

**ENERGY-AWARE FAST OPTIMIZATION IN CLOUD  
COMPUTING RESOURCE PROVISIONING**

BY

**Kh. Shahzada Shahid**

A Thesis Presented to the  
DEANSHIP OF GRADUATE STUDIES

**KING FAHD UNIVERSITY OF PETROLEUM & MINERALS**

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the  
Requirements for the Degree of

**MASTER OF SCIENCE**

In

**COMPUTER ENGINEERING**


May, 2015

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS  
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **KHAWAJA SHAHZADA SHAHID** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Thesis Committee

 Sadiq M. Sait. M

Dr. Sadiq M. Sait (Adviser)




Dr. Alaaeldin Amin (Member)



Dr. Ahmad Khayyat. (Member)



Dr. Ahmad Almulhem  
Department Chairman

  
Dr. Salam A. Zummo  
Dean of Graduate Studies



21/6/15

Date

©Kh. Shahzada Shahid  
2015

*To my parents, whose words of encouragement and push for tenacity ring in my ears. To my brothers and sisters, who have supported me throughout the process. To my wife, for being with me.*

# ACKNOWLEDGMENTS

First of all, infinite thanks and complete gratitude to the almighty Allah for his unstoppable blessings. Many thanks, to my father, my mother, and my whole family for their trust and support through this work as well as through all of my life.

Special thanks to my mentor and advisor Professor Sadiq M. Sait for his appreciated guidance and directions. Thanks to my committee members: Dr. Alaaeldin Amin and Dr. Ahmad Khayyat for their motivation and valuable suggestions.

Thanks to COE department and KFUPM for giving me this opportunity. Finally, special thanks are due to my senior colleagues at KFUPM for their Help and prayers.

# TABLE OF CONTENTS

|   |             |
|---|-------------|
| <b>ACKNOWLEDGEMENTS</b>                         | <b>iii</b>  |
| <b>LIST OF TABLES</b>                           | <b>vii</b>  |
| <b>LIST OF FIGURES</b>                          | <b>viii</b> |
| <b>ABSTRACT (ENGLISH)</b>                       | <b>x</b>    |
| <b>ABSTRACT (ARABIC)</b>                        | <b>xii</b>  |
| <b>CHAPTER 1 INTRODUCTION</b>                   | <b>1</b>    |
| 1.1 Background and Terminology . . . . .        | 2           |
| 1.1.1 Cloud computing . . . . .                 | 2           |
| 1.1.2 Deployment Models . . . . .               | 2           |
| 1.1.3 Service Models . . . . .                  | 3           |
| 1.1.4 Virtualization . . . . .                  | 5           |
| 1.2 Motivation . . . . .                        | 8           |
| 1.3 Research Objective . . . . .                | 8           |
| 1.4 Constraints and Complexity . . . . .        | 9           |
| 1.5 Thesis Contributions . . . . .              | 9           |
| 1.6 Thesis Organization . . . . .               | 10          |
| <b>CHAPTER 2 LITERATURE REVIEW</b>              | <b>12</b>   |
| 2.1 Vector Bin Packing Problem (VBPP) . . . . . | 13          |
| 2.1.1 Literature Review . . . . .               | 13          |

|  |   |           |
|--|---|-----------|
| 2.2  | Virtual Machine Assignment Problem . . . . .        | 15        |
| 2.2.1  | Optimization Formulation . . . . .                  | 16        |
| 2.2.2  | Literature Review . . . . .                         | 17        |
| 2.2.3  | Iterative Heuristics . . . . .                      | 21        |
| <b>CHAPTER 3 PROPOSED APPROACH</b>                           |   | <b>23</b> |
| 3.1  | Simulated Evolution . . . . .                       | 23        |
| 3.1.1  | <b>Goodness Evaluation</b> . . . . .                | 25        |
| 3.1.2  | <b>Selection</b> . . . . .                          | 26        |
| 3.1.3  | <b>Allocation</b> . . . . .                         | 29        |
| 3.2  | Complexity Analysis . . . . .                       | 31        |
| <b>CHAPTER 4 PERFORMANCE EVALUATION</b>                      |   | <b>33</b> |
| 4.1  | Simulation Setup . . . . .                          | 33        |
| 4.1.1  | Work Load . . . . .                                 | 35        |
| 4.2  | Results and Discussion . . . . .                    | 37        |
| 4.2.1  | Scalability of SimE . . . . .                       | 42        |
| <b>CHAPTER 5 DATA SET GENERATOR AND IMPROVED LOWER BOUND</b> |   | <b>44</b> |
| 5.1  | Data Generation for Multidimensional VBPP . . . . . | 44        |
| 5.1.1  | Parameters . . . . .                                | 45        |
| 5.1.2  | Procedure . . . . .                                 | 45        |
| 5.1.3  | Example . . . . .                                   | 47        |
| 5.2  | Improved Lower Bound . . . . .                      | 48        |
| 5.2.1  | Procedure for Calculation of $LB$ . . . . .         | 48        |
| <b>CHAPTER 6 EXTENDED EXPERIMENTS</b>                        |   | <b>53</b> |
| 6.1  | Performance Evaluation . . . . .                    | 53        |
| 6.1.1  | Simulation Setup . . . . .                          | 54        |
| 6.1.2  | Results and Discussion . . . . .                    | 55        |

|                                      |    |
|--------------------------------------|----|
| CHAPTER 7 CONCLUSION AND FUTURE WORK | 60 |
| REFERENCES                           | 62 |
| VITAE                                | 71 |



# LIST OF TABLES

|     |   |    |
|-----|---|----|
| 4.1 | Performance comparison of $\text{FFD}_{imp}$ , $\text{LL}_{imp}$ , SA and SimE. . . . . | 39 |
| 5.1 | Sample data set with negative correlation. . . . .                                      | 47 |
| 5.2 | Sample data set. . . . .  | 49 |
| 5.3 | Reduced problem set at an intermediate stage of lower bound procedure. . . . .          | 51 |
| 5.4 | Iterative steps to calculate set $t_2$ . . . . .  | 52 |
| 6.1 | Performance comparison of $\text{FFD}_{NB}$ , $\text{FFD}_{DP}$ , SA and SimE. . . . .  | 59 |

# LIST OF FIGURES

|     |   |    |
|-----|---|----|
| 1.1 | Cloud Deployment Models [1]. . . . .  | 3  |
| 1.2 | Cloud service models [2]. . . . .   | 4  |
| 1.3 | Virtualization [10]. . . . .  | 6  |
| 1.4 | The above diagrams show (a) Poor utilization with 5 active PMs,<br>(b) moderate utilization with 3 active PMs, and (c) optimal uti-<br>lization with 2 active PMs only. . . . .                                     | 7  |
| 2.1 | Example of MDVBPP for $R = 4$ . . . . .   | 14 |
| 2.2 | Bin-centric First Fit Decreasing (FFD) Procedure. . . . .   | 21 |
| 3.1 | Flowchart of SimE. . . . .  | 24 |
| 3.2 | Allocation of 6 VMs on 3 PMs. . . . .   | 27 |
| 3.3 | Simulated Evolution Algorithm for VM assignment. . . . .  | 28 |
| 3.4 | Sorted PMs and VMs for allocation. . . . .  | 30 |
| 3.5 | Allocation of selected VMs. . . . .   | 31 |
| 4.1 | Pseudo code to generate random problem instances with certain<br>correlations. . . . .  | 36 |
| 4.2 | Run time of $\text{FFD}_{\text{imp}}$ , $\text{LL}_{\text{imp}}$ , SA and SimE with 200 VMs for cases<br>of (a) $\overline{v^c} = \overline{v^m} = 25\%$ and (b) $\overline{v^c} = \overline{v^m} = 45\%$ . . . . . | 40 |
| 4.3 | Change in number of active PMs with iterations in (a) SimE and<br>(b) SA. . . . .   | 41 |
| 4.4 | SimE: Change in the average goodness of VMs with iterations. . .  | 41 |

|     |  |    |
|-----|--|----|
| 4.5 | SimE algorithm run-time versus number of VM requests for different correlations. . . . . | 43 |
| 5.1 | Procedure to generate data set for multidimensional VBPP. . . .                          | 46 |
| 5.2 | Procedure for determining the lower-bound $LB_2$ . . . . .                               | 50 |
| 6.1 | Change in number of active/open Bins with iterations in (a) SimE and (b) SA. . . . .     | 57 |
| 6.2 | SimE: Change in the average goodness of items with iterations. .                         | 58 |

# THESIS ABSTRACT

**NAME:** Khawaja Shahzada Shahid

**TITLE OF STUDY:** Energy-aware Fast Optimization in Cloud Computing Resource Provisioning

**MAJOR FIELD:** Computer Engineering

**DATE OF DEGREE:** MAY, 2015

*Cloud Computing Services business is rapidly growing in today's IT market. Its sharp growth is producing many challenges for cloud managers. One primary concern is to efficiently manage the cloud resources, i.e., to maximize utilization of hardware with minimum power consumption. Virtual Machine (VM) consolidation is a very helpful approach to achieve these goals. In the context of green computing, I investigate the VM assignment problem. In this work, the engineering of a non-deterministic iterative heuristic known as Simulated Evolution (SimE) is described to solve the well-known NP-hard problem of assigning VMs to hardware hosts. A 'goodness' function which is related to the target objective of the problem is defined. It guides the moves and helps traverse the search space in an intelligent manner. In the process of evolution, VMs with high goodness value have a smaller probability*

*of getting perturbed, while those with lower goodness value may be reallocated via a compound move. For performance evaluation, a new data set generation method is proposed that covers a wide variety of parameters that can potentially affect the difficulty of the problem. In addition to this, a new implementation of a tighter lower bound method is also presented. Results are compared with those published in previous studies, and it is found that the proposed approach is efficient both in terms of solution quality and computational time.*

## ABSTRACT (ARABIC)

### ملخص الرسالة

الاسم الكامل: خواجة شهزاده شاهد

عنوان الرسالة: التحسين السريع الشامل للطاقة في موارد الحوسبة السحابية

التخصص: هندسة حاسب آلي

تاريخ الدرجة العلمية: مايو 2015

تشهد اليوم تجارة خدمات أعمال الحوسبة السحابية نمواً سريعاً في سوق تقنية المعلومات، ولقد نتج عن هذا النمو الحاد العديد من التحديات لمديري السحابة. ويتمثل أحد الاهتمامات الرئيسية في هذا الموضوع في كفاءة إدارة الموارد السحابية، أي تحقيق أقصى استفادة من الأجهزة مع تحقيق الحد الأدنى من استهلاك الطاقة. ويعد دمج وتوحيد الآلة الافتراضية (VM) في الأجهزة المستضيفية نهجاً مفيداً جداً لتحقيق هذه الأهداف. وفي سياق الحوسبة المهمة بالبيئة، قمت بدراسة مشكلة تعيين الآلة الافتراضية، حيث تم وصف تصميم وهندسة الكشف عن مجريات الأمور التكرارية غير القطعية التي تعرف بتطور المحاكاة (SimE)، ثم تم استخدام هذا التصميم لحل مشاكل معروفة من النوع غير متعدد الحدود الصعب الخاصة بربط الآلات الافتراضية بالأجهزة المستضيفية. وقمنا بتعريف دالة صلاحية مرتبطة بالهدف من الخوارزمية بحيث يتم توجيه تحركات الخوارزمية والمساعدة في اجتياز فضاء البحث بطريقة ذكية. وفي عملية التطور كانت الآلة الافتراضية ذات الصلاحية الأكبر تتميز باحتمالية صغيرة لإجراء اضطرابات عليها، بينما إذا كانت الصلاحية قليلة فيتم إعادة تعيينها عن طريق إجراء تحركات مركبة. ولتقييم الأداء تم تقديم طريقة إنتاج مجموعة مدخلات جديدة لتغطي مجموعة كبيرة من المعطيات التي لها احتمالية تأثير على صعوبة المسألة، وبالإضافة إلى هذا تم تقديم تطبيق جديد يشكل حد أدنى أقل لهذه المسألة. وتمت مقارنة النتائج مع تلك التي نشرت في دراسات سابقة، وتبين أن النهج المقترح فعال سواء من حيث نوعية الحل أو الوقت المستغرق في العمليات الحسابية.

## CHAPTER 1

# INTRODUCTION

In recent years, cloud-based data centers have emerged as a popular choice for hosting and delivering IT services. Due to economies of scale and ease of accessibility, the IT industry is rapidly adopting the cloud computing paradigm [3]. Public-cloud market growth rate was recently forecasted to be 18.5% in the Gartner report 2013 [4]. With its fast growing market size, its energy consumption is also increasing alarmingly. In 2010, electricity consumption by data centers was estimated to be 1.1 – 1.5% of total electricity usage worldwide with an expectation of further growth [5, 6]. In data centers, energy is not only consumed for running the physical machines but also for cooling the infrastructure. It is estimated that energy consumption accounts for approximately 12% of monthly operational expenditures of a typical data center [7]. Also, large-scale data centers are facing regulatory restrictions on energy usage by governmental agencies who are promoting green computing [8]. Hence, reducing energy consumption is a primary goal of today's data center operations.

## 1.1 Background and Terminology

### 1.1.1 Cloud computing

Cloud computing generally refers to the deployment and usage of compute and storage resources over the internet. The usage of these resources follow pay-as-you-go pricing models. The most comprehensive definition of cloud computing, provided by the National Institute of Standards and Technology (NIST) is as follows “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [9].”

### 1.1.2 Deployment Models

The word “cloud” refers to a big network of shared resources. The cloud infrastructure may be designed either to serve different functional units of the single organization or to be shared among several organizations to reduce the overall infrastructure and operational cost of the IT sector. From the deployment point of view, the cloud infrastructure has four major types: *Private*, *Public*, *Community* and *Hybrid cloud*. In a private cloud, the cloud infrastructure is solely utilized by a single organization, while in public cloud, several organizations can publicly use these resources on lease basis through a network link. Community cloud also shares resources with multiple organizations, however it slightly differs from public cloud in a sense that provision of resources is restricted to group of



organizations that share common concerns and belong to a specific community. The scenario where an organization extends the capacity of its private cloud by leasing resources from a public or community cloud is referred to as hybrid cloud.

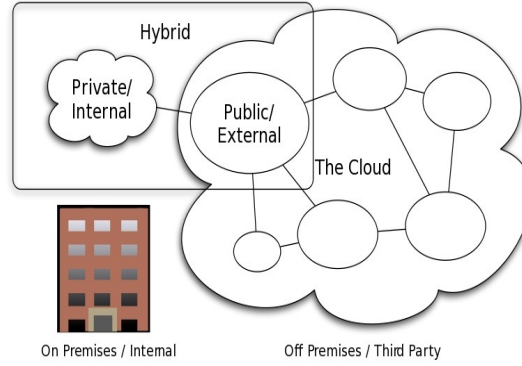


Figure 1.1: Cloud Deployment Models [1].

