 80 percentile value instead (CB2), we would achieve a significant amount of savings (CB1-CB2), while also ensuring that only 10% (90%ile - 80%ile) more values are being skipped as peak. This will result in huge savings in capacity provisioned while also keeping risk of potential violations by the peak values low.

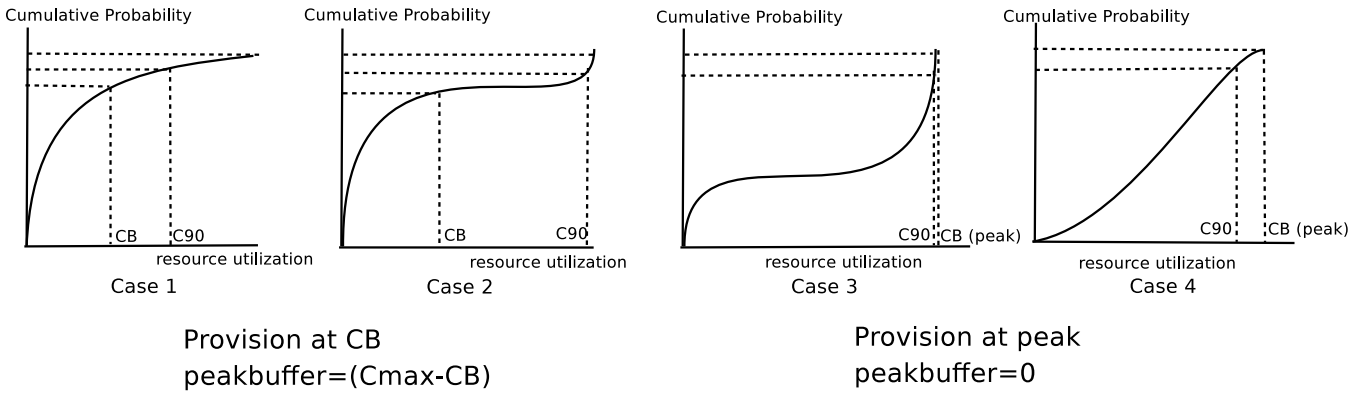

Figure 1.2: Different Cases of a CDF

Now, the typical cases that can occur in a CDF plot are given in figure 1.2. Also, a difference between a 90-percentile based strategy and an adaptive strategy is shown. Assume each of these represent a CPU utilization timeseries. The first two diagrams show the cases where there is a significant difference between peak and non-peak values, and provisioning at a value lower than 90-percentile can lead to a significant potential savings of provisioned resources. The third diagram shows a case where the application peaks too often and it would be inadvisable to provision at any value lower than the peak for the fear of increasing the chances of potential conflicts with these peaks, without gaining much in terms of capacity provisioned. Similarly, the last diagram shows a case where the workload is distributed uniformly with many peaks, and hence should be provisioned at the peak.

Looking at these cases, it becomes clear that there is a need for an adaptive provisioning strategy with respect to the workload distribution, which leads us to our problem statement.

1.2 Problem Definition

Our aim is to come up with a provisioning strategy which is adaptive to the distribution of the workload.
Given a timeseries of resource usage, we wish to:

- Choose an optimum size for the virtual machine such that all values greater than that size, called peaks, find complementary peak usage values in the timeseries of other virtual machines, while also ensuring minimum provisioned size.

- Obtain a placement plan using such a provisioned size, which minimizes the number of physical machines used.

1.3 Contributions

In this report, we first present some approaches using peak based provisioning, which is traditionally used for virtual machine placement. We then give a comparative account of these approaches in relation to each other. Our study shows that these approaches are almost similar with respect to consolidation.

Next, we present an adaptive provisioning algorithm which advocates provisioning based on off-peak values. Such off-peak values are chosen by analyzing the distribution of the workload values. We evaluate this algorithm against a 90-percentile based strategy which is discussed in [16]. We find that our algorithm performs better packing of virtual machines while also ensuring a reasonably low number of violations of the provisioned capacity.

The rest of this report is organized as follows. Chapter 2 gives an overview of the related work. Chapter 3 discusses and gives a comparative account of the traditional peak based provisioning strategies. Chapter 4 gives a detailed description and evaluation of our Adaptive Provisioning algorithm. Finally, chapter 5 presents conclusions and future work.

Chapter 2

Related Work

The process of placement of virtual machines involves two main parts: provisioning of resources for the virtual machines according to the capacity requirements of the corresponding applications, and the actual placement of these virtual machines onto the physical machines, given these provisioned values. During both these phases, one important constraint that needs to be considered is the correlation between the application workloads. Based on this constraint, we can classify the work done in this area as being correlation aware or correlation unaware. Another classification based on the goal of placement is given in [13]. To understand the rationale behind a correlation based classification, we first need to understand the importance of workload correlation.

2.1 Importance of Workload Correlation

[16] observes that (a) there is a risk of SLA violation if consolidation methods are not aware of correlation and (b) there is potential for placing non-correlated applications to mitigate this risk. Traditionally, this risk is curbed by reserving the maximum utilization for each application. However, [16] also observes that such a scheme may cause the application to take up all the capacity on a physical machine. Hence, there is a need to consider correlation between workloads during placement.

Literature has addressed the problem of virtual machine placement both with and without consideration for correlation. Thus, the approaches in literature can be broadly classified as correlation aware and correlation unaware methods. The correlation aware methods consider the fact that workloads of different virtual machines can be complementary to each other, which might help in a tighter packing of virtual machines on physical

machines. Correlation unaware methods do not consider any such correlation.

2.2 Correlation Unaware Methods

These methods do not consider the fact that VM workloads can be complementary to each other. They do, however, address other important research issues in VM placement such as cost of resource allocation, workload forecasting for minimizing SLA violations, etc.

- **Placement for minimizing Cost of Resource Allocation:** [7] and [11] address the issues of cost of allocation of resources while figuring out a placement plan. [7] proposes an optimal virtual machine placement algorithm which tries to minimize the cost spending in the reservation and on-demand payment plans generally offered by cloud providers. The cost here depends on the prices charged by each provider for each resource. It views the provisioning of resources in three phases: reservation, utilization and on-demand. The algorithm uses stochastic integer programming and works in two stages: the first stage calculates the number of VMs provisioned(demand) in the reservation phase and the second stage calculates the number of VMs allocated in both utilization and on-demand phases. It considers the set of all possible demands and prices, called realizations in general. Using these, an objective function and a set of constraints is developed considering the cost of resources charged by each provider in each phase, the probability of each demand being realized, the probability of each set of resource prices offered by each provider being realized. Decision variables are the number of VMs in a particular class to be allocated to a particular provider in a particular phase. Thus, with respect to provisioning, this method provisions based on the number of VMs required to meet the workload demand. Although this approach considers all possible demands and prices, it will not be scalable once the number of possible demands and prices goes high which is bound to happen in a real world scenario. Also, it ultimately formulates the problem as an integer linear program which is proved to be NP-hard [1]. So, this approach will not work with large problem sizes and would need approximations in that case.