### 1.1.3 Service Models

Architecture of a cloud computing environment is best described by a layered model as shown in Fig. 1.2. This model has three layers namely *Application*, *Platform* and *Infrastructure*. Each one of these layers provides a different type of services depending on the abstraction level at which the service is provided. At the application layer, all the underlying hardware and operating system information is hidden from the end users. Users can access the cloud hosted applications through either a thin client e.g., web browser or a program interface. This type of service is known as Software-as-a-service (SaaS). Facebook.com and salesforce.com are common examples of this service model. At the second layer, cloud provider offers computing platforms such as operating systems and application frameworks. At this level, users have the privileges to develop and deploy their own applica-

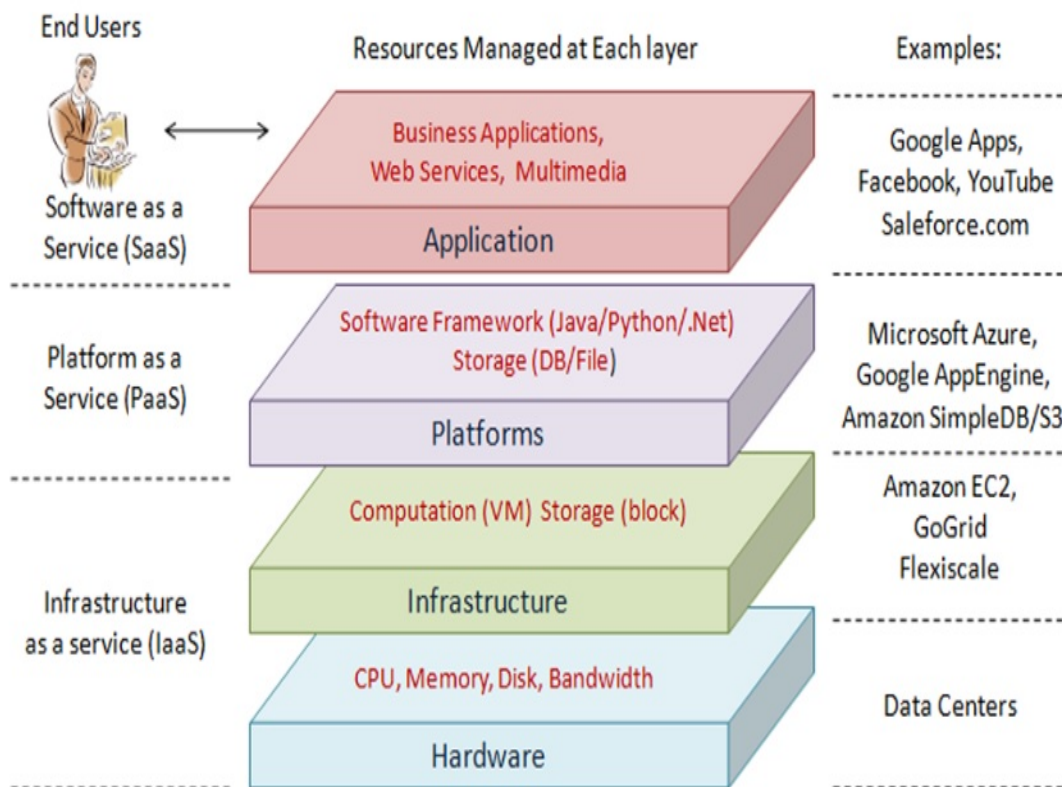


Figure 1.2: Cloud service models [2].

tions onto the cloud infrastructure. Microsoft Azure, Google App Engine and Amazon SimpleDB/S3 are typical examples [2]. This service model is known as Platform-as-a-service (PaaS). Lowest layer in the hierarchy offers Infrastructure-as-a-service (IaaS). It manages fundamental computing resources that include storage, processing, network, etc. These resources are provided to the end user, usually in terms of storage blocks and virtual machines (VMs) using virtualization technologies such as VMware [10], KVM [11], and Xen [12]. This layer is also known as virtualization layer [2].

#### **1.1.4 Virtualization**

Virtualization technology is the main technology that actually made cloud-computing possible. It allows us to divide a single physical machine into multiple execution units known as virtual machines (VMs). Multiple VMs can co-exist on a single physical machine (PM) as shown in Fig. 1.3. In each of the above mentioned service models, a user can request as much or as little computing resources as he desires. User requests are mapped to VMs with desired characteristics, and each VM, with its own operating system, works in an isolated environment while sharing the underlying machine's computing resources with other VMs. In this way, less number of active physical machines are needed for the same amount of workload.

Fig. 1.4 illustrates the benefits of virtualization and how optimal VM assignment helps to reduce the number of active PMs in a data center. Consider the

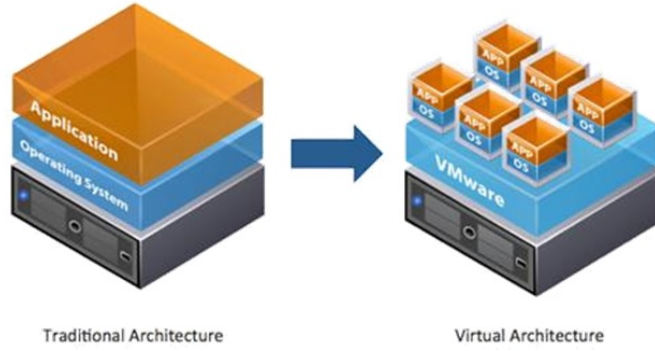
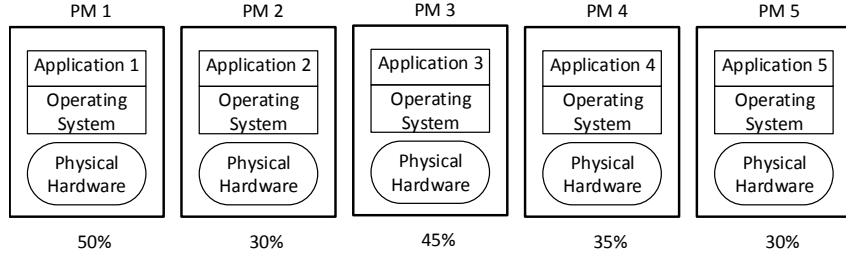
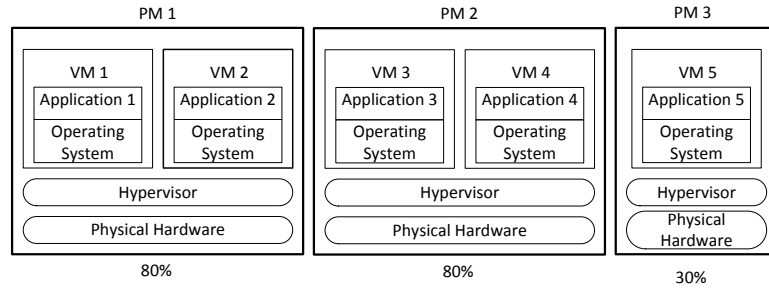


Figure 1.3: Virtualization [10].

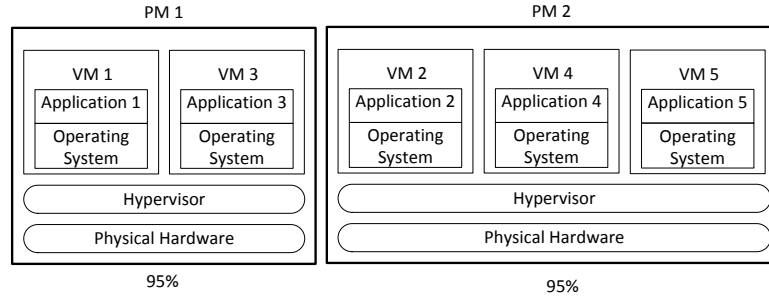
situation illustrated in Fig. 1.4. Five applications are running on five different PMs (Fig.1.4(a)). None of these applications is using more than 50% of the physical resources, resulting in huge resource wastage and unnecessary power consumption. With virtualization technology we can execute these applications in VMs, and these VMs can then be placed in a fewer number of physical machines that better utilize the available resources with reduced power cost (Fig.1.4(b)). However, by making an optimal placement of these VMs, we can achieve even higher utilization with lower power consumption (Fig.1.4(c)). For this purpose we need an efficient algorithm that can assign these virtual machines to minimum number of possible physical machines. In literature, this is known as virtual machine assignment problem. It closely resembles with the vector bin packing problem (VBPP) that is a well known NP-hard problem [13].



(a) Applications running on individual PMs.



(b) Applications running on individual VMs that share physical resources.



(c) Optimal placement of VMs

Figure 1.4: The above diagrams show (a) Poor utilization with 5 active PMs, (b) moderate utilization with 3 active PMs, and (c) optimal utilization with 2 active PMs only.

## 1.2 Motivation

Many techniques have been proposed to reduce the energy consumption of data centers. These techniques suggest better control of power distribution systems [14], efficient cooling systems [15], optimized computer hardware [16], virtualization technology [17], and load balancing mechanisms [18, 19]. It is known that turning off some unused machines by intelligently allocating the workload (VMs) to the smallest number of physical machines (known as VM consolidation) is an effective approach to reduce energy cost of a data center. For example, turning off a single x86 server from a data center can save approximately \$400 per annum [20]. This is because an idle machine (running with no load) consumes 60-70% of its peak-load power consumption [18, 19, 21]. Researchers are trying to leverage this fact to save energy by optimizing virtual machine (VM) assignment, an NP-hard problem [22, 23, 24, 13].

## 1.3 Research Objective

The aim of this work is to investigate the Virtual Machine Assignment Problem for large data centers. This includes a design of an efficient algorithm that minimizes the number of active PMs required for the given set of VMs in a short amount of time and helps to reduce operational cost, including power and maintenance.

## 1.4 Constraints and Complexity

VMs that run on the same physical machine (PM), share physical resources. Utilizing physical machines beyond a certain limit can cause significant performance degradation. Therefore, it is necessary to guarantee that, while minimizing the total number of PMs used, no PM gets utilized beyond a certain percentage of its maximum capacity. This is ensured by an appropriate upper limit on maximum utilization of PM resources. Due to multidimensional nature of VM requests, the assignment problem is very challenging. Each VM has its own CPU, memory, and bandwidth requirements. Similarly, every PM has a fixed capacity across each of these dimensions. Hence VM to PM assignment should minimize the total number of PMs used without violating these capacity constraints. VM assignment is formulated often as a *vector bin packing problem* (VBPP) [25], where the VMs that are treated as items ( $I$ ) are packed into PMs that are treated as bins ( $B$ ). Note that in rest of the manuscript, terms virtual machine and item, as well as physical machine and bin will be used interchangeably. The computational complexity of VBPP is  $O(B^I)$  [25]. Clearly, it is impractical to enumerate all possible solutions for a large number of VMs (items). Even the one-dimensional version of this problem is NP-hard.

## 1.5 Thesis Contributions

The contributions of this work can be summarized as follows:

- Simulated Evolution (SimE) based heuristic is proposed that efficiently finds a near-optimal solution in a reasonably short amount of time.
- Its performance is compared with that of another well known iterative heuristic, Simulated Annealing (SA), and with four popular constructive algorithms. Two of them are the improved versions of First Fit Decreasing algorithm referred to here as **FFD<sub>imp</sub>** and Least Loaded algorithm referred to here as **LL<sub>imp</sub>**, both proposed by Ajiro *et al.* [25]. And the other two are geometric heuristics that out-performed others, referred to here as **Norm Based-FFD** (FFD<sub>NB</sub>) and **Dot Product-Based FFD** (FFD<sub>DP</sub>), both of which are implemented in Microsoft's Virtual Machine Manager [26, 27].
- A method to generate a data set for vector bin packing problem (VBPP) covering a variety of parameters that can potentially affect the difficulty of the problem.
- A new lower bound method is also proposed for the problem instances where optimal solution contains few items per bin.

## 1.6 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 explores the literature related to virtual machine assignment problem and different heuristics approaches. Mathematical formulation of the problem is also presented in this chapter. Chapter 3 describes the Simulated Evolution-based proposed approach in detail. Chapter 4 provides some evaluations of the proposed methodology as



compared to other heuristics. Chapter 5 explains the proposed method to generate data set for multidimensional vector bin packing problem. In addition to this, a new lower bound procedure is explained with example. Simulation results using this new data set are reported in Chapter 6. Finally, Chapter 7 concludes this work and suggests future work.

## CHAPTER 2

# LITERATURE REVIEW

In this chapter, a brief overview of the literature, related to vector bin packing problem (VBPP) in general and virtual machine assignment problem in particular, is discussed. Packing of items with different sizes into given space is one of the fundamental problem in combinatorial optimization [28]. Several variants of this problem have been studied till today. VBPP is one of its popular variant that is often used to formulate problems that deals with the management of resources in a shared hosting environment [13]. Examples of VBPP applications include but are not limited to the following fields: computer network design [29], computer science (assignment problems, e.g., virtual machine placement (VMP) in a data center [13], assignment of jobs to processors, file placement for a multi-device storage system [30]), layout design [31], robot selection and workstation assignment [32], production planning and logistics (packing problems) [33], steel industry [34], etc. It is due to the fact that VBPP efficiently models the scheduling or assignment requirements that arise in different disciplines. Recently, VBPP

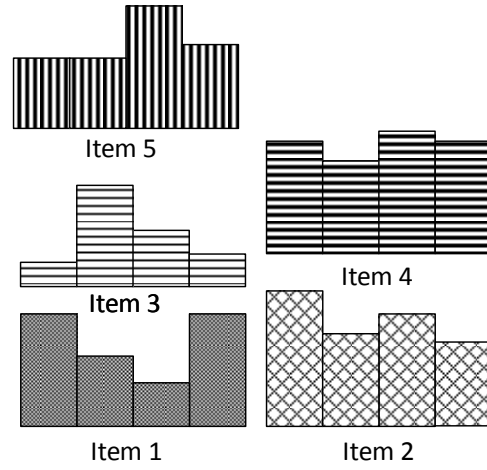
is gaining popularity among researchers' community who are working on virtual machine (VM) assignment problem in data centers.

## 2.1 Vector Bin Packing Problem (VBPP)

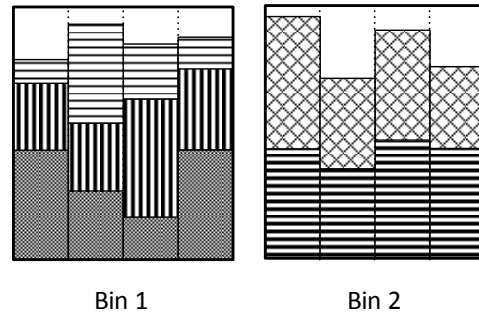
Vector Bin Packing Problem (VBPP) is a generalization of well known one-dimensional Bin Packing Problem (1R-BPP). Garey & Graham [35] discussed this problem in great detail, where we have a set,  $I$  of  $n$  items and each item  $I_i$ ,  $i \in \{1, 2, 3, \dots, n\}$  is defined as a  $D$ -dimensional requirement vector. These items have to be packed into a minimum number of bins with a fixed resource capacity in each of these  $D$ -dimensions. Fig. 2.1 illustrates the packing of five 4-dimensional items into two bins. VBPP is an NP-hard problem. Even its one-dimensional version which reduces to 1D-BPP is NP-Hard [35].

### 2.1.1 Literature Review

Over the past few decades, different exact methods, lower bounds, approximation methods and their worst case analysis for the 2-D version of VBPP have been discussed in literature [30, 36, 37, 38, 39]. Even for two dimensions, i.e., when  $D = 2$ , VBPP is APX-hard, which means that no asymptotic polynomial time approximation scheme exist unless  $P = NP$  [13]. For  $D$ -dimensional VBPP, Yao [40] showed that any algorithm that runs in  $O(n \log n)$ -time cannot give a solution with number of bins lesser than  $D$  times the optimum value. Fernandez de la Vega and Lueker proved that for VBPP, a linear time algorithm can find a



(a) Five items with 4-D resource requirement to be assigned.



(b) Assignment of five items in two bins.

Figure 2.1: Example of MDVBPP for  $R = 4$ .

solution with  $D + \epsilon$  times the number of bins in the optimal, where  $\epsilon$  is a positive constant [41]. Chekuri and Khanna [42] described an approximation algorithm that further improved this worst case performance ratio to  $(1 + \epsilon.R + O(\ln \epsilon^{-1}))$ .

### **Exact Methods for VBPP**

Spieksma proposed a lower bound (LB) method for two dimensional VBPP. This LB method is equivalent to find a maximal clique of a 2-threshold graph [36]. He incorporated this LB method in his proposed branch-and-bound algorithm. B.T. Han *et al.* presented exact algorithm and heuristics for a variant of VBPP, where the bins are not identical [30]. Caprara *et al.* discussed the 2-dimensional VBPP in detail and proposed heuristics, improved lower bounding techniques and exact algorithm [38]. Many of these techniques and exact algorithms are tailored to the 2-dimensional case and results are presented for instances with small number of items.

## **2.2 Virtual Machine Assignment Problem**

In a VM consolidation problem, the goal is to place the VMs in the minimum number of possible PMs without performance degradation. And, turn off the remaining PMs to save power. When mutiple VMs are placed on a single PM, the host operating system or hypervisor may consume some extra resources, e.g., for resource scheduling, or context switching [43]. To avoid performance degradation of PMs, we set an upper-bound on the maximum utilization of any resource of a

PM with some threshold value. This threshold value can be specified by the data center managers in terms of percentages.

In the context of vector bin-packing problems, VMs may be understood as items that are packed into PMs that are considered bins. The size of each item, i.e., VM, is defined by its resource demand vector. Dimensions of these resources may include CPU, memory, bandwidth, disk space etc., as described in Section 1.4. Capacity of each bin, i.e., PM, is bounded by the selected utilization threshold of its resources. Also note that placing multiple VMs on the same PM has an additive effect on the PM's utilization across each dimension [23]. For example, if (25%, 35%) is a pair of the CPU and memory utilization of a VM, and (20%, 40%) is of another VM, then, the utilization of a PM accommodating these two is (45%, 75%), i.e., the vector sum.

### 2.2.1 Optimization Formulation

In this work, we consider two dimensions of resources, CPU and memory. Suppose that there are  $n$  VMs to be assigned. Each VM  $v_i$ ,  $i \in \{1, 2, 3, \dots, n\}$  is defined as a 2-dimensional requirement vector,  $v_i = \{v_i^c, v_i^m\}$  where each dimension represents a normalized value of one type of resource requested (CPU and memory). These VMs are to be assigned to  $q$  PMs. Let  $T_k^c$  and  $T_k^m$  be the threshold values of CPU and memory resources, associated with each PM  $p_k$ ,  $k \in \{1, 2, 3, \dots, q\}$  respectively.

The assignment solution is represented by a  $q \times n$  matrix  $A$ , where

$$A_{ki} = \begin{cases} 1 & \text{if VM}_i \text{ is assigned to PM}_k \\ 0 & \text{otherwise} \end{cases}$$

Let  $f(p_k)$  be a function such that  $f(p_k) = 1$ , if PM  $p_k$  is loaded with at least one VM and  $f(p_k) = 0$  otherwise. The problem of assigning all VMs to the least number of PMs, and subject to the constraints, can be formulated as follows:

$$\text{minimize } \sum_{k=1}^q f(p_k) \quad (2.1)$$

$$\text{subject to } \sum_{i=1}^n A_{ki} \cdot v_i^c \leq T_k^c \quad \forall k \in \{1, 2, 3, \dots, q\} \quad (2.2)$$

$$\sum_{i=1}^n A_{ki} \cdot v_i^m \leq T_k^m \quad \forall k \in \{1, 2, 3, \dots, q\} \quad (2.3)$$

$$\sum_{k=1}^q A_{ki} \leq 1 \quad \forall i \in \{1, 2, 3, \dots, n\} \quad (2.4)$$

Constraints (2.2) and (2.3) impose threshold limit on maximum utilization while constraint (2.4) ensures that each VM will be assigned to only one PM.

### 2.2.2 Literature Review

In 2012 Uptime Institute Data Center Industry Survey, it is reported that over 60% of data center operators will consolidate their workloads (encapsulated in virtual machines) rather than adding additional server resources to existing data center facilities [44]. VM consolidation technique is increasingly becoming attractive for

large data centers due to its benefits such as reduction in hardware and operating costs as much as 50% and in energy cost as much as 80%. As a result, a good amount of research work has been done in this area. Both greedy deterministic and iterative non-deterministic approaches have been discussed in literature. A brief overview is given below.

### **Deterministic Heuristics**

In the category of deterministic heuristics, Doddvula *et al.* proposed a Magnitude Classified algorithm for server consolidation. Their algorithm first classifies the workload based on their resource requirements and then places the complementary workloads [45]. Other works such as that by Jhawar *et al.* [43] and Shi *et al.* [46] also consider security constraints along with maximizing utilization. First Fit Decreasing (FFD) and its variants are perhaps the most common deterministic methods applied to find an approximate solution to the vector bin-packing problem [22, 13]. FFD algorithm first sorts the VMs (items) in decreasing order of their sizes and then places them in PMs (bins) according to First Fit (FF) strategy. In the context of data center energy optimization, Lei Shi *et al.* presented and evaluated the performance of six different FFD-based VBP algorithms [22]. Ajiro *et al.* suggested improvements to the classical FFD and least loaded (LL) algorithms [25]. There are two differences between LL and FFD. First, LL restarts the placement process each time a new PM is added, when it has failed to place a VM into currently active PMs. Whereas FFD continues to place the subsequent VMs until all are placed. Second, in order to place a VM into a least-loaded active



PM, LL sorts active PMs in ascending order of their current utilizations each time before placing it. Improved FFD ( $\text{FFD}_{\text{imp}}$ ) is different from conventional FFD in the sense that it seeks near-optimal solution in multiple passes. In the first pass VMs are sorted in decreasing order of their highest resource demand, i.e., sum of all CPU demands or sum of all memory demands, whichever is larger. Then an attempt is made to pack all the VMs in number of PMs equal to the theoretical lower bound. If any VM cannot be packed, then it is moved to a priority queue and placement process is aborted. In the next pass, VMs in the priority queue are placed first, followed by the remaining VMs in the sorted list. The above steps are repeated MAXR times. If all VMs are not packed in MAXR iterations then the number of destination PMs is incremented by one and the above process is repeated until a solution is found, i.e., all VMs are packed.  $\text{LL}_{\text{imp}}$  is implemented in multi-passes in a similar way employing the LL heuristic. These improved versions provide better quality solutions than that of their single-pass implementations. This improvement, however, comes at the expense of increased run time [25].

In literature several extensions of simple greedy heuristics have been proposed such as First Fit Decreasing (FFD) [22], Best Fit Decreasing [47], Best-Fit, Worse-Fit, First-Fit [48]. Stillwell *et al.* [49] presented a performance comparison of several FFD-variants for VBPP. However, demands across different dimensions, in their synthetic problem instances were sampled independent and identically distributed, whereas in the real-world scenarios problem instances may have com-

plementary resource requirements across different dimensions, e.g., a VM request with scientific computations may have high CPU requirements but low I/O requirements while VMs that are acting as web servers would behave in the opposite way. FFD-variants that are not designed to take advantage of such complementary requests can be far from optimal [13]. Panigrahy *et al.* [13] systematically studied the family of FFD heuristics and their limitations and suggested new geometric based heuristics named **Norm Based-FFD** ( $\text{FFD}_{\text{NB}}$ ) and **Dot Product-Based FFD** ( $\text{FFD}_{\text{DP}}$ ) that are explained below.

#### **Dot Product-Based FFD** ( $\text{FFD}_{\text{DP}}$ )

This heuristic follows the bin-centric FFD approach (see Fig. 2.2), where size of each item is determined by the weighted dot product between the vector of remaining capacities of the current open bin and the vector of demands for the item. The weighted dot product is calculated using Equation 2.5.

$$\sum_d a_d \cdot I_i^d \cdot \text{RemCap}(b)_d \quad (2.5)$$

where  $\text{RemCap}(b)_d$  is the remaining capacity of bin  $b$  in  $d^{\text{th}}$  dimension, and  $a_d$  is the weight of  $d^{\text{th}}$  dimension that is calculated in the following manner:

$$a_d = e^{(0.01 * \frac{1}{n} \cdot \sum_{i=1}^n I_i^d)} \quad (2.6)$$

The item  $I_i$  that maximizes the dot product without violating the capacity constraint is considered to be placed first.

### **Norm-based FFD (FFD<sub>NB</sub>)**

This is another bin-centric heuristic that looks at the difference between the vectors  $I_i$  and the residual capacity  $RemCap(b)$  under a certain norm, instead of the dot product. For example, for the  $l_2$  norm, from all unassigned items, it places the item  $I_i$  that minimizes the quantity  $\sum_d a_d \cdot (I_i^d - RemCap(b)_d)^2$  and the assignment does not violate the capacity constraints. The weights  $a_d$  are again chosen as in Equation (2.6).

#### **Procedure**

**While** *there are items remaining to be placed* **Do**

*Open a new bin.*

**While** *some item fits in this bin* **Do**

*Place “largest” remaining item that fits in the bin.*

**EndWhile**

**EndWhile**

Figure 2.2: Bin-centric First Fit Decreasing (FFD) Procedure.

### **2.2.3 Iterative Heuristics**

Recently, in the category of non-deterministic metaheuristics, genetic algorithm [24], particle swarm optimization [50], and ant-colony system algorithm [23] have been attempted for optimizing the VM placement problem. However these are implemented for the one or two dimensional problem. There are very few metaheuristic implementations available for  $R \geq 2$ . For  $R \geq 2$ , Stillwell *et*

*al.* [49] showed that FFD-based heuristics perform better than genetic algorithm. Jing Xu *et al.* proposed a modified genetic algorithm to simultaneously minimize the total power consumption, resource wastage and thermal dissipation using a fuzzy multi-objective cost function [24]. Breitgand *et al.* modeled the problem as a stochastic bin-packing problem using statistical multiplexing of physical resources [51]. Gao *et al.* presented VMPACS, a modified ant colony optimization algorithm which gives a Pareto set (non-dominated solutions) that minimizes total resource wastage and power consumption [23]. Recently, for large-scale machine reassignment and packing problems that have multiple resources, Masson *et al.* proposed a Multi-Start Iterated Local Search for Packing Problems (MS-ILS-PP) [52]. Karmer *et al.* used the concept of dynamic voltage/frequency scaling (DVFS) along with VM consolidation to further improve the energy efficiency. Their approach is based on trade-off between power and performance [53].

In this work we engineer an evolutionary nondeterministic optimization heuristic known as Simulated Evolution (SimE). Similar to other nondeterministic algorithms, SimE also possesses hill-climbing capability. One key requirement of SimE is to define an appropriate way to estimate the *goodness* of the current assignment of a movable element, in our case the movable elements are VMs. The process of evolution, guided by goodness value, tends to converge reasonably fast to a good quality solution. Many other nondeterministic heuristics, such as Simulated Annealing, tabu search, etc., lack this domain knowledge feature and work mostly with random moves. Further details of goodness function are in Chapter 3.

## CHAPTER 3

# PROPOSED APPROACH

This section explains our proposed VM consolidation approach that is based on an iterative heuristic Simulated Evolution (SimE). We briefly discuss basic steps of SimE.

### 3.1 Simulated Evolution

Back in 1987, Simulated Evolution (SimE) was proposed by Kling and Banerjee [54]. The algorithm combines constructive perturbation and iterative improvement and refrain itself from getting stuck to the local minima by following a stochastic approach. In SimE, the search space is traversed by making intelligent moves, unlike in other nondeterministic algorithms such as Simulated Annealing (SA), where random moves are made. The core of the algorithm is the goodness estimator. SimE assigns each moveable element a goodness value. The goodness value indicates how well a certain movable element is currently assigned. The more the goodness value, the lesser is the probability of the element being selected for

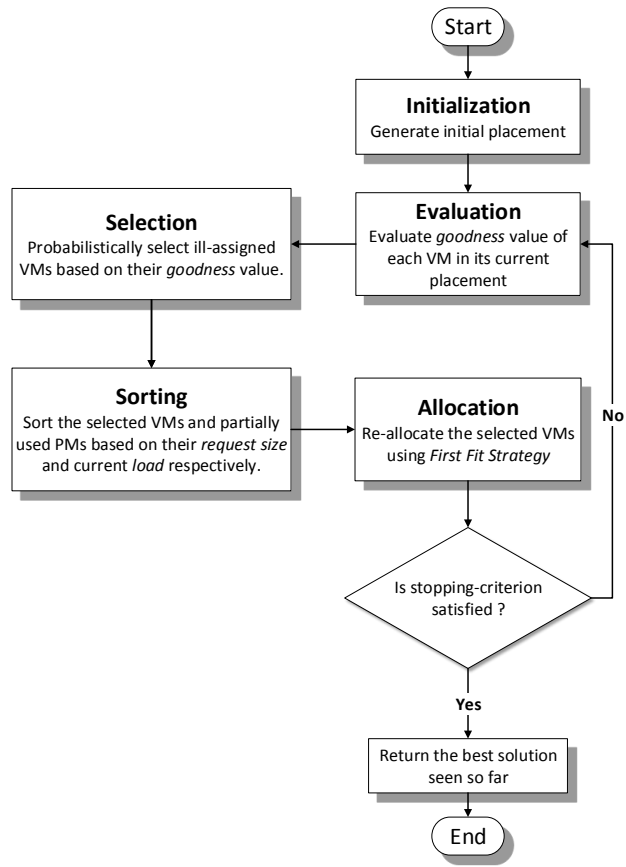


Figure 3.1: Flowchart of SimE.

reallocation.

The flow of our proposed (SimE) algorithm is shown in Fig. 3.1. SimE starts with an initial solution  $\Phi$  of a set  $V$  containing  $n$  movable elements (VMs). It, then follows an evolution-based approach to find better solutions from one iteration to the next by perturbing some ill-assigned elements (VMs) while retaining the near-optimal ones. The algorithm consists of three sequential steps, **evaluation**, **selection** and **allocation** that are executed in each iteration. The process of iterative improvements continues until the solution average goodness value reaches at its maximum, or no considerable improvement in solution quality is observed after a given number of iterations [55].

### 3.1.1 Goodness Evaluation

This step involves the evaluation of goodness (fitness)  $g_i$  of each VM  $v_i$  assigned to PM  $p_k$  in current solution  $\Phi'$ . Effective goodness measures can be thought of based on the domain knowledge of the optimization problem [56]. This goodness measure is expressed as a single number in the range of zero to one. For our VM assignment problem we define the goodness measure as:

$$g_i = \frac{v_i^c + v_i^m}{p_k^c + p_k^m} \quad (3.1)$$

where  $v_i^c$  and  $v_i^m$  are CPU and memory requirements of VM  $v_i$ , and  $p_k^c$  and  $p_k^m$  are the available CPU and memory resources of partially used PM  $p_k$  after removing VM  $v_i$  from PM  $p_k$  in the current solution  $\Phi'$ . Equation (3.1) assumes a minimiza-

tion of resource wastage in PM  $p_k$  (maximization of goodness). The goodness of a VM  $v_i$  will be 1 if it is assigned to such a partially used PM  $p_k$  that  $v_i^c = p_k^c$  and  $v_i^m = p_k^m$ . It means that the current assignment of VM  $v_i$  exactly packs the PM  $p_k$  and hence optimally utilizes the PM  $p_k$ . For example, the goodness values of VMs  $v_1$ ,  $v_2$  and  $v_3$  (in Fig. 3.2) are 1 as their combined placements optimally utilize the resources of PM  $p_1$ . On the other hand, the goodness  $g_i$  will be near 0, when a VM  $v_i$ , with a very small resource requirements, is placed in an empty PM  $p_k$  i.e.,  $v_i^c \ll p_k^c$  and  $v_i^m \ll p_k^m$ . Such an assignment will result in maximum resource wastage. The VM  $v_6$ , in (Fig. 3.2), has approximately zero goodness value. Note that this goodness estimation is strongly reflects the target objective of the given problem. The quality of a solution can also be estimated by summing up the goodness of all of its constituent elements (VMs).

The goodness measure given in Equation (3.1) can be generalized for  $D$ -dimensional case to incorporate requirements in other dimensions, e.g., I/O, storage, etc. A generalized version is shown in Equation 3.2:

$$g_i = \frac{v_i^{d_1} + v_i^{d_2} + \dots v_i^{d_D}}{p_k^{d_1} + p_k^{d_2} + \dots p_k^{d_D}} \quad (3.2)$$

### 3.1.2 Selection

In this step, elements are selected for relocation probabilistically. Elements with lesser *goodness* values have more probabilities of getting selected. This step divides  $\Phi'$  into two disjoint sets; a set  $V_s$  of selected elements and a partial solution  $\Phi_p$



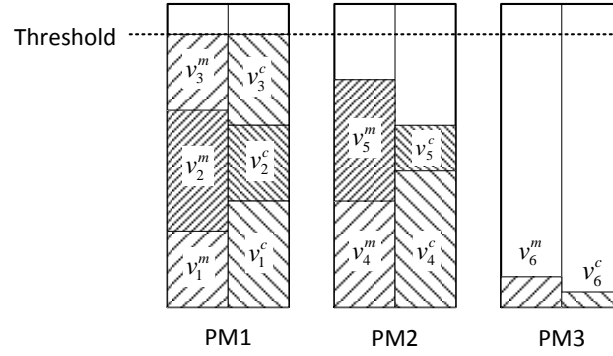


Figure 3.2: Allocation of 6 VMs on 3 PMs.

containing rest of the elements of the solution  $\Phi'$ . Every element of the solution is considered separately from all other elements. The decision of selecting an element  $v_i$  to the set  $V_s$  depends on its goodness  $g_i$ . The selection operator has a nondeterministic nature, i.e., an individual with a high goodness (close to one) still has a non-zero probability of being assigned to the selection set  $V_s$ . It is this element of nondeterminism that makes SimE capable of escaping local minima. Each time a VM  $v_i$  is considered for selection a random number is generated. The inequality  $Random \leq (1 - g_i + B)$  is used for this purpose (see Fig. 3.3). Error in goodness estimation is compensated by using a selection *bias* ( $B$ ). The objective of this bias value is to deflate or inflate the goodness of elements. A high positive value of bias increases the probability of selection while the negative value has the opposite effect. Large selection sets may lead to better solution, but will require higher run time. On the other hand, small selection sets will speed-up the algorithm, but with the risk of an early convergence to a sub-optimal solution (local minima). Values of  $B$  are recommended to be in the range  $[-0.2, 0.2]$ . In

### ALGORITHM

```
Simulated_Evolution(V, Stopping – criteria);
/*  $\Phi_i$ : Initial Solution; */
/*  $\Phi_p$ : Partial Solution; */
/*  $\Phi'$ : New Solution; */
/*  $V$ : Set of all VMs, where  $|V| = n$ ; */
/*  $V_s$ : Selected VMs for reallocation; */
/*  $P_a$ : Active PMs in  $\Phi_p$ ; */
/*  $B$ : Selection bias; */
/* maxSelection: Upper limit of the selection set size; */
INITIALIZATION:
     $\Phi_i = \text{initial\_placement}(V)$ ;
     $\Phi' = \Phi_i$ ;
Repeat
EVALUATION:
    ForEach  $v_i \in V$  Do
         $g_i = \text{Evaluate}(v_i)$ ; /* Evaluate using goodnes estimator (Equation 3.1) */
    EndForEach;
SELECTION:
     $\Phi_p = \Phi'$ ;
    counter = 0;
    ForEach  $v_i \in V$  Do
        If ( $\text{Random} \leq (1 - g_i + B)$ )  $\wedge$  (counter  $\leq$  maxSelection) Then
             $V_s = V_s \cup \{v_i\}$ ;
             $\Phi_p = \Phi_p - \{v_i\}$ ;
            counter = counter + 1;
        EndIf;
    EndForEach;

ALLOCATION:
    Sort the VMs in set  $V_s$  based on their resource demand (Equation (3.3));
    Sort the active PMs  $P_a$  based on their current load (Equation (3.4));
    ForEach  $v_i \in V_s$  Do
         $\text{Allocate}(v_i, \Phi_p)$ ; /* Allocate  $v_i$  in  $\Phi_p$ , using First Fit Strategy */
    EndForEach;
     $\Phi' = \Phi_p$ ;
Until Stopping-criterion is satisfied;
Return (BestSolution);
End Simulated_Evolution.
```

Figure 3.3: Simulated Evolution Algorithm for VM assignment.

many cases a value of  $B = 0$  would be a reasonable choice as in our case [55]. In addition to bias value, *maxSelection* also provides control over the selection process and restricts the maximum size of the selection set. In this work, a value of 40% of the total number of items was adopted. This keeps the time requirement of the SimE algorithm under control, especially during the allocation step, which is the most time consuming step of the algorithm.