  [11] proposes an autonomic resource manager which pre-defines a set of Virtual Machine classes for applications to choose from. They propose a local decision module that measures each application's satisfaction (called a resource level utility

function) with the Virtual Machines allocated to it. Then, a global decision module evaluates these values coming from each application and tries to maximize a global utility function which also considers the cost of making the allocations. They do not, however, propose any specific cost function. These VM to application allocation vectors are then fed to another phase which uses a Constraint Satisfaction Problem (CSP) formulation to figure out a placement of these VMs onto Physical Machines so as to minimize the number of active PMs. Similar to [7], this method too provisions based on number of VMs required to meet the demand. Again, here the placement variables are modelled as binary which renders it unusable for large problem sizes. Another main drawback of their approach is that the VM sizes need to be discretized into specific VM classes. For a near accurate calculations, there should be many VM classes defined. But that would increase the number of variables in the constraints and hence increase complexity of their solution.

- **Placement for minimizing the number of active PMs:** [5] aims to minimize the number of active Physical Machines while coming up with a placement plan. It uses an Integer Linear Programming formulation to get a VM placement plan. They have proposed decision models for a static server allocation problem with and without variable workloads and also a dynamic server allocation problem which calculates allocation plans in individual time steps. All the decision models built by them have variables modelled as binary which again, would suffer from scalability issues.

- **Placement for balancing load across PMs:** [14] proposes an algorithm called VectorDot which tries to determine the best choice of physical machines for relocation of virtual machines from overloaded nodes using dot products of capacity usage and resource requirement vectors. The dot product based fit implemented by us uses a similar technique but is tuned to do server consolidation instead of load balancing.

- **Placement for reduction in the level of SLA violations:** [6] tries to provide a placement plan with minimum level of service level agreement violations. Their algorithm is based on measuring historical data, forecasting the future demand and then remapping the Virtual Machines to the Physical Machines given this information. Resource demands are predicted at regular intervals using resource demand data. These predicted values are used by a placement module to compute VM to PM mappings. This module uses first fit approximation.

Table 2.1: Comparison of Correlation Unaware Methods

	OVMP [7]	**Autonomic Res Manager** [11]	**ILP** [5]	**VectorDot** [14]	**SLA** [6]
Placement Criteria	↓ Cost	↓ Cost, No of PMs	↓ No of PMs	↓ Load	↓ SLA violations
Provisioning value calculated to meet the demand	No of VMs	No of VMs	Capacity Provisioned per VM	Capacity Provisioned per VM	Capacity Provisioned per VM
Is the solution scalable?	No	No	No	Yes	Yes
Underlying Placement Heuristic	Stochastic Integer Programming (SIP)	Constraint Satisfaction Problem (CSP)	Integer Linear Program(ILP)	Their own Extended Vector Product(EVP) in combination with FFD, BFD and variants	First Fit Decreasing(FFD)

All of the above methods are do not consider the correlation between workloads while provisioning for virtual machines, and assume the requirements of individual VMs to be the peak usage of the corresponding workload. In contrast to this, our work is aware of such correlation. Some correlation aware methods are discussed in the next section.

2.3 Correlation Aware Methods

These methods do consider the fact that correlation between workloads needs to be considered while coming up with a placement strategy.

2.3.1 Using a Forecast Error Model

[10] presents a method which forecasts the future workload and uses the corresponding forecast error model to determine the capacity which will restrict the number of violations below a threshold.

They first identify compatible pairs of VMs which have negatively correlated workloads, thus partitioning the VMs into multiple sets. Then, for each set, they calculate its aggregate capacity need using an SLA model, which considers their specified individual SLA

requirements. For such calculation of aggregate capacity, their method takes the historic workload timeseries as input and infers future workload patterns using forecasting methods. They use either a standard or an empirical density function of the forecast error model alongwith the forecasted values to infer the aggregate capacity required according to the parameters to the SLA model. However, they only identify pairs of compatible VMs for creating the multiple sets, which should be extended to consider more VMs.

2.3.2 Using Effective sizing of VMs

[8] comes up with an 'effective size' of a VM which considers the gain due to statistical multiplexing of many VMs on physical servers. It estimates the contribution of a VM's resource requirement to the aggregate resource demand of a server through two parts: an 'intrinsic demand' which is a function of the VM's own demand and the server's capacity; and a 'correlation aware' demand which is a function of the VM's demand and its correlation with resource demand of other VMs on the server. Basically, the intrinsic demand ignores the correlation between VM workloads and the correlation aware demand compensates the error due to such ignorance, by looking at the statistical properties and correlation between VM workloads. Further, [8] also mentions three placement algorithms for static, semi-static and dynamic consolidation (as mentioned in [16]). These algorithms are called 'VM placement - Independent Workload', 'VM placement - Correlation Aware' and 'VM placement - History and Correlation Aware' respectively, and are in that order of increasing complexity. These algorithms are variants of the traditional First Fit Decreasing (FFD) and Best Fit Decreasing(BFD) algorithms for bin packing. However, they have only discussed CPU workloads for their provisioning and placement algorithms. This should be extended to consider other resources too, like disk and network bandwidth.

2.3.3 Using a Peak Buffer

[16] has made some important observations about data center workloads, and based on these, introduced two methods for VM placement: Correlation Based Placement(CBP) and Peak Clustering based Placement(PCP).

CBP sizes the applications based on a 90 percentile value. Then, it adds co-location constraints between any two positively correlated pair of applications. Finally, during placement, if any of the already placed applications on a PM has a co-location constraint with the application to be placed, that server is marked ineligible for placement of the

VM and the next server is considered.

PCP considers correlation between the peaks of applications and groups applications with correlated peaks (peak buffers) into clusters, that is, the algorithm groups together applications which have concurrent peaks. It then selects a set of applications from each cluster in a proportional manner - number of applications selected from a cluster are proportional to the size of the cluster divided by the addition of the total size of all clusters and the maximum of all peak buffers. Also, some capacity on the physical machine is reserved which is equal to the maximum of peak buffers across all clusters.

Although the idea of using peak buffers to represent workload peaks seems quite promising, [16] has used a flat 90 percentile value of the workload to consider as a cut-off for the application size and the upper 10 percentile forms the peak buffer. However, we argue that such a flat 90-percentile based provisioning may not be the right quantum to provision for the application size. With respect to this, we propose a new algorithm which is based on the analysis of the Cumulative Density Function of the workload timeseries to determine the right size of the peak buffer. The details of our algorithm are given in Chapter 4.

In the next chapter, we present some correlation unaware approaches using peak based provisioning, which are traditionally used for Virtual Machine provisioning and placement. Also, we give a comparative account of these approaches in relation to each other.