### 3.1.3 Allocation

The allocation step has great influence on solution quality. Allocation takes the elements of set  $V_s$  and the partial solution  $\Phi_p$  and generates a complete new solution  $\Phi'$  with the elements of set  $V_s$  mutated according to allocation strategy. The goal of *Allocation* strategy is to favor improvements over the previous iteration, without being too greedy [55]. Superior alterations gradually improve the individual goodness values as the goodness of each individual element also reflects the target objective. Hence, *Allocation* helps the search to progressively converge towards a target configuration where every individual is optimally located.

The design of allocation strategy is problem specific. Just like in the design of goodness function, the choice of allocation strategy also requires ingenuity on the part of the designer. In this work we adopted a variant of FFD heuristic as our allocation strategy. The VMs selected during the *selection* step are sorted in decreasing order of their request sizes ( $Rv_i$ ) computed using Equation (3.3).

$$Rv_i = (v_i^c)^2 + (v_i^m)^2 \quad (3.3)$$

The active PMs in partial solution  $\Phi_p$  are also sorted in decreasing order of the linear sum of their occupied resources ( $Op_k$ ) computed using Equation (3.4):

$$Op_k = (1 - p_k^c) + (1 - p_k^m) \quad (3.4)$$

Subsequently, First Fit algorithm is applied to generate the new solution  $\Phi'$ .

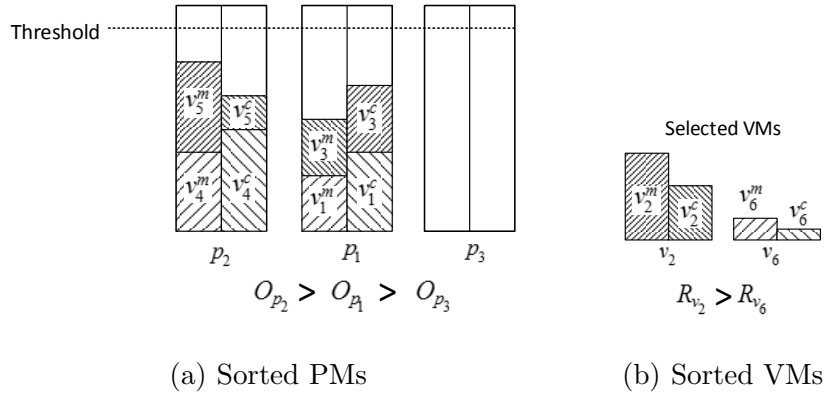


Figure 3.4: Sorted PMs and VMs for allocation.

To illustrate this, consider the placement solution in Fig. 3.2. Suppose that  $v_2$  and  $v_6$  are selected for reallocation. These VMs are sorted according to their size using Equation (3.3) and PMs are sorted according to their utilized size using Equation (3.4). This is illustrated in Fig. 3.4. In the next step, the first fit algorithm will first attempt to place VM  $v_2$  in PM  $p_2$ . Since the remaining capacity of  $p_2$  is not sufficient to accommodate  $v_2$ , it will be placed in next PM, that is  $p_1$ . Next  $v_6$  will be attempted & successfully placed in  $p_2$ . This resulting

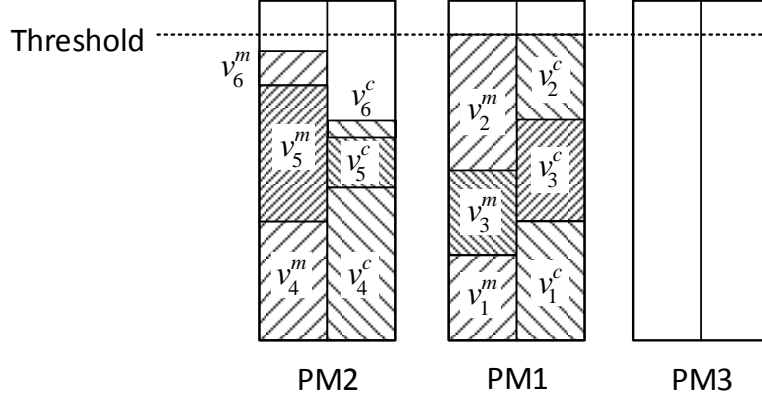


Figure 3.5: Allocation of selected VMs.

solution is shown in Fig. 3.5. This new solution is better than the previous one as it requires one less PM, and it thereby results in energy saving.

Initial placement  $\Phi_i$  is also obtained by this same *allocation* strategy but with the difference that all the VMs are treated as *selected* and are placed in an empty set of PMs.

## 3.2 Complexity Analysis

Our proposed SimE-based algorithm consists of four steps in a loop as illustrated in the flowchart in Fig. 3.1. The evaluation step computes *goodness* value of all  $n$  VMs using Equation (3.1). This takes  $O(n)$  time. The selection step probabilistically selects ill-assigned VMs and this also takes  $O(n)$  time. In the sorting step prior to allocation, both the lists of selected VMs and active PMs are sorted and this takes  $O(n \log n)$  time. In the allocation step, First Fit (FF) algorithm sequentially checks if all selected VMs can be packed into one of  $q$  current active

PMs. FF then packs each selected VM into a PM first found to be able to accommodate it. If a VM cannot be packed into any current active PM, the  $(q + 1)$ -th PM is turned ON to accommodate it. The complexity of this step is  $O(n^2)$ . The overall complexity of our algorithm is  $O(I.n^2)$ , where  $I$  is the number of iterations. Experiments have indicated that  $I$  remains fairly constant as  $n$  increases, e.g.,  $I$  varies in the range of  $65 - 75$  when  $n$  is increased from 200 to 1000. Hence the complexity of the algorithm reduces to  $O(n^2)$ .

# CHAPTER 4

## PERFORMANCE EVALUATION

In this section we provide performance evaluation of our proposed approach with respect to solution quality and run time. First, we compare it with the improved versions of classic FFD ( $\text{FFD}_{\text{imp}}$ ) and LL ( $\text{LL}_{\text{imp}}$ ) algorithms proposed by Ajiro *et al.* [25], and a well-known iterative heuristic, Simulated Annealing (SA) [55]. Then we discuss the solution quality, performance, and scaling of the SimE heuristic.

### 4.1 Simulation Setup

The programs for the proposed SimE algorithm, SA,  $\text{FFD}_{\text{imp}}$  and  $\text{LL}_{\text{imp}}$  heuristics were coded in MATLAB and run on an Intel *core<sup>TM</sup>i5* with 1.80 GHz CPU and 4 GB RAM.  $\text{FFD}_{\text{imp}}$  and  $\text{LL}_{\text{imp}}$  try to pack all the VMs in number of PMs equal to the theoretical lower bound (LB). If that can not be achieved then a different packing sequence based on reordered VMs (as explained in Section 2.2.2) is at-

tempted before a new PM is turned ON. This is done upto a maximum of  $MAXR$  times, where  $MAXR$  is a control parameter that provides trade-off between quality of solution and time. Details of implementation and experiments to determine the appropriate value of  $MAXR$  are as discussed by Ajiro *et al.* According to the experimental study,  $MAXR$  for  $FFD_{imp}$  is set to 10-30% and for  $LL_{imp}$  equal to 10% of total number of VMs [25].

The details of Simulated Annealing (SA) algorithm can be found in [55]. Its four important parameters have to be tuned very carefully. These are: initial temperature  $T_0$ , cooling rate  $\alpha$ , constant  $\beta$ , and  $M$  which represents the time until the next parameter update. After trial runs, appropriate values of these parameters were found to be  $T_0 = 36$ ,  $\alpha = 0.9$ ,  $\beta = 1.1$ ,  $M = 3$  for placement of 200 VMs. For 500 VMs, the best values of the parameters used were  $T_0 = 120$ ,  $\alpha = 0.9$ ,  $\beta = 1.09$ ,  $M = 2.2$ . The quality of solution obtained by SimE improves when the number of iterations is increased. The improvement is quite steep in the early iterations. It gets less steeper in later iterations until it becomes almost insignificant. The number of required iterations can be easily tuned after a few initial experimental runs [55]. In this work SimE algorithm was set to stop exploring the search space if 75 consecutive iterations fail to improve the solution. For SimE, *Bias* value was set to 0, and maximum size (*maxSelection*) of selection set was restricted to 40% of total VMs (reasons are discussed in Section 3.1.2).



### 4.1.1 Work Load

The problem instances were a set of two-resource demand vectors representing the CPU and memory utilization of 200 and 500 VMs. PMs were assumed to be identical, that is, all PMs have the same resource capacity fixed at 90% although the proposed approach is equally applicable for the heterogeneous case. Due to nondeterministic behaviour, average of results obtained from 100 independent runs are reported. In combinatorial problems, hardness is defined according to a worst-case scenario. However, in practical applications engineers invariably are more interested in typical instances of an optimization task rather than looking for the hardest possible instances. For this reason, suitably parametrized random ensembles of instances of problems are introduced. In this context, it was observed that in some regions of the ensemble space instances are typically easy to solve, while in other regions instances are found to be typically hard. This change in behaviour resembles the phase transitions observed in physical systems [57]. Two properties that determine the phase transition in our case are: (a) the correlation between the resource demand vectors (that is between CPU and memory sizes); and (b) average size of the VM resource demand in the data set. If the CPU and memory utilization demands are almost equal for all VMs, then this is a special case of strong-positive correlation. In this case the instance is similar to a one-dimensional bin-packing problem that is relatively easier to solve than the two-dimensional problem typically is [25]. However, for negatively correlated instances, more effort is required to find good solutions [13]. High negative cor-

relation between CPU and memory requirements or high average size of the VM resource demands makes it difficult to find a solution near the theoretical lower bound. Such solutions either take more time, or for the same time the quality of results obtained is lower than those where the average size of the VM resource demands is small. To make synthetic instances more representative and cover a wide range of possible workloads, problem instances are generated with two different average resource values and several correlations of CPU and memory utilizations, employing the method proposed by Ajiro *et al.* [25]. The pseudo code for this is given in Fig. 4.1. In Fig. 4.1,  $\text{rand}(1)$  is a function that returns uniformly

```

For  $i = 1$  To  $n$  Do
     $v_i^c = \text{rand}(2\overline{v^c})$ 
     $v_i^m = \text{rand}(\overline{v^m})$ 
     $r = \text{rand}(1)$ 
    If  $(r < P \wedge v_i^c \geq \overline{v^c}) \vee (r \geq P \wedge v_i^c < \overline{v^c})$  Then
         $v_i^m = v_i^m + \overline{v^m}$ 
    EndIf
EndFor;
```

Figure 4.1: Pseudo code to generate random problem instances with certain correlations.

distributed random real numbers in the range  $[0, 1)$ ;  $\overline{v^c}$  denotes the average CPU utilization while  $\overline{v^m}$  represents the average memory utilization. The probability  $P$  is used to decide whether both the utilization of CPU and memory would be equal to or higher than the average values, or both utilizations would be lesser than the average values. By varying this probability  $P$ , one can control the correlations of CPU and memory utilizations to some extent.

In this experiment, I used two kinds of average values and five different

probabilities. Both  $\overline{v^c}$  and  $\overline{v^m}$  were set to 25%, and then to 45%. The distributions of CPU and memory utilizations were in the range of  $[0, 50\%)$  when  $\overline{v^c} = \overline{v^m} = 25\%$ , and  $[0, 90\%)$  when  $\overline{v^c} = \overline{v^m} = 45\%$ . For  $\overline{v^c}$  and  $\overline{v^m} = 25\%$ ,  $P$  was set equal to 0.00, 0.25, 0.50, 0.75, and 1.0, and for this the average correlation coefficients obtained are  $-0.7485$ ,  $-0.3813$ ,  $0.0081$ ,  $0.3736$ , and  $0.7493$  for each set of instances. These coefficients correspond to strong-negative, weak-negative, no, weak-positive, and strong-positive correlations. The same values of  $P$  were used for  $\overline{v^c}$  and  $\overline{v^m} = 45\%$  and then the correlation coefficients were  $-0.7508$ ,  $-0.3703$ ,  $0.0019$ ,  $0.3857$ , and  $0.7476$ . Threshold values of both utilizations were kept at  $T_k^c = T_k^m = 90\%$ ,  $k \in \{1, 2, 3, \dots, q\}$  throughout these experiments.

## 4.2 Results and Discussion

To evaluate the efficiency of the proposed SimE algorithm, its performance is compared to that of FFD<sub>imp</sub>, LL<sub>imp</sub>, and SA. The comparison metrics are the number of active PMs ( $q$ ) and time to find the solution. The value  $q/LB$  represents consolidation ratio calculated as ratio of active PMs  $q$  to the theoretical lower bound  $LB$  which is estimated using Equation (4.1). A value  $q/LB$  closer to 1.0 represents higher efficiency. Table 4.1 lists the average number of active PMs obtained by these algorithms for different correlation and reference mean values.

$$LB = \max \left( \left\lceil \sum_{i=1}^n \frac{v_i^c}{T^c} \right\rceil, \left\lceil \sum_{i=1}^n \frac{v_i^m}{T^m} \right\rceil \right) \quad (4.1)$$

From Table 4.1 we note the following:

- The timing performance of both  $FFD_{imp}$  and  $LL_{imp}$  strongly depends on the correlation between CPU and memory utilizations. They take more time to reach a good solution for the instances with negative correlation (see Fig. 4.2). On the other hand, execution time of both SA and SimE only slightly varies across different correlations.

| Corr                     | Algorithm  | MAXR            | VMs = 200 |        |           | VMs = 500 |        |           |
|--------------------------|------------|-----------------|-----------|--------|-----------|-----------|--------|-----------|
|                          |            |                 | q         | q/LB   | Time(Sec) | q         | q/LB   | Time(Sec) |
| $\frac{v^c}{v^m} = 25\%$ | Strong -ve | $FFD_{imp}$ 20% | 68.67     | 1.1884 | 10.5987   | 171.95    | 1.2091 | 568.4928  |
|                          |            | $FFD_{imp}$ 30% | 65.68     | 1.1366 | 11.5814   | 163.90    | 1.1524 | 601.2377  |
|                          |            | $LL_{imp}$ 10%  | 63.47     | 1.0982 | 2.4263    | 156.15    | 1.0978 | 68.5390   |
|                          |            | SimE -          | 59.22     | 1.0246 | 1.1137    | 145.00    | 1.0193 | 8.8654    |
|                          |            | SA -            | 70.29     | 1.2173 | 7.7961    | 182.45    | 1.2793 | 122.7781  |
|                          | Weak -ve   | $FFD_{imp}$ 20% | 65.79     | 1.1411 | 8.0819    | 162.85    | 1.1489 | 385.2580  |
|                          |            | $FFD_{imp}$ 30% | 64.11     | 1.1118 | 9.5391    | 159.35    | 1.1242 | 471.4340  |
|                          |            | $LL_{imp}$ 10%  | 62.24     | 1.0789 | 1.9123    | 152.25    | 1.0736 | 47.0452   |
|                          |            | SimE -          | 58.86     | 1.0203 | 0.9870    | 144.35    | 1.018  | 7.0985    |
|                          |            | SA -            | 69.89     | 1.2153 | 7.7669    | 180.55    | 1.2664 | 126.7085  |
|                          | Zero       | $FFD_{imp}$ 20% | 63.25     | 1.1122 | 6.5057    | 160.40    | 1.1364 | 365.4011  |
|                          |            | $FFD_{imp}$ 30% | 62.51     | 1.0991 | 8.3291    | 158.95    | 1.1262 | 486.3127  |
|                          |            | $LL_{imp}$ 10%  | 60.26     | 1.0594 | 1.4108    | 149.85    | 1.0615 | 35.8382   |
|                          |            | SimE -          | 57.89     | 1.0177 | 0.9166    | 143.40    | 1.016  | 5.9491    |
|                          |            | SA -            | 69.74     | 1.2179 | 7.7257    | 180.1     | 1.2717 | 131.3036  |
|                          | Weak +ve   | $FFD_{imp}$ 20% | 61.80     | 1.0849 | 5.0155    | 156.75    | 1.1004 | 307.2129  |
|                          |            | $FFD_{imp}$ 30% | 61.62     | 1.0817 | 6.7264    | 156.35    | 1.0976 | 416.3680  |
|                          |            | $LL_{imp}$ 10%  | 60.04     | 1.0535 | 1.1589    | 150.00    | 1.0529 | 30.6913   |
|                          |            | SimE -          | 57.84     | 1.0152 | 0.9330    | 144.30    | 1.013  | 6.8773    |
|                          |            | SA -            | 69.10     | 1.2098 | 7.6981    | 178.15    | 1.2596 | 126.4890  |
|                          | Strong +ve | $FFD_{imp}$ 20% | 60.44     | 1.0650 | 3.6218    | 149.70    | 1.0708 | 175.7809  |
|                          |            | $FFD_{imp}$ 30% | 60.40     | 1.0643 | 4.8976    | 149.70    | 1.0708 | 241.3745  |
|                          |            | $LL_{imp}$ 10%  | 59.53     | 1.0488 | 1.0285    | 146.50    | 1.0479 | 23.8151   |
|                          |            | SimE -          | 57.63     | 1.0155 | 0.8121    | 141.50    | 1.0122 | 5.3684    |
|                          |            | SA -            | 67.86     | 1.1917 | 7.6885    | 172.85    | 1.2432 | 118.1567  |

continued on next page

continued from previous page

|  | Corr       | Algorithm          | MAXR | VMs = 200 |        |           | VMs = 500 |        |           |
|--|------------|--------------------|------|-----------|--------|-----------|-----------|--------|-----------|
|  |            |                    |      | q         | q/LB   | Time(Sec) | q         | q/LB   | Time(Sec) |
| $\frac{v^c}{v^m} = \frac{v^m}{v^m} = 45\%$ | Strong -ve | FFD <sub>imp</sub> | 20%  | 124.62    | 1.2065 | 28.35     | 300.20    | 1.1777 | 1330.0000 |
|  |            | FFD <sub>imp</sub> | 30%  | 123.91    | 1.1996 | 39.4475   | 298.25    | 1.1701 | 1820.0000 |
|  |            | LL <sub>imp</sub>  | 10%  | 125.63    | 1.2163 | 8.6206    | 294.50    | 1.1553 | 464.8868  |
|  |            | SimE               | -    | 121.47    | 1.1759 | 1.418     | 286.25    | 1.1229 | 10.5846   |
|  |            | SA                 | -    | 130.98    | 1.2634 | 10.1118   | 329.05    | 1.2885 | 154.7883  |
|  | Weak -ve   | FFD <sub>imp</sub> | 20%  | 123.16    | 1.1865 | 26.4677   | 301.05    | 1.1774 | 1260.0000 |
|  |            | FFD <sub>imp</sub> | 30%  | 122.03    | 1.1756 | 35.6037   | 298.35    | 1.1669 | 1690.0000 |
|  |            | LL <sub>imp</sub>  | 10%  | 122.39    | 1.179  | 7.1789    | 293.50    | 1.1479 | 428.7857  |
|  |            | SimE               | -    | 118.86    | 1.1449 | 1.4415    | 286.50    | 1.1205 | 10.0072   |
|  |            | SA                 | -    | 128.23    | 1.2436 | 9.5241    | 326.50    | 1.2766 | 150.9985  |
|  | Zero       | FFD <sub>imp</sub> | 20%  | 119.14    | 1.1618 | 21.7631   | 293.90    | 1.1534 | 1060.0000 |
|  |            | FFD <sub>imp</sub> | 30%  | 118.31    | 1.1536 | 29.4344   | 291.40    | 1.1436 | 1420.0000 |
|  |            | LL <sub>imp</sub>  | 10%  | 118.82    | 1.1585 | 5.8913    | 287.45    | 1.1279 | 342.8841  |
|  |            | SimE               | -    | 114.94    | 1.1205 | 1.3314    | 278.60    | 1.093  | 9.2724    |
|  |            | SA                 | -    | 126.36    | 1.2316 | 9.2873    | 318.25    | 1.2639 | 144.1075  |
|  | Weak +ve   | FFD <sub>imp</sub> | 20%  | 116.65    | 1.1361 | 17.3948   | 286.40    | 1.1263 | 798.6678  |
|  |            | FFD <sub>imp</sub> | 30%  | 116.18    | 1.1316 | 24.1385   | 285.15    | 1.1214 | 1110.0000 |
|  |            | LL <sub>imp</sub>  | 10%  | 116.35    | 1.1331 | 4.8078    | 281.30    | 1.106  | 269.7920  |
|  |            | SimE               | -    | 113.10    | 1.1013 | 1.2424    | 273.35    | 1.0746 | 8.5355    |
|  |            | SA                 | -    | 123.68    | 1.2103 | 9.1476    | 313.65    | 1.2395 | 141.7936  |
|  | Strong +ve | FFD <sub>imp</sub> | 20%  | 112.74    | 1.1136 | 13.4092   | 278.65    | 1.1064 | 646.3755  |
|  |            | FFD <sub>imp</sub> | 30%  | 112.15    | 1.1077 | 18.5815   | 277.10    | 1.1002 | 885.2129  |
|  |            | LL <sub>imp</sub>  | 10%  | 110.69    | 1.0932 | 3.1699    | 270.45    | 1.0738 | 151.7008  |
|  |            | SimE               | -    | 108.10    | 1.0675 | 1.0909    | 264.60    | 1.0505 | 7.7824    |
|  |            | SA                 | -    | 119.59    | 1.1817 | 8.8983    | 303.25    | 1.2130 | 134.8037  |

Table 4.1: Performance comparison of FFD<sub>imp</sub>, LL<sub>imp</sub>, SA and SimE.

- For all algorithms applied, consolidation ratio decreases with change of correlation from strong-positive to strong-negative.
- Similarly, consolidation ratio decreases by decreasing average resource demand value from 45% to 25%.
- In each case our proposed algorithm SimE gives better consolidation efficiency in a shorter amount of time as compared to FFD<sub>imp</sub>, LL<sub>imp</sub>, and SA.

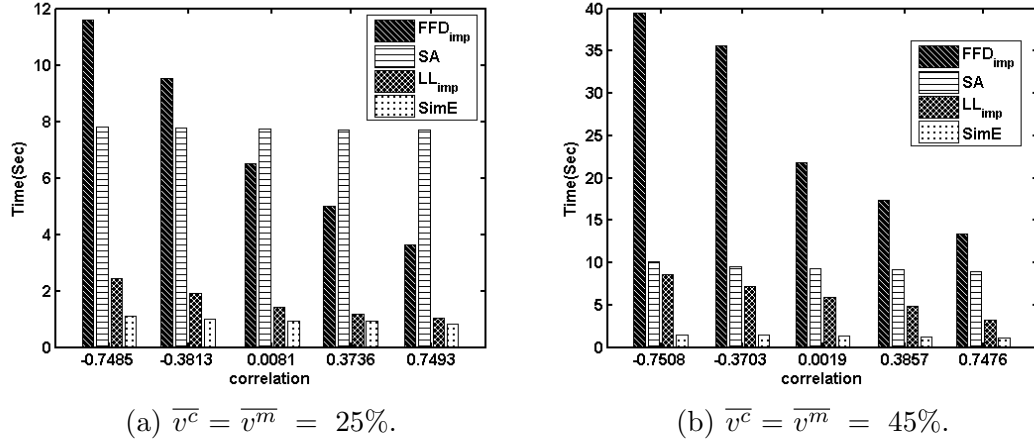


Figure 4.2: Run time of FFD<sub>imp</sub>, LL<sub>imp</sub>, SA and SimE with 200 VMs for cases of (a)  $\bar{v}^c = \bar{v}^m = 25\%$  and (b)  $\bar{v}^c = \bar{v}^m = 45\%$ .

SimE performs better than the two deterministic algorithms FFD<sub>imp</sub> and LL<sub>imp</sub> because these heuristics consider only one dimension, CPU or memory, the sum of whichever resource request is larger, when sorting them. The order obtained may not be suitable for optimal assignment. Therefore, the entire sorting and assignment steps are repeated several times, each time giving priority to VMs that have failed to be packed in currently active PMs. While proposed SimE only picks a small number of VMs with low goodness value and sorts them considering both dimensions of utilization. This precise selection of a small number of VMs and proper sorting plays a key role in improving the solution quality and reducing run time. Although SimE and SA both are iterative nondeterministic heuristics, but SimE is more intelligent, and thus requires fewer iterations to converge towards a desirable solution [55]. Change in cost of SimE and SA with iterations is illustrated in Fig. 4.3. It is clearly seen that SimE quickly finds a good solution through a few initial iterations. The plot of average goodness of the solution with

iteration is shown in Fig. 4.4. This graph reflects the behaviour of SimE. It can be observed that the average goodness increases with iterations. It validates that as the algorithm progresses, more and more VMs are approaching their respective near optimal assignments in the solution. It also shows that the algorithm possesses the hill-climbing phenomena.

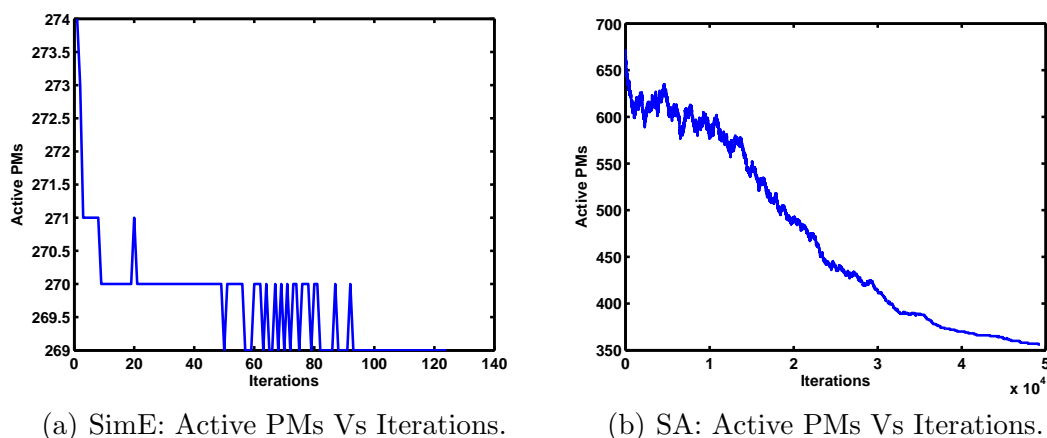


Figure 4.3: Change in number of active PMs with iterations in (a) SimE and (b) SA.

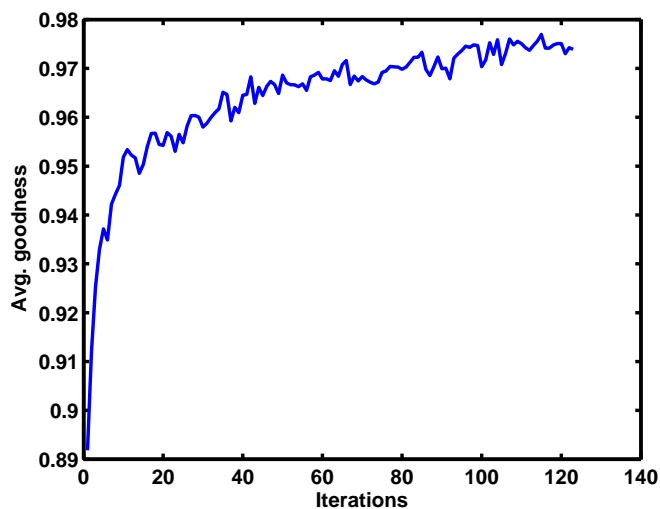
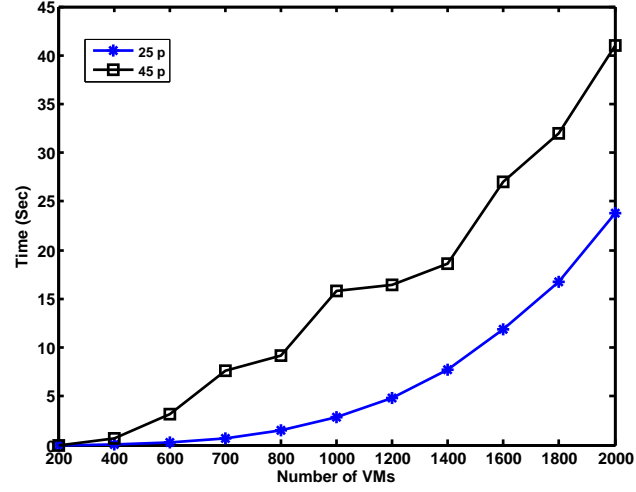


Figure 4.4: SimE: Change in the average goodness of VMs with iterations.

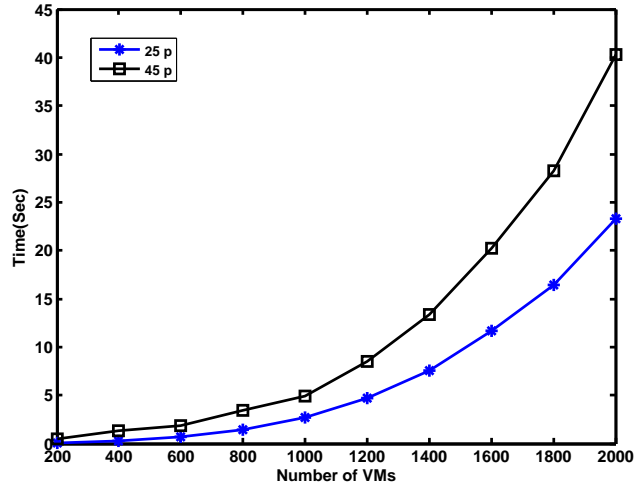
### 4.2.1 Scalability of SimE

In this subsection, I provide results of another set of experiments that are conducted to study whether the proposed algorithm is scalable to larger data centers and more VM requests. In the experiment, the number of VMs is varied from 200 to 2000 for three different levels of correlations (strong-negative, zero and strong-positive correlations), and for two average resource demand values (25% and 45%). SimE was set to stop exploring the the search space when the consolidation ratio is reduced to lower than 1.07, or no improvement was observed in the last 35 iterations. The behaviour of the heuristic is shown in Fig. 4.5. It can be observed from the graph that in the case of  $\overline{v^c} = \overline{v^m} = 25\%$  it takes upto 4 seconds to find a new placement of 1000 VMs but the running time increases to 25 seconds when the number of VMs are increased to 2000. However, in the case of  $\overline{v^c} = \overline{v^m} = 45\%$ , it increases faster than for  $\overline{v^c} = \overline{v^m} = 25\%$ . The reason for two different results with the same number of VMs is that the number of servers used to contain VMs differs in each case: on average 90/25 VMs per server in the case of  $\overline{v^c} = \overline{v^m} = 25\%$  and two VMs in the case of  $\overline{v^c} = \overline{v^m} = 45\%$ . The execution time is measured on Intel *core<sup>TM</sup>2* Quad with 2.67 GHz CPU and 4 GB RAM by taking an average of 50 runs. The algorithm takes less than a minute to solve a difficult assignment problem with up to 2000 VMs.