Chapter 3

Evaluation of Peak Based VM placement heuristics

Traditionally, the peak resource requirements of virtual machine workloads have been considered for provisioning resource capacities. Further, to come up with a placement plan, heuristics like First Fit, Best Fit, etc. and strategies like Linear Programming have been used. In this chapter, we evaluate these traditional approaches. We first discuss the inputs and outputs for these approaches.

Inputs:
The main inputs are the resource requirements of each virtual machine to be placed. These requirements can be captured in a requirements matrix as follows:

$$Requirements\ Matrix\ = \begin{bmatrix} r_{11} & r_{12} & ... & r_{1d} \\ r_{21} & r_{22} & ... & r_{2d} \\ : & : & : & : \\ r_{n1} & r_{n2} & ... & r_{nd} \end{bmatrix}$$

where each r_{ij} indicates the requirement of VM i along the dimension j.

Requirements in this matrix are expressed as fractions of the total capacity of a physical machine (See appendix A for an example).

Outputs:

Given the resource requirements of virtual machines, a placement plan is desired as output, which gives a mapping of which virtual machine is to be placed on which physical machine. This placement plan can be captured in the form of an 'Allocation Matrix' as follows:

$$Allocation\ Matrix\ = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1m} \\ x_{21} & x_{22} & ... & x_{2m} \\ : & : & : & : \\ x_{n1} & x_{n2} & ... & x_{nm} \end{bmatrix}$$

where each x_{ij} indicates whether VM i is placed on the physical machine j. So, x_{ij} will be 1 if VM i is placed on PM j, 0 otherwise. To solve our problem, we essentially have to calculate this allocation matrix.

Our implementations read the input requirements matrix in a specific format (given in appendix B), and calculate the allocation matrix as output. So as to have an ordered list of Virtual Machines, the requirements matrix is pre-ordered while reading from the configuration file.

In the next section, we discuss an LP formulation of the VM placement problem and related issues.

3.1 LP formulation for VM placement

The VM placement problem has been viewed as a form of the bin packing problem, with the virtual machines considered as objects with known dimensions to be fit into a minimum number of physical machines analogous to bins. Hence, a linear programming formulation, which is usually the first choice for solving a bin packing problem, can help solve the problem of VM placement [5]. In this section, we implement an Integer Linear Program and discuss the related issues.

LP Formulation:
We formulate the VM placement problem as an Integer Linear Program with the following constraints:

- **Capacity Constraints:** For each dimension of a given PM j, the sum of the

resource requirements of all VMs placed on it should be less than or equal to the total available capacity:

$$\sum_i r_{ik}.x_{ij} \leq 1 \qquad \forall k \; \forall j \tag{3.1}$$

where, r_{ik} is the fractional resource requirement of an individual VM i along the dimension k. For each PM, this inequality is created for each dimension.

- **Placement Guarantee Constraints:** All virtual machines should be placed. So, for each VM i, exactly one of the x_{ij}'s is expected to be one, and others 0. So we get the equation:

$$\sum_j xij = 1 \qquad \forall i \tag{3.2}$$

- Now, to be able to formulate the objective function, we associate a variable y_j for each PM j as given below. y_j will be either 0 or 1 depending on whether PM j has to be used or not.

$$y_j \geq x_{ij} \qquad \forall i, j \tag{3.3}$$

- And the objective function will be:

$$Minimize \sum y_j \tag{3.4}$$

Thus, summarizing the formulation:

$$Minimize : \sum y_j$$

$$Subject\ to :$$

$$\sum_i r_i.x_{ij} \leq 1 \qquad \forall j$$

$$\sum_j xij = 1 \qquad \forall i$$

$$y_j \geq x_{ij} \qquad \forall i, j$$

Once we solve this, y_j will tell us whether PM j is to be used or not, while the values of x_{ij} will give us the VM to PM mapping.

3.1.1 Evaluation of LP formulation

We implemented the formulation using the open source linear programming tool, lpsolve [4]. However, when the variables are modelled as binary, this problem is NP-hard [1]. Hence, although the ILP formulation gives an exact placement plan, it does not scale well with the increase in the number of Virtual or Physical Machines. Some quick experimentation revealed (table 3.1) that the time required to figure out a placement plan for as low as 20 VMs reached more than 8.5 hours! This is undesirable in a practical scenario.

Table 3.1: Time taken for ILP

Number of VMs	**Time taken**
6	30ms
9	4110ms
10	4261ms
20	>8.5hrs (!)

To avoid this, the constraint that each variable be 0 or 1 can be relaxed by using a weaker constraint instead, that each variable be fractional in the interval [0,1]. This technique is called LP Relaxation [2]. But using such a relaxation would mean getting the solutions to the LP as fractions (in the range [0,1]). Which means the solutions would suggest that different fractions of a virtual machine be allocated to different physical machines, which is undesirable. Hence, techniques for rounding the resulting solutions (values of the decision variables) to 0 or 1 have to be included.

In the next section, we describe different heuristics used for VM placement and give a comparative account of how they fair in relation to each other.

3.2 Heuristics for VM Placement

These heuristics take one VM at a time from amongst an ordered list of VMs and try to place it amongst a fixed set of Physical Machines. They mainly differ from each other in the way in which they choose Physical Machines to test for a possible placement. The algorithm 1 gives a generic algorithmic framework under which each of these heuristics fit in.

Algorithm 1

Input: Physical Machines: Current_Resource_Usage, list of vms_placed
Input: Ordered Matrix containing VM Requirements: req_matrix[ ]
Input: List of Physical Machines and their current usages: PM[ ]
Output: Allocation Matrix alloc_matrix[ ]

```
1: for each VM i do
2:    j=0
3:    order_of_PMvisit = <order defined by the corresponding heuristic>
4:    while VM not placed do
5:       nextPM = order_of_PMvisit[j++]
6:       Try to place VM i on nextPM considering capacity left on nextPM
7:       Update alloc_matrix[ ] if placed
8:       if number of PMs is finished and VM is not placed then
9:          Flag inability to find a placement plan.
10:      end if
11:   end while
12: end for
13: return alloc_matrix[ ]
```

As given above, we maintain an ordered matrix of VM requirements called 'req_matrix'. Also, we maintain a list of physical machines and their current resource usages (initially zero). For each VM, we determine the order in which these physical machines should be considered for placement. We visit the physical machines in that order, and place the VM onto the first physical machine with enough capacity. The decision of what order to choose depends on the heuristic we use. These heuristics are as described below.

Heuristics:

First Fit: This heuristic considers the Physical Machines in a sequential order for placement. Effectively, this tries to place the VM onto the first physical machine in the list that has enough space for it to fit.

Single Dimensional Best Fit: This heuristic visits the Physical Machines in a decreasing order of capacity used along a single dimension. That is, the Physical Machine which has the maximum of capacity used along the given dimension will be considered first.