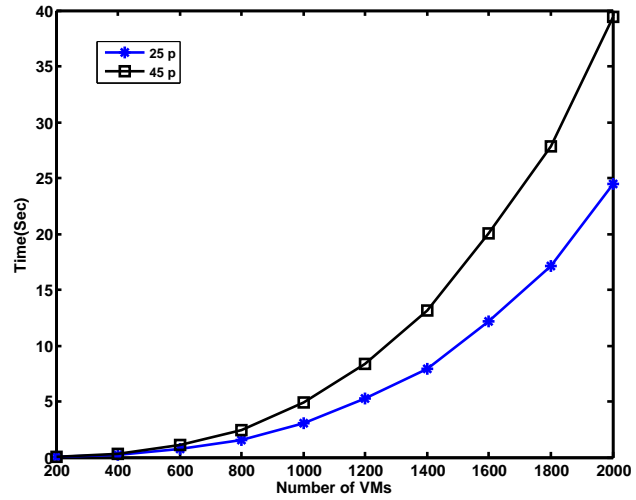




(a) Strong negative correlation.



(b) Zero correlation.



(c) Strong positive correlation.

Figure 4.5: SimE algorithm run-time versus number of VM requests for different correlations.

## CHAPTER 5

# DATA SET GENERATOR AND IMPROVED LOWER BOUND

In this chapter, a new method of data set generation is presented that has multi-dimensional nature and covers a wide variety of parameters that can potentially affect the difficulty of the problem. In addition to this a new lower bound technique is also proposed that is efficient for the problem instances where optimal solution contains few items per bin.

## 5.1 Data Generation for Multidimensional VBPP

In this section, details of the proposed data set generation method are discussed.

### 5.1.1 Parameters

Each instance of multidimensional VBPP can be characterized by a tuple  $(C, n, D, Corr, l, u)$ , where  $C$  represents the bin capacity in each dimension,  $n$  the number of items to be packed,  $D$  the number of dimensions of each item,  $Corr$  is the correlation among the  $d^{th}$  and  $(d-1)^{th}$  dimensions, and,  $l$  and  $u$  define the interval  $[l.C, u.C]$  for the range of item size in each dimension.

### 5.1.2 Procedure

Each tuple  $(C, n, D, Corr, l, u)$  describes a specific class of instances of the multidimensional VBPP. For fixed problem parameters  $C, n, D, Corr, l$ , and  $u$ , any test problem can be interpreted as the realization of a  $D$ -dimensional  $n$  random variable vectors  $I_i$ , i.e.,

$$I_i = [I_i^1, I_i^2, I_i^3, \dots, I_i^D], \quad \forall i \in \{1, 2, 3, \dots, n\} \quad (5.1)$$

These values are generated using the procedure shown in Fig. 5.1. In Fig. 5.1, *halfDiff* is half of the range of interval  $[l.C, u.C]$ , and *rand(1)* is a function that returns a uniformly distributed random real number in the range  $[0, 1)$ . Size of each item in first dimension, i.e.,  $I_i^1$  is set to a random number uniformly distributed in the interval  $[l.C, u.C]$  (refer to lines 9-11). Similarly, sizes in other dimensions, i.e.,  $I_i^d, \forall d \in \{2, 3, \dots, D\}$  are first generated through another uniform random variable in the range equal to half of the the given interval, i.e.,  $[l.C, \frac{l+u}{2}.C]$  and then a probability  $P_c$  is used to decide whether  $I_i^d$  is to be increased by a value

**PROCEDURE**

*DataSet\_Generator*( $C, D, n, Corr, l, u$ );

1. **If** ( $Corr < 0$ ) **Then**
2.      $P_c = 0.0$ ;
3. **ElseIf** ( $Corr = 0$ ) **Then**
4.      $P_c = 0.5$ ;
5. **Else** ( $Corr > 0$ ) **Then**
6.      $P_c = 1.0$ ;
7. **EndIf**;
8.  $halfDiff = C * [\frac{l+u}{2} - l]$ ;
9. **ForEach**  $I_i \in I$  **Do**
10.      $I_i^1 = ((l + (u - l) * rand(1)) * C; /*  $I_i^1 \in U(l.C, u.C)$  */$
11. **EndForEach**;
12. **For**  $d = 2$  **To**  $D$  **Do**
13.      $meanValue = mean(I_i^{d-1})$ ;
14.     **ForEach**  $I_i \in I$  **Do**
15.          $I_i^d = ((l + (\frac{u-l}{2}) * rand(1)) * C; /*  $I_i^d \in U(l.C, \frac{u-l}{2}.C)$  */$
16.          $r = rand(1)$ ;
17.         **If** ( $r < P_c \wedge I_i^d \geq meanValue$ )  $\vee$  ( $r \geq P_c \wedge I_i^d < meanValue$ ) **Then**
18.              $I_i^d = I_i^d + halfDiff$ ;
19.         **EndIf**;
20.         **If**  $I_i^d > C$  **Then**
21.              $I_i^d = C$ ;
22.         **EndIf**;
23.     **EndForEach**;
24. **EndFor**;
25. **Return** ( $I$ );
26. **End** *DataSet\_Generator*.

Figure 5.1: Procedure to generate data set for multidimensional VBPP.

*halfDiff* or not (refer to lines 15 – 18). This step is to introduce the required correlation between  $d^{th}$  and  $(d - 1)^{th}$  dimensions. The probability  $P_c$  is selected according to the required correlation value (refer to lines 1-7).

### 5.1.3 Example

Let a class of problem instances be characterized by  $(C, n, D, Corr, l, u) = (1000, 20, 4, Corr, 0.001, 0.9)$ , where *Corr* can be Negative, Zero or Positive.

For a negative correlation a sample data set is given in Table 5.1.

| Item No.    | Item size            |         |
|-------------|----------------------|---------|
| 1           | [676, 128, 722, 279] |         |
| 2           | [178, 538, 307, 539] |         |
| 3           | [868, 189, 861, 81]  |         |
| 4           | [550, 204, 493, 121] |         |
| 5           | [89, 797, 389, 747]  |         |
| 6           | [208, 836, 68, 500]  |         |
| 7           | [719, 242, 703, 466] |         |
| 8           | [682, 137, 846, 183] |         |
| 9           | [436, 477, 246, 519] |         |
| 10          | [198, 572, 98, 551]  |         |
| 11          | [157, 730, 444, 821] |         |
| 12          | [858, 173, 608, 246] |         |
| 13          | [754, 328, 578, 243] |         |
| 14          | [347, 878, 329, 645] |         |
| 15          | [548, 73, 520, 461]  |         |
| 16          | [899, 188, 707, 99]  |         |
| 17          | [302, 845, 111, 857] |         |
| 18          | [367, 636, 276, 516] |         |
| 19          | [884, 366, 818, 239] |         |
| 20          | [293, 893, 199, 689] |         |
| Correlation | Dimension 1 and 2    | -0.8041 |
|             | Dimension 2 and 3    | -0.7887 |
|             | Dimension 3 and 4    | -0.7343 |

Table 5.1: Sample data set with negative correlation.

## 5.2 Improved Lower Bound

In order to assess the performance of heuristic, heuristics are compared across the consolidation ratio( $q/LB$ ), which is defined as the ratio of the obtained cost, i.e., number of bins  $q$  to the estimated lower-bound ( $LB$ ). For this reason, in order to get a better estimate of the performance of proposed heuristic, a new procedure for a tighter lower bound is proposed.

### 5.2.1 Procedure for Calculation of $LB$

In case of 1D-BPP, a continuous lower bound is calculated with an assumption that items can be allocated in fractional quantities of their sizes. For MDVBPP, one natural way of estimating the lower bound is to calculate maximum of the continuous lower bound values obtained by considering each of its  $D$  dimensions individually at a time. This can be done using Equation 5.2. Similar lower bound is proposed by Spieksma [36] for  $D = 2$ .

$$LB_c = \max \left( \left\lceil \sum_{i=1}^n \frac{I_i^1}{C^1} \right\rceil, \left\lceil \sum_{i=1}^n \frac{I_i^2}{C^2} \right\rceil, \dots, \left\lceil \sum_{i=1}^n \frac{I_i^D}{C^D} \right\rceil \right) \quad (5.2)$$

This lower bound  $LB_c$  is trivially computed in  $O(n)$  time, and seems to be appropriate when items are relatively small in size w.r.t. the capacity of bins [36] (e.g., in our case for classes where  $l \in [0.001, 0.25]$  and  $u \in [0.1, 0.4]$ ). This is due to the fact that optimal solutions of such instances tend to have only few bins and little empty spaces in them that leads to less amount of error in lower bound

estimation. On the other hand, this lower-bound ( $LB_c$ ) inherently performs poor for the instances where optimal solution contains many empty spaces. For such instances another lower-bound procedure is proposed, called  $LB_2$ . The idea is to find the maximum number of items for which it is known that no two of these items can be assigned to the same bin. Evidently, this number serves as a lower bound for the optimal solution. The procedure to calculate this number is shown in Fig. 5.2. Core of the procedure is to divide a given set of items  $I$  into three subsets  $S_{alone}$ ,  $S_t$  and  $S_r$ . For illustration of the procedure, let's consider the data set given in Table. 5.2.

| $I_i$    | Item size            | $S_i$  | $ S_i $ |
|----------|----------------------|--|---------|
| $I_1$    | [321, 666, 878, 220] | $S_1 = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$   | 11      |
| $I_2$    | [324, 212, 525, 667] | $S_2 = \{1, 3, 4, 5, 6, 7, 8, 9, 11, 12\}$       | 10      |
| $I_3$    | [315, 232, 566, 358] | $S_3 = \{1, 2, 4, 5, 7, 9, 10, 11, 12\}$         | 9       |
| $I_4$    | [87, 67, 680, 258]   | $S_4 = \{1, 2, 3, 5, 7, 9, 10, 11, 12\}$         | 9       |
| $I_5$    | [636, 233, 619, 759] | $S_5 = \{1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12\}$   | 11      |
| $I_6$    | [560, 482, 319, 568] | $S_6 = \{1, 2, 5, 7, 8, 9, 11, 12\}$             | 8       |
| $I_7$    | [7, 428, 847, 774]   | $S_7 = \{1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12\}$   | 11      |
| $I_8$    | [562, 636, 43, 398]  | $S_8 = \{1, 2, 5, 6, 7, 9, 11, 12\}$             | 8       |
| $I_9$    | [204, 781, 710, 739] | $S_9 = \{1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12\}$   | 11      |
| $I_{10}$ | [409, 294, 469, 197] | $S_{10} = \{1, 3, 4, 5, 7, 9, 12\}$              | 7       |
| $I_{11}$ | [350, 311, 221, 775] | $S_{11} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 12\}$     | 10      |
| $I_{12}$ | [464, 210, 598, 794] | $S_{12} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ | 11      |

Table 5.2: Sample data set.

*Step1* generates sets  $S_i$  corresponding to each item  $I_i$ , where each set  $S_i$  contains all those items that cannot be combined with item  $I_i$ . These sets are shown in third column of Table. 5.2. Complexity of this step is  $O(n^2)$  as feasibility of packing of each item with every other item is to be tested one by one. In *step2*, set  $S_{alone}$  is constructed by adding all those items that cannot be packed with any

**Procedure***LowerBound*( $C, I$ );/\* **n**: Number of items; \*//\* **I**: Set of all items, where  $|I| = n$ ; \*//\* **C**: Total capacity in each dimension; \*/*STEP 1* :**ForEach**  $I_i \in I$  **Do**

/\* Construct a set  $S_i$  containing all the items that cannot be combined with  $I_i$  due to capacity constraints in any dimension. \*/

$S_i = \{I \setminus I_i \mid \exists d \in \{1, 2, 3, \dots, D\} : I_i^d + I_j^d > C\}$ ;

**EndForEach**;*STEP 2* :

/\* Construct a set  $S_{alone}$  containing all the items that cannot be combined with any other item. \*/

$S_{alone} = \{I_i : |S_i| = n - 1\}$ ;

*STEP 3* :

/\* Subtract the items in set  $S_{alone}$  from all other sets, i.e., set  $I$  and each set  $S_i$  to reduce the problem size for remaining steps. \*/

$I = I \setminus S_{alone}$ ;

**ForEach**  $I_i \in I$  **Do**

$S_i = S_i \setminus S_{alone}$ ;

**EndForEach**;*STEP 4* :**ForEach**  $I_i \in I$  **Do**

/\* Construct a set  $t_i$  such that none of its member item can share the same bin with any other item belonging to the same set  $t_i$  due to capacity constraint violation in any dimension. \*/

$t_i = \{I_i\}$ ;

$temp = S_i$ ;

**While**  $temp \neq \emptyset$

pick one item say  $I_j$  from the set  $temp$

$t_i = t_i \cup \{I_j\}$ ;

$temp = temp \cap \{S_j\}$ ;

**EndWhile**;

**EndForEach**;*STEP 5* :

$S_t = \text{largest } t_i$ ;

$LB_2 = |S_{alone}| + |S_t|$ ;

**Return** ( $LB_2$ );**End** *LowerBound*.Figure 5.2: Procedure for determining the lower-bound  $LB_2$ .



other item in set  $I$ . Such items are identified by the cardinality of their corresponding set  $S_i$ . This step is executed in  $O(n)$  time. In our example  $S_{alone}$  is as follows:

$$S_{alone} = \{1, 5, 7, 9, 12\}$$

As these five items will not share a bin with any other item in set  $I$ , so we subtract these items from set  $I$  and focus on the remaining items only. This is done in *step3* of the procedure (Fig. 5.2). This subtraction step significantly reduces the execution time of the rest of the steps. The reduced problem set for further calculation is shown in Table. 5.3.

| $I_i$    | Item size            | $S_i$                        |
|----------|----------------------|------------------------------|
| $I_2$    | [324, 212, 525, 667] | $S_2 = \{3, 4, 6, 8, 11\}$   |
| $I_3$    | [315, 232, 566, 358] | $S_3 = \{2, 4, 10, 11\}$     |
| $I_4$    | [87, 67, 680, 258]   | $S_4 = \{2, 3, 10, 11\}$     |
| $I_6$    | [560, 482, 319, 568] | $S_6 = \{2, 8, 11\}$         |
| $I_8$    | [562, 636, 43, 398]  | $S_8 = \{2, 6, 11\}$         |
| $I_{10}$ | [409, 294, 469, 197] | $S_{10} = \{3, 4\}$          |
| $I_{11}$ | [350, 311, 221, 775] | $S_{11} = \{2, 3, 4, 6, 8\}$ |

Table 5.3: Reduced problem set at an intermediate stage of lower bound procedure.

Then in *step4* all possible sets  $t_i$  corresponding to each item  $I_i$  in reduced problem set are calculated in such a way that items in each set  $t_i$  cannot share the same bin with other items in the same set. This step constructs each set  $t_i$  in multiple iterations. Iterative steps to calculate set  $t_2$  are shown in Table. 5.4. Clearly no two items in set  $t_2 = \{2, 3, 4, 11\}$  can be combined with each other in one bin. Similarly remaining sets  $t_3 = \{3, 2, 4, 11\}$ ,  $t_4 = \{4, 2, 3, 11\}$ ,  $t_6 = \{6, 2, 8, 11\}$ ,  $t_8 = \{8, 2, 6, 11\}$ ,  $t_{10} = \{10, 3, 4\}$ ,  $t_{11} = \{11, 2, 3, 4\}$  can also be calculated. In the last step, largest of these sets  $t_i$  is declared as set  $S_i$ . In

this example,  $S_t = \{2, 3, 4, 11\}$ . Rest of the items are placed in set  $S_r$ , these items may or may not share a bin with the items in set  $S_t$ . In this example,  $S_r = I \setminus \{S_{alone} \cup S_t\} = \{6, 8, 10\}$ . At the end the procedure returns the sum of cardinality of sets  $S_{alone}$  and  $S_t$ .  $LB_2 = |S_{alone}| + |S_t| = 9$ . Note that in this example  $LB_2 > LB_c$ , where

$$LB_c = \max(\lceil 4239/1000 \rceil, \lceil 4552/1000 \rceil, \lceil 6475/1000 \rceil, \lceil 6507/1000 \rceil) = 7$$

| Iter. | while | $t_2$                | temp   |
|-------|-------|----------------------|--|
| -     | -     | $t_2 = \{2\}$        | $\text{temp} = S_2 = \{3, 4, 6, 8, 11\}$   |
| 1     | true  | $t_2 = \{2,3\}$      | $\text{temp} = \text{temp} \cap S_3 = \{3, 4, 6, 8, 11\} \cap \{2, 4, 10, 11\} = \{4,11\}$ |
| 2     | true  | $t_2 = \{2,3,4\}$    | $\text{temp} = \text{temp} \cap S_4 = \{4,11\} \cap \{2, 3, 10, 11\} = \{11\}$             |
| 3     | true  | $t_2 = \{2,3,4,11\}$ | $\text{temp} = \text{temp} \cap S_{11} = \{11\} \cap \{2, 3, 4, 6, 8\} = \{\}$             |
| 4     | false | -                    | -  |

Table 5.4: Iterative steps to calculate set  $t_2$ .

Complexity of lower-bound  $LB_2$  procedure is  $O(n^2)$ . This lower-bound is suited for problem instances where several items have high values for one or more number of dimensions. Evidently, the overall lower-bound is equal to the maximum of the two.

$$LB = \max\{LB_c, LB_2\} \quad (5.3)$$

## CHAPTER 6

# EXTENDED EXPERIMENTS

In this chapter, detail of another experiment is presented. Experiment is designed to evaluate the performance of the proposed algorithm for multi-dimensional data set generated using the method discussed in Chapter 5.

### 6.1 Performance Evaluation

In this section we provide performance evaluation of our proposed approach with respect to solution quality and run time. First, we compare it with the Norm Based-FFD ( $\text{FFD}_{\text{NB}}$ ) and Dot Product-Based FFD ( $\text{FFD}_{\text{DP}}$ ) that are implemented in Microsoft's Virtual Machine Manager [26, 27], and a well-known iterative heuristic, Simulated Annealing (SA) [55]. Then we discuss the solution quality and performance of the SimE heuristic.

### 6.1.1 Simulation Setup

The programs for the proposed SimE algorithm, SA,  $\text{FFD}_{\text{NB}}$  and  $\text{FFD}_{\text{DP}}$  heuristics were coded in MATLAB and run on an Intel Xeon E5405 with 2.00 GHz CPU (2 processors) and 4 GB RAM. Again the SimE algorithm was set to stop exploring the search space if no improvement was observed in the last 75 iterations. The *Bias* value was set to 0, and maximum size (*maxSelection*) of selection set was restricted to 40% of total items (as discussed in Section 3.1.2).

#### Test Problems

To make synthetic instances more representative and cover a wide range of possible workloads, problem instances were generated with the following different parameter values, using the procedure discussed in Sec. 5.1 Fig. 5.1.

Bin capacity in each dimension:  $C = 1000$

Problem size (number of items):  $n = 250, 500$

Resource dimensions:  $D = 4$

Correlation:  $\text{Corr} = \text{Negative, Zero, Positive}$

lower limit of item weight  $l = 0.05, 0.15, 0.25, 0.35$

upper limit of item weight  $u = 0.1, 0.2, 0.3, \dots, 1.0$

Due to nondeterministic behaviour, average of results obtained from 20 independent runs are reported.

### 6.1.2 Results and Discussion

To evaluate the efficiency of the proposed SimE algorithm, its performance is compared to that of  $\text{FFD}_{\text{NB}}$ ,  $\text{FFD}_{\text{DP}}$  and SA. The comparison metric is consolidation ratio ( $q/LB$ ) calculated as ratio of number of bins used  $q$  to the theoretical lower bound  $LB$  which is estimated using Equation 5.3. A value  $q/LB$  closer to 1.0 represents higher efficiency. Table 6.1 lists the average value  $q/LB$  obtained by these algorithms for different correlation and lower and upper limits of item weight values.

From Table 6.1 we note the following:

- For all algorithms applied, consolidation ratio increases with change of correlation from positive to negative. However a deviation from this trend is observed in the test cases where average value of the item size is high. The reason is, for the negatively correlated instances of these test cases, most packing solutions have only one or two items per bin, and hence obtaining the optimum solution becomes a little trivial.
- Timing performance of both deterministic heuristics is better than SimE and SA as expected. SimE takes far less time than the other non-deterministic heuristic, SA.
- For all cases considered, the engineered SimE heuristic gives better consolidation efficiency when compared to  $\text{FFD}_{\text{DP}}$ ,  $\text{FFD}_{\text{NB}}$ , and SA.

Although SimE has a little inferior performance than  $\text{FFD}_{\text{DP}}$  and  $\text{FFD}_{\text{NB}}$  with respect to run time, yet if we consider the solution quality, we see a significant

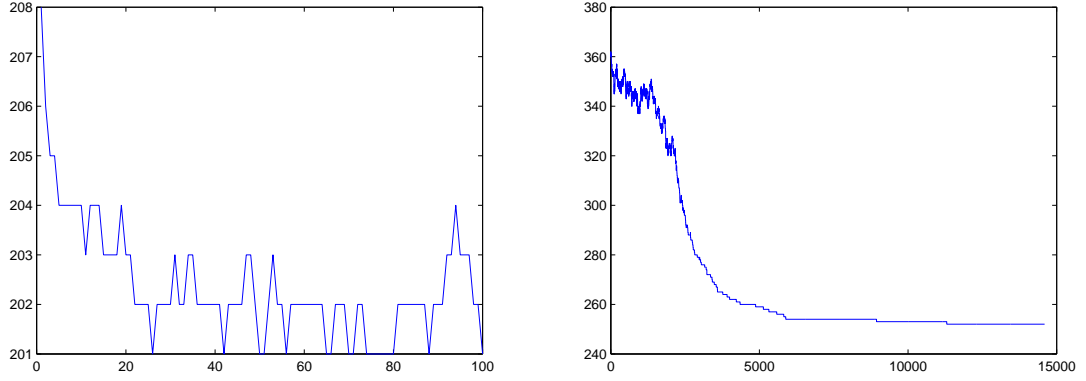
improvement by SimE. This is because these heuristics are single pass and pack the items with best effort in one go. While the proposed SimE is a multi-pass heuristic that after each pass probabilistically picks a small number of items with low goodness value and mutates them. This helps improving the solution quality. The difference in execution times is due to the reason that  $\text{FFD}_{\text{DP}}$  and  $\text{FFD}_{\text{NB}}$  algorithms are constructive heuristics, while SimE is an iterative one.

SimE and SA both are iterative nondeterministic heuristics that incorporate domain specific knowledge to dictate the search strategy. They also tolerate some element of non-determinism that helps the search escape out of local minima. They rely on the use of a suitable cost function which provides feedback to the algorithm as the search progresses. The principle difference among them is how and where domain-specific knowledge is used. For example, in simulated annealing such knowledge is mainly included in the cost function (i.e., number of bins in this case). Items involved in a perturbation are selected randomly, and perturbations are accepted or rejected according to the Metropolis criterion which is a function of the cost [55]. SimE has following differences with SA:

- (1) In SA, a perturbation of current state (solution) is a single move, while for SimE it is a compound move.
- (2) For SA the elements involved in the move are selected at random, while for SimE the elements (usually more than two) are selected based on their goodnesses;
- (3) For SA the iterative process is guided by a parameter called *temperature*,

while for SimE the search process is guided by the individual goodnesses of the solution components.

Thus SimE requires fewer iterations to converge towards a desirable solution [55]. Change in cost of SimE and SA with iterations is illustrated in Fig. 6.1. It is clearly seen that SimE quickly converges towards better solution through a few initial iterations. The plot of average goodness of the solution with iteration is shown in Fig. 6.2, where hill-climbing phenomena is observable.



(a) SimE: Active/open Bins Vs Iterations. (b) SA: Active/open Bins Vs Iterations.

Figure 6.1: Change in number of active/open Bins with iterations in (a) SimE and (b) SA.

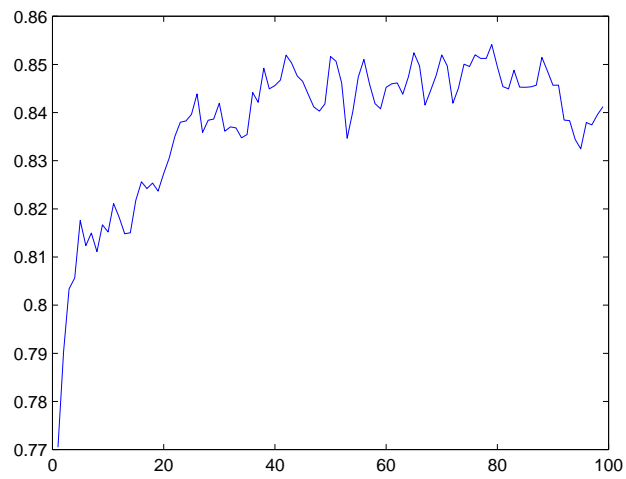


Figure 6.2: SimE: Change in the average goodness of items with iterations.



| $l$  | $u$ | Avg.  | Algo.             | n = 250    |        |          |        |            |        |            |         | n = 500  |         |            |         |      |      |      |      |
|------|-----|-------|-------------------|------------|--------|----------|--------|------------|--------|------------|---------|----------|---------|------------|---------|------|------|------|------|
|      |     |       |                   | Neg. Corr. |        | No Corr. |        | Pos. Corr. |        | Neg. Corr. |         | No Corr. |         | Pos. Corr. |         |      |      |      |      |
|      |     |       |                   | q/LB       | Time   | q/LB     | Time   | q/LB       | Time   | q/LB       | Time    | q/LB     | Time    | q/LB       | Time    | q/LB | Time | q/LB | Time |
| 0.05 | 0.2 | 0.125 | FFD <sub>NB</sub> | 1.127      | 0.293  | 1.11     | 0.293  | 1.065      | 0.285  | 1.112      | 1.11    | 1.111    | 1.105   | 1.065      | 1.084   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.12       | 0.314  | 1.104    | 0.31   | 1.067      | 0.303  | 1.114      | 1.182   | 1.118    | 1.179   | 1.065      | 1.151   |      |      |      |      |
|      |     |       | SA                | 1.545      | 21.978 | 1.527    | 21.732 | 1.521      | 21.676 | 1.664      | 143.2   | 1.678    | 141.601 | 1.636      | 144.019 |      |      |      |      |
|      |     |       | SimE              | 1.056      | 2.527  | 1.045    | 2.59   | 1.02       | 1.474  | 1.049      | 12.119  | 1.051    | 12.43   | 1.02       | 9.873   |      |      |      |      |
|      | 0.5 | 0.275 | FFD <sub>NB</sub> | 1.204      | 0.48   | 1.205    | 0.485  | 1.143      | 0.444  | 1.206      | 1.828   | 1.194    | 1.808   | 1.142      | 1.699   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.203      | 0.524  | 1.205    | 0.53   | 1.142      | 0.489  | 1.206      | 2.008   | 1.195    | 2       | 1.14       | 1.855   |      |      |      |      |
|      |     |       | SA                | 1.299      | 24.572 | 1.336    | 24.454 | 1.257      | 23.786 | 1.34       | 155.349 | 1.371    | 154.12  | 1.291      | 154.093 |      |      |      |      |
|      |     |       | SimE              | 1.111      | 5.089  | 1.072    | 4.816  | 1.042      | 3.998  | 1.111      | 29.351  | 1.066    | 24.287  | 1.039      | 20.971  |      |      |      |      |
|      | 0.6 | 0.325 | FFD <sub>NB</sub> | 1.336      | 0.573  | 1.261    | 0.556  | 1.194      | 0.491  | 1.322      | 2.191   | 1.241    | 2.073   | 1.177      | 1.883   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.331      | 0.635  | 1.261    | 0.611  | 1.197      | 0.54   | 1.322      | 2.426   | 1.248    | 2.301   | 1.179      | 2.048   |      |      |      |      |
|      |     |       | SA                | 1.393      | 26.836 | 1.333    | 26.309 | 1.245      | 25.044 | 1.424      | 165.748 | 1.368    | 162.359 | 1.277      | 160.187 |      |      |      |      |
|      |     |       | SimE              | 1.172      | 7.449  | 1.093    | 6.158  | 1.052      | 4.367  | 1.158      | 39.936  | 1.081    | 34.45   | 1.047      | 23.821  |      |      |      |      |
|      | 0.9 | 0.475 | FFD <sub>NB</sub> | 1.2        | 0.777  | 1.182    | 0.785  | 1.229      | 0.676  | 1.237      | 2.897   | 1.196    | 2.946   | 1.215      | 2.567   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.202      | 0.866  | 1.187    | 0.868  | 1.225      | 0.752  | 1.233      | 3.242   | 1.196    | 3.299   | 1.219      | 2.842   |      |      |      |      |
|      |     |       | SA                | 1.176      | 40.107 | 1.164    | 40.247 | 1.237      | 32.771 | 1.277      | 226.709 | 1.23     | 229.117 | 1.257      | 241.198 |      |      |      |      |
|      |     |       | SimE              | 1.059      | 5.568  | 1.046    | 7.018  | 1.073      | 5.393  | 1.063      | 27.673  | 1.051    | 35.054  | 1.055      | 24.684  |      |      |      |      |
| 0.15 | 0.3 | 0.225 | FFD <sub>NB</sub> | 1.109      | 0.396  | 1.112    | 0.398  | 1.097      | 0.39   | 1.108      | 1.502   | 1.113    | 1.502   | 1.102      | 1.499   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.112      | 0.425  | 1.111    | 0.425  | 1.103      | 0.424  | 1.11       | 1.629   | 1.114    | 1.633   | 1.101      | 1.618   |      |      |      |      |
|      |     |       | SA                | 1.23       | 22.87  | 1.24     | 22.721 | 1.225      | 22.572 | 1.271      | 146.324 | 1.287    | 146.16  | 1.269      | 177.612 |      |      |      |      |
|      |     |       | SimE              | 1.101      | 2.916  | 1.06     | 3.112  | 1.019      | 2.453  | 1.097      | 15.042  | 1.064    | 17.736  | 1.017      | 10.674  |      |      |      |      |
|      | 0.5 | 0.325 | FFD <sub>NB</sub> | 1.314      | 0.573  | 1.248    | 0.477  | 1.154      | 0.5    | 1.301      | 2.152   | 1.255    | 2.084   | 1.15       | 1.93    |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.32       | 0.628  | 1.249    | 0.503  | 1.152      | 0.553  | 1.308      | 2.405   | 1.25     | 2.312   | 1.149      | 2.106   |      |      |      |      |
|      |     |       | SA                | 1.383      | 26.314 | 1.312    | 25.715 | 1.21       | 24.879 | 1.403      | 162.245 | 1.351    | 160.452 | 1.233      | 155.373 |      |      |      |      |
|      |     |       | SimE              | 1.148      | 6.718  | 1.076    | 4.581  | 1.083      | 5.442  | 1.121      | 32.171  | 1.066    | 31.506  | 1.082      | 28.959  |      |      |      |      |
|      | 0.6 | 0.375 | FFD <sub>NB</sub> | 1.312      | 0.541  | 1.278    | 0.615  | 1.269      | 0.571  | 1.318      | 2.394   | 1.276    | 2.34    | 1.251      | 2.197   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.313      | 0.571  | 1.277    | 0.672  | 1.252      | 0.637  | 1.316      | 2.629   | 1.272    | 2.575   | 1.246      | 2.417   |      |      |      |      |
|      |     |       | SA                | 1.31       | 27.684 | 1.283    | 27.571 | 1.245      | 26.667 | 1.323      | 168.111 | 1.307    | 168.263 | 1.266      | 163.977 |      |      |      |      |
|      |     |       | SimE              | 1.29       | 6.665  | 1.146    | 7.525  | 1.07       | 5.379  | 1.287      | 52.965  | 1.144    | 40.799  | 1.062      | 27.05   |      |      |      |      |
|      | 1.0 | 0.575 | FFD <sub>NB</sub> | 1.204      | 0.53   | 1.202    | 0.525  | 1.148      | 0.501  | 1.199      | 2.023   | 1.193    | 2.014   | 1.151      | 1.938   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.203      | 0.585  | 1.195    | 0.576  | 1.149      | 0.551  | 1.199      | 2.235   | 1.195    | 2.212   | 1.151      | 2.112   |      |      |      |      |
|      |     |       | SA                | 1.291      | 25.614 | 1.272    | 25.324 | 1.2        | 24.687 | 1.325      | 176.725 | 1.311    | 190.82  | 1.218      | 183.514 |      |      |      |      |
|      |     |       | SimE              | 1.036      | 3.535  | 1.033    | 3.457  | 1.039      | 4.32   | 1.026      | 16.857  | 1.024    | 16.884  | 1.034      | 23.898  |      |      |      |      |
| 0.25 | 0.4 | 0.325 | FFD <sub>NB</sub> | 1.311      | 0.619  | 1.3      | 0.623  | 1.206      | 0.576  | 1.318      | 2.395   | 1.303    | 2.391   | 1.199      | 2.237   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.311      | 0.679  | 1.3      | 0.688  | 1.201      | 0.636  | 1.318      | 2.662   | 1.304    | 2.639   | 1.196      | 2.457   |      |      |      |      |
|      |     |       | SA                | 1.314      | 27.17  | 1.308    | 26.979 | 1.204      | 26.191 | 1.325      | 185.728 | 1.322    | 201.133 | 1.224      | 192.838 |      |      |      |      |
|      |     |       | SimE              | 1.311      | 6.243  | 1.251    | 7.66   | 1.089      | 4.633  | 1.318      | 34.955  | 1.244    | 56.622  | 1.088      | 26.159  |      |      |      |      |
|      | 0.7 | 0.475 | FFD <sub>NB</sub> | 1.203      | 0.71   | 1.218    | 0.757  | 1.235      | 0.701  | 1.196      | 2.657   | 1.272    | 2.834   | 1.221      | 2.655   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.207      | 0.792  | 1.215    | 0.835  | 1.236      | 0.766  | 1.196      | 2.952   | 1.276    | 3.154   | 1.223      | 2.93    |      |      |      |      |
|      |     |       | SA                | 1.18       | 35.198 | 1.193    | 37.529 | 1.19       | 32.109 | 1.223      | 229.548 | 1.303    | 266.469 | 1.196      | 228.65  |      |      |      |      |
|      |     |       | SimE              | 1.036      | 3.49   | 1.053    | 5.257  | 1.02       | 2.876  | 1.017      | 13.83   | 1.066    | 23.826  | 1.001      | 4.073   |      |      |      |      |
|      | 0.9 | 0.575 | FFD <sub>NB</sub> | 1.001      | 1.076  | 1.019    | 1.008  | 1.189      | 0.82   | 1.002      | 4.174   | 1.027    | 3.92    | 1.187      | 3.201   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.001      | 1.198  | 1.017    | 1.131  | 1.189      | 0.913  | 1.002      | 4.686   | 1.028    | 4.391   | 1.187      | 3.562   |      |      |      |      |
|      |     |       | SA                | 1.001      | 95.013 | 1.012    | 62.908 | 1.179      | 39.942 | 1.003      | 527.84  | 1.028    | 415.137 | 1.185      | 240.335 |      |      |      |      |
|      |     |       | SimE              | 1          | 1.653  | 1.004    | 6.321  | 1.167      | 7.686  | 1          | 14.198  | 1.007    | 65.411  | 1.167      | 42.681  |      |      |      |      |
| 0.35 | 0.5 | 0.425 | FFD <sub>NB</sub> | 1.166      | 0.615  | 1.164    | 0.618  | 1.167      | 0.618  | 1.17       | 2.398   | 1.166    | 2.407   | 1.17       | 2.414   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1.166      | 0.683  | 1.164    | 0.682  | 1.167      | 0.688  | 1.17       | 2.651   | 1.166    | 2.657   | 1.17       | 2.653   |      |      |      |      |
|      |     |       | SA                | 1.17       | 26.953 | 1.167    | 23.753 | 1.173      | 25.335 | 1.177      | 168.27  | 1.172    | 168.807 | 1.178      | 166.176 |      |      |      |      |
|      |     |       | SimE              | 1.166      | 4.9    | 1.164    | 4.848  | 1.167      | 4.833  | 1.17       | 25.999  | 1.166    | 25.731  | 1.17       | 25.878  |      |      |      |      |
|      | 0.7 | 0.525 | FFD <sub>NB</sub> | 1          | 1.081  | 1.007    | 1.042  | 1.124      | 0.839  | 1.001      | 4.21    | 1.01     | 4.053   | 1.125      | 3.292   |      |      |      |      |
|      |     |       | FFD <sub>DP</sub> | 1          | 1.203  | 1.007    | 1.166  | 1.125      | 0.938  | 1.001      | 4.727   | 1.011    | 4.561   | 1.125      | 3.668   |      |      |      |      |
|      |     |       | SA                | 1          | 0.085  | 1.004    | 71.519 | 1.103      | 43.325 | 1.001      | 205.851 | 1.011    | 604.484 | 1.122      | 324.597 |      |      |      |      |
|      |     |       | SimE              | 1          | 0.435  | 1.001    | 2.796  | 1.079      | 6.602  | 1.001      | 23.751  | 1.002    | 41.736  | 1.088      | 36.552  |      |      |      |      |