Volume Based Best Fit: This visits the Physical Machines in the decreasing order of the product of their capacity usages along *all* dimensions. We have implemented this function as the product of the capacity usages, because multiplication is nearer to the real world concept of 'volume'. Thus, the volume of a PM is given by:

$$Volume_j = \prod_i C_{ij}$$

where, C_{ij} is the capacity usage of the physical machine j along dimension i.

Dot Product Based Fit: Here, the requirements of the VM along the specified dimensions and PM capacity usages along those dimensions are expressed as vectors. Then, a dot product of these two vectors is taken for each PM. The PMs are visited in decreasing order of this dot product.

Load Balanced Fit: In this heuristic, we visit the Physical Machines in an *increasing* order of a function of capacity used along all dimensions. The volume function is used here too. Clearly, this heuristic tries to place VMs on Physical Machines with highest remaining capacity. And hence, effectively, it provides a placement plan which balances the load amongst the given set of Physical Machines, thus using more number of Physical Machines than the other heuristics. Therefore, we use this heuristic as a benchmark for comparison amongst the set of heuristics as explained in the next section.

3.2.1 Evaluation of Placement Heuristics

Through this evaluation, we are interested in finding out answers to the following questions: given a sufficiently large number of sets of input configurations of VMs, (i) How many times does any particular heuristic work better than others? (ii) How much better does it perform? (iii) Generally, which heuristic works best amongst all the heuristics? (iv) Is there a heuristic which performs no better than the other heuristics?

In order to be able to differentiate between the heuristics, we define something called as a '**Packing Factor**'. For any given heuristic, the Packing Factor (P.F.) is defined as follows:

$$PackingFactor = \frac{Average\ no.\ of\ PMs\ used\ by\ the\ heuristic}{Average\ no.\ of\ PMs\ used\ by\ Load\ Balanced\ Fit} \tag{3.5}$$

The lower the packing factor, the better is the heuristic with respect to consolidation.

Setup:

To conduct the experiment, we generate thousand configuration files with virtual machine requirements distributed uniformly between 0.0 to 1.0. We run the heuristics over each of these configuration files and record the number of physical machines each heuristic took in each case. From this information, we calculate the number of times a heuristic performed better than others, i.e., the number of times it took lesser number of physical machines

than others. We repeated the experiment for different ordering of virtual machines. The results are as follows:

When the VMs are ordered in decreasing order of dimensions starting dimension 1, then dimension 2, then 3, the results are as shown in table below:

Table 3.2: Comparison of Traditional Heuristics - VMs ordered by Dimensions

	Number of machines taken							**Diff from the next best heuristic**					
Stat	nFF	n1DBF1	n1DBF2	n1DBF3	nVol	nDotP		FF	1DBF1	1DBF2	1DBF3	Vol	DotP
MAX	85	85	83	83	82	82		28	28	28	28	28	28
MIN	63	62	60	59	59	60		0	0	0	0	0	0
AVG	73.34	73.3	70.41	70.38	69.72	70.16		0.14	0.14	0.73	0.75	1.18	0.84
STDDEV	3.22	3.25	3.52	3.53	3.6	3.52		1.88	1.88	2.1	2.1	2.03	2.1
MEDIAN	73	73	70	71	70	70		0	0	0	0	1	0
80 %ile	76	76	73	73	73	73		0	0	1	1	1	1
MODE	73	73	71	72	69	71		0	0	0	0	1	0
Best Perf	9	9	382	396	771	469							
P.F.	0.73336	0.73304	0.70406	0.70383	0.69715	0.70155							

In the table above, the column nFF corresponds to the statistics for number of machines used by the First Fit heuristic. Similarly, n1DBF1, n1DBF2, n1DBF3 correspond to the same for Single Dimensional Best Fit along the three dimensions each, nVol corresponds to the number of machines used by Volume Based Fit and nDotP corresponds to the Dot Product Based Fit. The first half of the table reports values for the number of physical machines taken by the heuristics. Second half of the table reports values for the difference of the number of physical machines used from the next best performing heuristic, for cases when the heuristic performed the best. Same explanation applies to the table 3.3.

The table 3.3 given below shows results when the VMs are ordered in decreasing order of volume:

Table 3.3: Comparison of Traditional Heuristics - VMs ordered by Volume

	Number of machines taken							**Diff from the next best heuristic**					
Stat	nFF	n1DBF1	n1DBF2	n1DBF3	nVol	nDotP		FF	1DBF1	1DBF2	1DBF3	Vol	DotP
MAX	82	82	82	82	82	82		37	37	37	37	37	37
MIN	56	56	57	55	56	56		0	0	0	0	0	0
AVG	68.43	69.01	68.98	69	68.43	68.46		5.32	4.92	4.95	4.93	5.32	5.29
STDDEV	3.88	3.73	3.77	3.77	3.88	3.87		10.39	10.56	10.55	10.56	10.39	10.4
MEDIAN	68	69	69	69	68	68		1	0	0	0	1	1
80 %ile	72	72	72	72	72	72		2	1	1	1	2	2
MODE	68	70	68	69	68	68		1	0	0	0	1	1
Best Perf	803	447	473	451	803	778							
P.F.	0.68433	0.69006	0.68982	0.68996	0.68433	0.68456							

From the above results, we can observe a few points. Firstly, in table 3.2, First Fit has performed best only 9 times as opposed to much better performances by other heuristics. In table 3.3, it has performed better (803 times), but Volume Based Fit has also performed equally well. Hence, we can conclude that First Fit does not inherently perform better than the other heuristics and depends on the ordering of VMs.

Secondly, apart from the order of PM visit, ordering of VMs also affects the packing-Factor. The packing factors in table 3.3 are about 3-4% lower than those in table 3.2. Hence, VMs ordered by volume give better packingFactors.

In both the tables 3.2 and 3.3, Volume Based Fit has given packing factors of 0.697 and 0.684 respectively, both of which are lowest amongst all the heuristics. Hence, the Volume Based Fit seems to generally perform better than other heuristics.

However, as can be seen from both the tables, whenever the Volume Based Fit performed best, 80% of the times (see 80%ile value) it took atmost 1 or 2 machines less than the next best performing heuristic (true for other heuristics too) - But it did so in 771 and 803 cases out of 1000 respectively. Hence, in summary we can say that Volume Based Fit consistently performs better than the other heuristics but the savings are not much. Since the savings of volume based fit over other heuristics are not much, we can conclude that all heuristics are almost similar with respect to consolidation.

Since these peak based heuristics do not yield any significant savings in comparison to

each other, it would be worthwhile to explore the provisioning of virtual machines at off-peak values. In the next chapter, we present and evaluate an algorithm which advocates such provisioning at off-peak values by consideration of correlation of workload peaks and presents a strategy for adaptive provisioning based on the workload distribution.

Chapter 4

Adaptive Peakbuffer based Provisioning and Placement

In the last chapter, we have given a comparative evaluation of the traditional peak based correlation unaware VM placement heuristics. In this chapter, we present a strategy which takes into account the correlation between VM workloads and places non-correlated virtual machines together.

As we note in chapter 2, [16] has made some important observations about data center workloads. Based on these observations, they have come up with two placement methods namely, Correlation Based Placement(CBP) and Peak Clustering based Placement(PCP). In both these methods, they advocate the use of a 90-percentile based provisioning of virtual machines. In our work, we argue that rather than such a flat style of provisioning, a method which is more adaptive to the workload distribution would be desirable for provisioning. For this, we must first understand the concept of a peak buffer introduced by [16]. This is explained in the following section.

4.1 What is a Peak Buffer?

Peak Buffer is a range of values in the resource utilization timeseries which can be considered as 'peak' and be multiplexed with the peaks of other VMs placed on the same physical machine. The upper bound of the peak buffer's range is the maximum resource utilization value in the timeseries, and the lower bound is a value called as the 'size' (CB) of the application. This size is the value which is provisioned exclusively for this application. The values values above CB can be multiplexed with the peak buffers of other

applications having complementary patterns.

The fact that the peak buffer can be multiplexed with other peak buffers helps in placement of more VMs on the same physical machine. Just so that such multiplexing of peak buffers does not cause too many violations, it is desirable that the actual peaks inside any two such peak buffers do not occur at the same time. In other words, the peak buffers should be complementary to each other.

What are the desirable characteristics of a peakbuffer?
There are two main characteristics desirable in a peakbuffer:

- It should span across a large range of resource utilization values - a larger peakbuffer means smaller CB and hence, more potential multiplexing.
- However, this range should also contain lesser number of actual resource utilization values - the more the number of values, the higher the possibility of correlation with other peak buffers.

These characteristics can be better understood by going back to the figure in chapter 1, given again below:

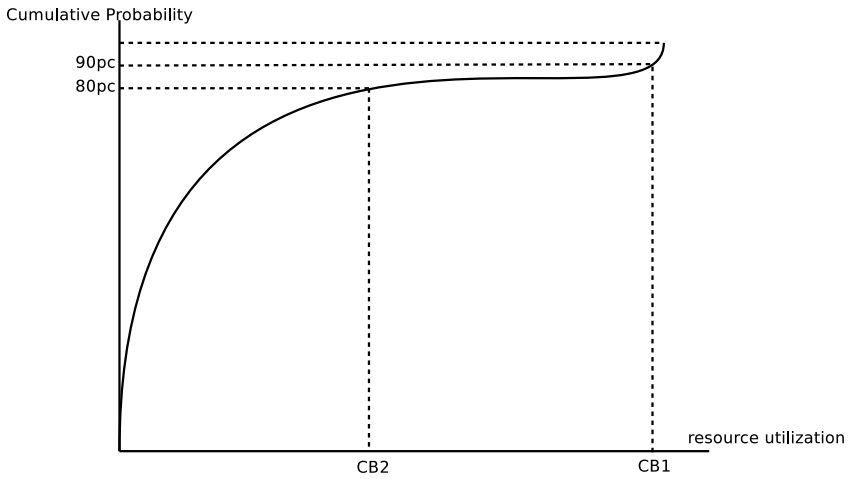

Figure 4.1: Drawbacks of a flat 90-percentile based provisioning

One point of observation here is that only 10% (90%ile - 80%ile) of the values in the timeseries fall within the range [CB2,CB1] - which means, this range is a sparse value range of the timeseries which corresponds to a flat portion in the CDF curve. Such sparse value ranges are desirable to be put in the peakbuffer.

Similarly, the ranges which correspond to steep portions of the CDF curve are undesirable for the peakbuffer, because they represent too many workload utilization values within a short range. (The flatness/steepness of the curve can be quantified using a ratio '$\Delta P/\Delta C$' which we have described in our algorithm).

Our Adaptive Provisioning Algorithm takes these observations into consideration and analyzes the CDF of the workload utilization timeseries to come up with a value for the size of the application. The detailed algorithm is described in the next section.

4.2 Adaptive Provisioning Algorithm

Algorithm 2 Adaptive Peakbuffer based Provisioning Algorithm

Input: probability: list of values along the probability axis of the CDF curve
Input: value: list of values along the workload usage axis of CDF curve
Input: violationThreshold
Input: windowCount
Input: ratioThreshold
Input: windowSize
Input: minProbabilityLimit

```
1: currentWindow = {peak, peak-windowSize}
2: n=0, smallWindowCount=0
3: while probability(currentWindow[0]) > minProbabilityLimit do
4:    if ((probability(currentWindow[0])-probability(currentWindow[1])) /
      (value(currentWindow[0])-value(currentWindow[1]))) ≤ ratioThreshold then
5:       CB = value(currentWindow[1])
6:    else
7:       n++
8:    end if
9:    if n > violationThreshold then
10:      break
11:   end if
12:   currentWindow=(currentWindow[1], currentWindow[1]-windowSize)
13:   if smallWindowCount < windowCount then
14:      smallWindowCount++
15:   else
16:      smallWindowCount=0, n=0
17:   end if
18: end while
19: return CB, peak-CB
```

The algorithm described above basically divides the CDF curve of the workload utilization timeseries into windows of size 'windowSize'. Starting with the rightmost such window (containing the peak usage), it calculates the ratio '$\Delta P/\Delta C$' for each window, where, ΔP equals the difference in the probability co-ordinates corresponding to the boundary points of the window, and ΔC equals the difference in the corresponding normalized workload utilization values. If this ratio falls below a threshold called 'ratioThreshold', the portion of the curve within that window is said to be flat, else, it is considered steep. 'violationThreshold' is a threshold on the number of windows with violations that are allowed out of a 'windowCount' number of windows. Therefore, if 'violationThreshold' or more number of windows out of 'windowCount' windows have crossed the ratioThreshold, the curve is considered steep. All workload values analyzed by the algorithm are normalized values within the range [0,1].

The algorithm basically skips the flat portions of the CDF curve while moving away from the peak, and provisions at a point just after which a steep curve starts. Once we find such a point(CB), the range between itself and the peak of the series forms the peak buffer.

Once this provisioning is done, we follow the following steps which lead to the placement of VMs:

1. **Creation of Envelope:** After the provisioning step, the timeseries is converted into an envelope with all values above CB being considered as 'peak' and all values below it being considered as CB. Such a calculation of CB, peak buffer and envelope of the series is done for each resource usage timeseries of each application.

2. **Calculation of Correlation Matrix:** In the next step, we calculate a correlation matrix for each resource k. Each element in this correlation matrix ($C_k(x, y)$) is the Pearson's correlation coefficient between the corresponding applications' envelopes.