Table 6.1: Performance comparison of FFD<sub>NB</sub>, FFD<sub>DP</sub>, SA and SimE.

## CHAPTER 7

# CONCLUSION AND FUTURE WORK

A major concern for today's cloud service managers is reducing energy consumption. This study investigated a multi-dimensional VM consolidation model to solve the problem. In this work we presented the engineering of Simulated Evolution (SimE) search heuristic to find better solutions for the combinatorial NP-hard optimization problem, virtual machine assignment. Solutions in Simulated Evolution heuristic evolve based on the current goodness value of the assignments of VMs to PMs. A goodness measure is developed that enables SimE heuristic to quickly find the near-optimal solution. Its performance is evaluated for wide range of different problem instances. The important finding from this study is that the performance of SimE does not get affected by the correlation between different dimensions of VMs. This feature makes this heuristic desirable for all scenarios. In terms of consolidation efficiency, simulation results obtained are better than

those published in literature and with savings in required computation time.

In this work we considered the scenario where all the VM requests are known before placement and the controller allocates them at once, trying to find the optimal allocation in accordance with the objectives and constraints. Such situations arise when a data center starts its operation after a maintenance state or when the data center optimizer/controller takes a decision at the back-end. However, in operational data centers VM requests arrive incrementally over time. In order to address this issue, it is recommended that future studies look into modifications of the algorithm that would work for an online scenario. In such cases existing VMs may have to be migrated for better allocation of new VMs.

# REFERENCES

- [1] Cloud computing. Wikipedia. [Online].  
Available: [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing)
- [2] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.
- [3] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein, “Dynamic energy-aware capacity provisioning for cloud computing environments,” in *Proceedings of the 9th International Conference on Autonomic Computing*. ACM, 2012, pp. 145–154.
- [4] Gartner report, 2013. [Online].  
Available: <http://www.gartner.com/newsroom/id/2352816>
- [5] J. Koomey, “Growth in data center electricity use 2005 to 2010,” *A report by Analytical Press, completed at the request of The New York Times*, 2011.

- [6] A. Beloglazov, “Energy-efficient management of virtual machines in data centers for cloud computing,” Ph.D. dissertation, Department of Computing and Information Systems, The University Of Melbourne, 2013.
- [7] Technology research - Gartner Inc., 2010. [Online].  
Available: <http://www.gartner.com/newsroom/id/1442113>
- [8] Energy star computers specification, 2012. [Online]. Available: [http://www.energystar.gov/ia/partners/prod\\_development/revisions/downloads/computer/ES\\_Computers\\_Draft\\_1\\_Version\\_6.0\\_Specification.pdf](http://www.energystar.gov/ia/partners/prod_development/revisions/downloads/computer/ES_Computers_Draft_1_Version_6.0_Specification.pdf)
- [9] P. Mell and T. Grance, “The nist definition of cloud computing,” *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009.
- [10] VMware esx server. [Online].  
Available: <http://www.vmware.com/products/esx>
- [11] Kernal based virtual machine. [Online]. Available: <http://www.linux-kvm.org/page/MainPage>
- [12] Xensource inc, xen. [Online]. Available: <http://www.xensource.com>
- [13] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, “Heuristics for vector bin packing,” *research. microsoft. com*, 2011.
- [14] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, “No power struggles: Coordinated multi-level power management for the data

- center,” in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 1. ACM, 2008, pp. 48–59.
- [15] C. E. Bash, C. D. Patel, and R. K. Sharma, “Dynamic thermal management of air cooled data centers,” in *Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronics Systems*. IEEE, 2006.
- [16] G. Von Laszewski, L. Wang, A. J. Younge, and X. He, “Power-aware scheduling of virtual machines in dvfs-enabled clusters,” in *Proc. of IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–10.
- [17] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, “Server workload analysis for power minimization using consolidation,” in *Proceedings of the USENIX Annual Technical Conference*. USENIX Association, 2009, pp. 28–28.
- [18] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, “Energy-aware server provisioning and load dispatching for connection-intensive internet services.” in *NSDI*, vol. 8, 2008, pp. 337–350.
- [19] Y. Fu, C. Lu, and H. Wang, “Robust control-theoretic thermal balancing for server clusters,” in *International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–11.
- [20] Technology research - Gartner Inc., 2009. [Online].  
Available: <http://www.gartner.com/newsroom/id/1234513>

- [21] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, “Vmware distributed resource management: Design, implementation, and lessons learned,” *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, 2012.
- [22] L. Shi, J. Furlong, and R. Wang, “Empirical evaluation of vector bin packing algorithms for energy efficient data centers,” in *Symposium on Computers and Communications (ISCC)*. IEEE, 2013, pp. 9–15.
- [23] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, “A multi-objective ant colony system algorithm for virtual machine placement in cloud computing,” *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1230–1242, 2013.
- [24] J. Xu and J. A. Fortes, “Multi-objective virtual machine placement in virtualized data center environments,” in *International Conference on Green Computing and Communications (GreenCom)*. IEEE/ACM, 2010.
- [25] Y. Ajiro and A. Tanaka, “Improving packing algorithms for server consolidation,” in *Int. CMG Conference*, 2007, pp. 399–406.
- [26] Microsoft systems center virtual machine manager. [Online]. Available: <http://www.microsoft.com/systemcenter/virtualmachinemanager>
- [27] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, “Validating heuristics for virtual machines consolidation,” *Microsoft Research, MSR-TR-2011-9*, 2011.

- [28] C. S. Rao, J. J. Geevarghese, and K. Rajan, “Improved approximation bounds for vector bin packing,” *arXiv preprint arXiv:1007.1345*, 2010.
- [29] L. Kou and G. Markowsky, “Multidimensional bin packing algorithms,” *IBM Journal of Research and Development*, vol. 21, no. 5, pp. 443–448, Sept 1977.
- [30] B. T. Han, G. Diehr, and J. S. Cook, “Multiple-type, two-dimensional bin packing problems: Applications and algorithms,” *Annals of Operations Research*, vol. 50, no. 1, pp. 239–261, 1994.
- [31] S. Sarin and W. Wilhelm, “Prototype models for two-dimensional layout design of robot systems,” *IIE transactions*, vol. 16, no. 3, pp. 206–215, 1984.
- [32] J. S. Cook and B. T. Han, “Optimal robot selection and workstation assignment for a CIM system,” *IEEE Transactions on Robotics and Automation*, vol. 10, no. 2, pp. 210–219, 1994.
- [33] S. Y. Chang, H.-C. Hwang, and S. Park, “A two-dimensional vector packing model for the efficient use of coil cassettes,” *Computers & Operations Research*, vol. 32, no. 8, pp. 2051–2058, 2005.
- [34] D. Vercruyssen and H. Muller, “Simulation in production,” *A report of the University of Gent, Belgium*, 1987.
- [35] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C.-C. Yao, “Resource constrained scheduling as generalized bin packing,” *Journal of Combinatorial Theory, Series A*, vol. 21, no. 3, pp. 257–298, 1976.



- [36] F. C. Spieksma, “A branch-and-bound algorithm for the two-dimensional vector packing problem,” *Computers & operations research*, vol. 21, no. 1, pp. 19–25, 1994.
- [37] G. J. Woeginger, “There is no asymptotic {PTAS} for two-dimensional vector packing,” *Information Processing Letters*, vol. 64, no. 6, pp. 293 – 297, 1997. [Online]. Available:  
<http://www.sciencedirect.com/science/article/pii/S0020019097001798>
- [38] A. Caprara and P. Toth, “Lower bounds and algorithms for the 2-dimensional vector packing problem,” *Discrete Applied Mathematics*, vol. 111, no. 3, pp. 231–262, 2001.
- [39] H. Kellerer and V. Kotov, “An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing,” *Operations Research Letters*, vol. 31, no. 1, pp. 35 – 41, 2003. [Online]. Available:  
<http://www.sciencedirect.com/science/article/pii/S0167637702001736>
- [40] A. Yao, “New algorithms for bin packing,” *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 207–227, 1980.
- [41] W. F. De La Vega and G. S. Lueker, “Bin packing can be solved within  $1 + \varepsilon$  in linear time,” *Combinatorica*, vol. 1, no. 4, pp. 349–355, 1981.
- [42] C. Chekuri and S. Khanna, “On multidimensional packing problems,” *SIAM journal on computing*, vol. 33, no. 4, pp. 837–851, 2004.

- [43] R. Jhawar, V. Piuri, and P. Samarati, “Supporting security requirements for resource management in cloud computing.” in *CSE*, 2012, pp. 170–177.
- [44] M. Liu and T. Li, “Optimizing virtual machine consolidation performance on NUMA server architecture for cloud workloads,” in *41st International Symposium on Computer Architecture (ISCA)*. ACM/IEEE, 2014, pp. 325–336.
- [45] S. K. Doddavula, M. Kaushik, and A. Jain, “Implementation of a fast vector packing algorithm and its application for server consolidation,” in *Third International Conference on Cloud Computing Technology and Science (Cloud-Com)*. IEEE, 2011, pp. 332–339.
- [46] L. Shi, B. Butler, D. Botvich, and B. Jennings, “Provisioning of requests for virtual machine sets with placement constraints in iaas clouds,” in *International Symposium on Integrated Network Management*. IEEE, 2013, pp. 499–505.
- [47] A. Beloglazov and R. Buyya, “Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers,” in *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, vol. 4. ACM, 2010.
- [48] O. Shai, E. Shmueli, and D. G. Feitelson, “Heuristics for resource matching in intels compute farm,” in *Job Scheduling Strategies for Parallel Processing*. Springer, 2014, pp. 116–135.

- [49] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, “Resource allocation algorithms for virtualized service hosting platforms,” *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, pp. 962–974, 2010.
- [50] X. An-ping and X. Chun-xiang, “Energy efficient multiresource allocation of virtual machine based on PSO in cloud data center,” *Mathematical Problems in Engineering*, 2014.
- [51] D. Breitgand and A. Epstein, “Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds,” in *Proceedings of INFOCOM*. IEEE, 2012, pp. 2861–2865.
- [52] R. Masson, T. Vidal, J. Michallet, P. H. V. Penna, V. Petrucci, A. Subramanian, and H. Dubedout, “An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems,” *Expert Systems with Applications*, vol. 40, no. 13, pp. 5266–5275, 2013.
- [53] H. H. Kramer, V. Petrucci, A. Subramanian, and E. Uchoa, “A column generation approach for power-aware optimization of virtualized heterogeneous server clusters,” *Computers & Industrial Engineering*, vol. 63, no. 3, pp. 652–662, 2012.
- [54] R.-M. Kling and P. Banerjee, “ESP: A new standard cell placement package using simulated evolution,” in *Proceedings of the 24th Design Automation Conference*. ACM/IEEE, 1987, pp. 60–66.

- [55] S. M. Sait and H. Youssef, *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. IEEE Computer Society Press., 1999.
- [56] —, *VLSI Physical Design Automation: Theory and Practice*. McGraw-Hill, Inc., 1994.
- [57] A. Hartmann, *Phase Transitions in Combinatorial Optimization Problems - Basics, Algorithms and Statistical Mechanics*. Wiley-VCH, 2005.

# Vitae

- Name: Kh. Shahzada Shahid
- Nationality: Pakistani
- Date of Birth: June 29, 1990
- Email: *khawaja.shahzada@gmail.com*
- Permanent Address: Gujranwala, Pakistan
- Publications:

**Journal:** Sadiq M. Sait, Kh. Shahzada Shahid. Engineering Simulated Evolution for Virtual Machine Assignment Problem. *Applied Intelligence*, 2015. DOI: 10.1007/s10489-014-0634-x.

**Conference:** Kh. Shahzada Shahid, Optimal Resource provisioning in Cloud Computing environment. Sixth Scientific Conference for Students of Higher Education in K.S.A. 2015.