   Next, we calculate a Joint Correlation Matrix (C) which considers all the resource types simultaneously, by taking the average of the corresponding correlation coefficients for all resources:

   $$C(x, y) = \frac{\sum_k C_k(x, y)}{number\ of\ resources} \tag{4.1}$$

3. **Placement Step:** In the next step, we cluster these applications into groups of least correlated applications with an upper bound of physical machine capacity on

the cluster size. We start with N applications as N clusters. Next, we find the least correlated pair of clusters c1 and c2. If the correlation between these clusters is below a correlation threshold, and their total capacity for any resource alongwith the peak buffer does not exceed the physical machine capacity, we merge them into one. Thus, we go on merging clusters until no more clusters can be merged. Merging clusters essentially means that we add up their sizes(CBs) and take maximum of their peak buffers as the peak buffer of the merged cluster. This, we do for each resource. Also, correlation between two clusters means the joint correlation between the most correlated pair of applications, one from each cluster.
Each cluster thus formed represents the applications that can be placed on a physical machine.

A small example demonstrating the calculation of Joint Correlation Matrix and clustering based on it is given in Appendix C.

4.3 Evaluation of Adaptive Provisioning Algorithm

In this section, we evaluate our adaptive peak buffer based provisioning algorithm against a flat 90-percentile based provisioning. To do this, we need to answer two questions: (i) Are there lesser, or atleast acceptably low number of violations with respect to the provisioned capacities? and (ii) Does the algorithm use lesser number of physical machines?

Setup:

The setup we have used for the evaluation is shown in the figure 4.2.

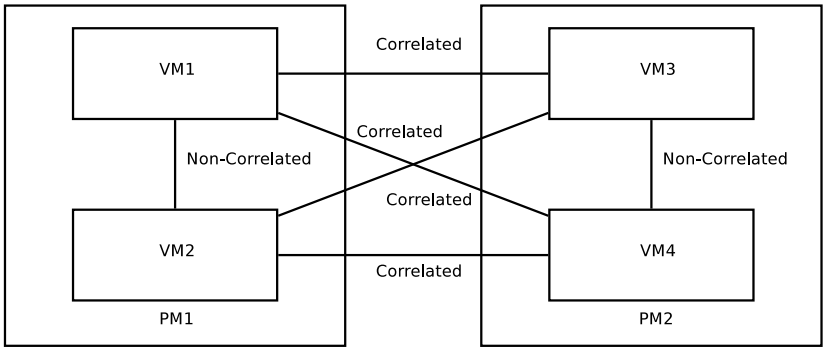

Figure 4.2: Setup for Evaluation of Adaptive Provisioning Algorithm

As can be seen from the figure, we choose 4 applications (VMs) with the following inter-application correlations:

$$\begin{bmatrix} & App1 & App2 & App3 & App4 \\ App1 & & non-corr & corr & corr \\ App2 & non-corr & & corr & corr \\ App3 & corr & corr & & non-corr \\ App4 & corr & corr & non-corr & \end{bmatrix}$$

That is, we deploy complementary workloads on the pairs [VM1,VM2] and [VM3,VM4] respectively. We use CPU workloads for this experiment.

For the experiment, we generate a timeseries of 100 Gaussian random numbers for each VM which give the required correlations according to the figure 4.2. Next, we place the VMs according to the recommendation of the algorithm, deploy CPU workloads on them which follow the respective timeseries, and monitor violations - if the number of violations are negligible, the peakbuffer strategy is validated.

While deploying the timeseries onto the VMs, one challenge is that following the timeseries with a granularity of a second is difficult. So, we stretch the timeseries to a higher granularity, say 4 seconds. That is, each data point in the timeseries is deployed as a load level for the duration of 4 seconds. Thus, the total duration of the workload is 400 seconds. Logging can be at the same or a finer granularity and is done every 2 seconds.

When is it a violation?
If the measured total usage of a resource on the physical machine is greater than the sizes(CB) of all applications + the peakbuffer on that physical machine, it is a violation.

Results:

The results of the experiment are given in the table 4.1. In this table, the column CB represents the size of each timeseries provisioned by our algorithm, whereas C90 represents the size in a 90-percentile based approach. The next two columns give the capacity provisioned on physical machines, by our algorithm and by 90-percentile approach respectively. For e.g., the first row indicates that our algorithm has used 2 physical machines with provisioned capacities of 95.75 and 96.48, whereas 90-percentile approach needs 4

physical machines with provisioned capacities 59.26, 46.96, 63.42 and 49.01 respectively. The next column titled 'nViolations (Adaptive)' gives the number of violations reported when the timeseries were deployed according to the recommendation of our algorithm. Thus, 1 and 2 violations were reported respectively on the two physical machines used by our algorithm in the first case. The next column mentions the intensities of these violations in terms of the number of percentage points the measured values went above the provisioned values on the physical machines. The next two columns named 'nViolations(90)' and 'Violation Intensity(90)' report the same values for the 90-percentile based approach. The last two columns report the number of Physical Machines used by both the approaches. Also, the different rows in the table represent the runs for different sets of timeseries.

Table 4.1: Adaptive Provisioning Experiment Results

Sr No.	CB	C90	Provisioned capacity (Adaptive)	Provisioned capacity (90)	nViolations (Adaptive)	Violation Intensity (Adaptive)	nViolations (90)	Violation Intensity (90)	No of PMs (Adaptive)	No of PMs (90)
1	49.52	51.8	95.75, 96.48	59.26, 46.96, 63.42, 49.01	1, 2	(4.25), (1.82, 1.82)	0, 0, 0, 0	-	2	4
	24.84	32.54								
	46.74	53.88								
	27.73	31.02								
2	50.66	52.89	90.47, 96.89	64.71, 41.01, 93.65	5, 0	(0.43, 4.13, 2.63, 0.73, 6.63), -	0, 0, 0	-	2	3
	32.18	32.18								
	50.85	54.02								
	28.16	31.34								
3	54.95	56.01	92.52, 97.43	66.79, 42.40, 95.96	3, 0	(3.77, 0.67, 5.47), -	0, 0, 0	-	2	3
	30.65	33.96								
	46.86	50.29								
	26.94	31.89								
4	50.31	52.14	96.65, 97.08	68.79, 46.77, 58.34, 49.10	2, 0	(4.64, 2.04), -	0, 0, 0, 0	-	2	4
	24	32.56								
	47.56	52.14								
	23.27	31.44								
5	46.37	52.65	94.33, 95.41	59.67, 45.07, 99.12	0, 0	-	0, 0, 0	-	2	3
	30.98	35.77								
	50.35	53.51								
	28.07	33.16								
6	49.17	53.17	95.23, 91.75	62.24, 37.89, 97.15	0, 2	-, (1.74, 7.84)	0, 0, 0	-	2	3
	30.2	31.31								
	49.53	54.61								
	29.52	32.38								
7	49.35	52.99	96.32, 98.87	70.31, 47.14, 66.64, 46.71	0, 0	-	0, 0, 0, 0	-	2	4
	28.56	34.66								
	48.41	54.53								
	29.69	34.32								
8	46.48	53.76	97.19, 96.88	68.32, 45.14, 64.49, 46.92	0, 0	-	0, 0, 0, 0	-	2	4
	28.87	33.1								
	49.97	54.64								
	26.92	31.93								
9	50.67	54.29	99.25, 93.07	69.87, 45.48, 99.75	0, 0	-	0, 0, 0	-	2	3
	29.38	32.36								
	45.15	55.01								
	29.63	31.06								
10	50.14	51.98	97.56, 93.35	63.46, 43.22, 71.73, 41.90	0, 0	-	0, 0, 0, 0	-	2	4
	27.43	34.22								
	50.69	53.6								
	25.84	34.95								

From the table, we can observe that 90-percentile based approach always takes more than 2 physical machines. Hence, our algorithm always takes lesser number of physical machines than 90-percentile based approach. However, the tradeoff is that this may reflect in an increased number of violations for our algorithm. Hence, let us analyze the data given in the table for violations caused by each of the approaches.

As we can see from the table, 90-percentile based provisioning gives no violations in any of the runs. However, if our algorithm also gives reasonably low number of violations, it can still be called better because of the better consolidation.

To this end, for our algorithm, we first plot a CDF of the number of violations observed on each physical machines across all cases . This is given in figure 4.3.

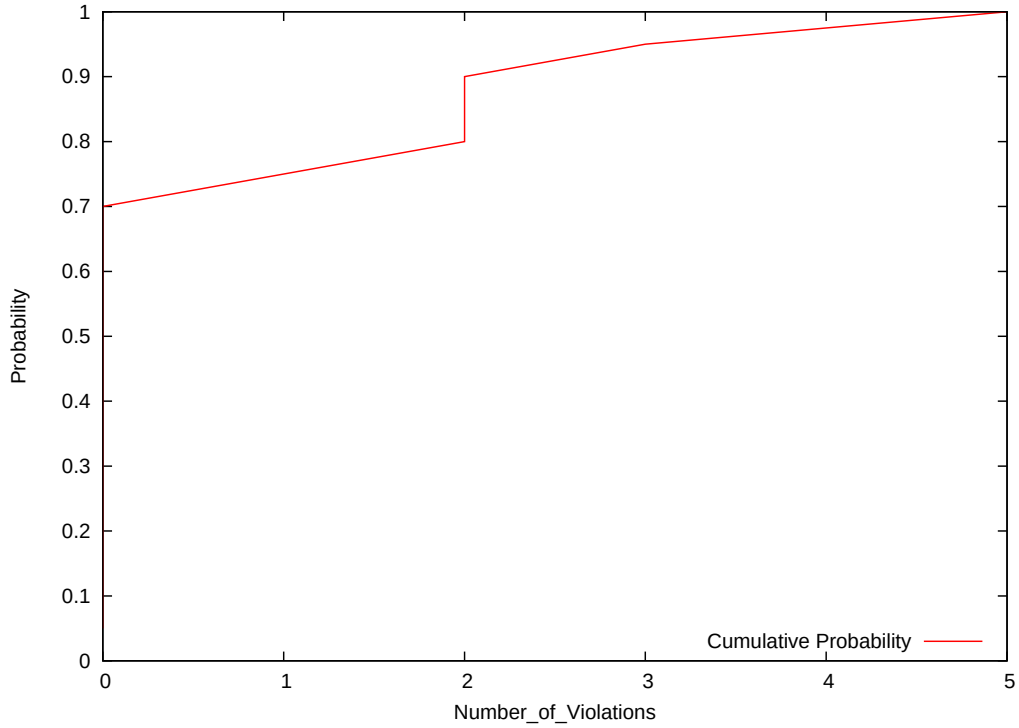

Figure 4.3: CDF of the number of violations

From the figure, it is clear that, about 70% of the times, there were no violations observed. And, 90% of the times, only two or less violations were observed. Hence, we can conclude that our algorithm causes a reasonably low number of violations.

Next, we also analyse the intensity of these observed violations. To this end, we plot a CDF of the intensity of violations in percentage points. This is given in figure 4.4.

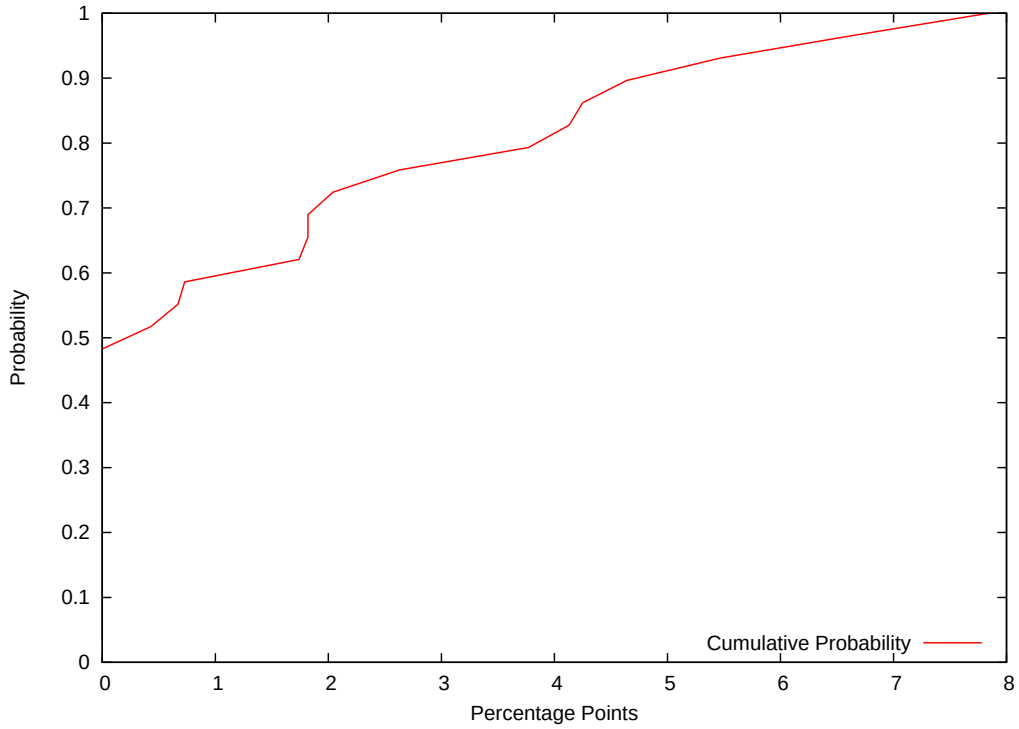

Figure 4.4: CDF of the intensity of violations

From this figure, we observe that, 80% of the times, violation intensity was below 4 percentage points. And about 60% of the times, it was less than 1 (negligible). This means that even within the reasonably low number of observed violations, the intensities of violation were not very high.

Hence, from the above analysis, we can say that our Adaptive Provisioning Algorithm provides a better consolidation of VMs than a 90-percentile based strategy. And it does so with a reasonably low number of violations of the provisioned capacity. Also, whenever there is a violation, its intensity is reasonably low.

Chapter 5

Conclusions and Future Work

In our work, we have first evaluated some traditional correlation unaware approaches using a peak value based provisioning, and found that they are almost similar to each other with respect to consolidation. Next, we have presented an off-peak based adaptive provisioning algorithm for virtual machines which is based on the analysis of the workload distribution of the corresponding applications. Our study shows that such an adaptive provisioning algorithm performs better than a flat 90 percentile based provisioning which has been used in literature. We observe that our algorithm performs better packing of virtual machines while also ensuring reasonably low number of violations of the provisioned capacity.

As a part of future work, it will be useful to bring in other constraints during VM placement such as consideration for affinity or interference between virtual machines. Also, the activities at the guest operating system may introduce some overheads for the host operating system. One possible direction could be the consideration of such virtualization overheads in the calculations for placement, for different virtualization technologies like Xen, KVM, etc. One immediate extension to our work can be the decision of the configuration parameters to our algorithm by looking at the SLA requirements that are typically given by the customers to data center providers.

Appendix A

Sample Requirements Matrix

Sample Requirements Matrix:
A sample requirements matrix is shown below:

$$\begin{bmatrix} 0.2 & 0.4 & 0.3 \\ 0.2 & 0.4 & 0.5 \\ 0.6 & 0.2 & 0.3 \\ 0.6 & 0.2 & 0.5 \end{bmatrix}$$

In this matrix, each row corresponds to one virtual machine's requirements as fractions of total capacity of a physical machine along the dimensions of CPU, network and disk I/O. For example, the first row specifies that the corresponding VM uses 20% of CPU, 40% of network capacity and 30% of disk I/O capacity of a physical machine.

Appendix B

Configuration File Format

Configuration File:
To be able to read the inputs like the requirements matrix and the number of Physical Machines available, we use a configuration file. We define a specific format for the configuration file as given below.

```
Dimensions: <no_of_dimensions>
VMs: <no_of_VMs>
PMs: <no_of_PMs>
Matrix:
r11  r12  ...  r1d
r21  r22  ...  r2d
:     :    :     :
rn1  rn2  ...  rnd
```

(where under 'Matrix', the Requirements Matrix is to be specified)

Appendix C

Example for VM Placement in Adaptive Provisioning Algorithm

Let the correlation matrices corresponding to CPU, network and disk usage for four applications A1, A2, A3 and A4 be:

$$\begin{bmatrix} 1.0 & 0.9 & 0.6 & -0.7 \\ 0.9 & 1.0 & -0.7 & 0.6 \\ 0.6 & -0.7 & 1.0 & 0.9 \\ -0.7 & 0.6 & 0.9 & 1.0 \end{bmatrix}, \begin{bmatrix} 1.0 & 0.9 & 0.7 & -0.8 \\ 0.9 & 1.0 & -0.8 & 0.7 \\ 0.7 & -0.8 & 1.0 & 0.9 \\ -0.8 & 0.7 & 0.9 & 1.0 \end{bmatrix} \textit{and} \begin{bmatrix} 1.0 & 0.9 & 0.8 & -0.9 \\ 0.9 & 1.0 & -0.9 & 0.8 \\ 0.8 & -0.9 & 1.0 & 0.9 \\ -0.9 & 0.8 & 0.9 & 1.0 \end{bmatrix}$$

The joint correlation matrix is calculated by taking the average of corresponding elements of these three matrices. Hence, the Joint Correlation Matrix in this case will be as follows:

	A1	A2	A3	A4
A1	1.0	0.9	0.7	-0.8
A2	0.9	1.0	-0.8	0.7
A3	0.7	-0.8	1.0	0.9
A4	-0.8	0.7	0.9	1.0

This Joint Correlation Matrix is used in the further calculations for clustering. For simplicity, let us assume all resources have maximum available capacity as 100 and the same provisioned values for applications as given in table C.1 below:

Table C.1: Example Provisioned Application Capacities

	A1	A2	A3	A4
CB	20	30	20	30
Peak Buffer	30	40	30	40

According to the Joint Correlation Matrix, the application pairs (A1, A4) and (A2, A3) have the least correlation amongst themselves (-0.8), and hence can be grouped together. Thus, we get two clusters:

Cluster	**Total Size**	**Peak Buffer**	**Provisioned Size**
(A1,A4)	20+30 = 50	max(30,40) = 40	50+40 = 90
(A2,A3)	30+20 = 50	max(40,30) = 40	50+40 = 90

These clusters cannot be merged further since their combined size is greater than the maximum available capacity of 100.

Each of these clusters now represents the sets of applications to be grouped together to be placed on the same physical machine.

Bibliography

[1] http://en.wikipedia.org/wiki/0-1_integer_programming#integer_unknowns.

[2] http://en.wikipedia.org/wiki/linear_programming_relaxation.

[3] http://www.slideshare.net/topline/virtualization-business-case.

[4] lpsolve - mixed integer linear programming (milp) solver.

[5] Martin Bichler, Thomas Setzer, and Benjamin Speitkamp. Capacity Planning for Virtualized Servers. *Presented at Workshop on Information Technologies and Systems (WITS), Milwaukee, Wisconsin, USA, 2006*.

[6] Norman Bobroff, Andrzej Kochut, and Kirk A. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *Integrated Network Management*, pages 119–128. IEEE, 2007.

[7] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. Optimal virtual machine placement across multiple cloud providers. In Markus Kirchberg, Patrick C. K. Hung, Barbara Carminati, Chi-Hung Chi, Rajaraman Kanagasabai, Emanuele Della Valle, Kun-Chan Lan, and Ling-Jyh Chen, editors, *APSCC*, pages 103–110. IEEE, 2009.

[8] Ming Chen, Hui Zhang, Ya-Yunn Su, Xiaorui Wang, Guofei Jiang, and Kenji Yoshihira. Effective vm sizing in virtualized data centers. In *the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, May 2011.

[9] Kernel Based Virtual Machine (KVM). http://www.linux-kvm.org/page/Main_Page.

[10] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 11–20, New York, NY, USA, 2010. ACM.

[11] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Autonomic virtual resource management for service hosting platforms. In *CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Paul Barham et. al. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003.

[13] Anjana Shankar. Virtual machine placement in computing clouds. Technical report, Indian Institute of Technology Bombay, apr 2010.

[14] Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[15] Benjamin Speitkamp and Martin Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Transactions on Services Computing*, 99(RapidPosts), 2010.

[16] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 28–28, Berkeley, CA, USA, 2009. USENIX Association.

[17] VMWare. http://www.padovatech.com/solutions/virtualization/vmware-virtualization-cloud-computing/.