# A UTILIZATION BASED GENETIC ALGORITHM FOR VIRTUAL MACHINE PLACEMENT IN CLOUD COMPUTING SYSTEMS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

By

Mustafa Can Çavdar

September 2016

A Utilization Based Genetic Algorithm for Virtual Machine Placement
in Cloud Computing Systems
By Mustafa Can Çavdar
September 2016

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Özgür Ulusoy (Advisor)

İbrahim Körpeoğlu (co-Advisor)

Mustafa Özdal

Ahmet Coşar

Approved for the Graduate School of Engineering and Science:

Levent Onural
Director of the Graduate School

# ABSTRACT

## A UTILIZATION BASED GENETIC ALGORITHM FOR VIRTUAL MACHINE PLACEMENT IN CLOUD COMPUTING SYSTEMS

Mustafa Can Çavdar
M.S. in Computer Engineering
Advisors: Özgür Ulusoy and İbrahim Körpeoğlu
September 2016

Due to increasing demand for cloud computing and related services, cloud providers need to come up with methods and mechanisms that increase performance, availability and reliability of datacenters and cloud computing systems. Server virtualization is a key component to achieve this, which enables sharing of resources of a physical machine among multiple virtual machines in a totally isolated manner. Optimizing virtualization has a very significant effect on the overall performance of cloud computing systems. This requires efficient and effective placement of virtual machines into physical machines. Since this is an optimization problem that involves multiple constraints and objectives, we propose a method based on genetic algorithms to place virtual machines. By considering utilization of machines and node distances, our method aims at reducing resource waste, network load, and energy consumption at the same time. We compared our method with several other methods in terms of utilization achieved, networking bandwidth consumed, and energy costs incurred, using the publicly available CloudSim simulation platform. The results show that our approach provides improved performance compared to other similar approaches.

*Keywords:* Cloud computing, virtualization, genetic algorithm, virtual machine placement.

# ÖZET

# BULUT SİSTEMLERİNDE SANAL MAKİNE YERLEŞTİRİMİ İÇİN FAYDALANMA TEMELLİ BİR GENETİK ALGORİTMA

Mustafa Can Çavdar
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanları: Özgür Ulusoy ve İbrahim Körpeoğlu
Eylül 2016

Bulut bilişim ve ilgili hizmetlere yönelik artan talepten dolayı, bulut sağlayıcıları, veri merkezlerinin ve bulut sistemlerinin performansını, elverişliliğini ve güvenirliliğini artıracak yöntemler oluşturmak zorundadır. Bir fiziksel makinenin kaynaklarının birden çok sanal makine tarafından paylaşılmasını sağlayan sunucu sanallaştırması, bunun gerçekleştirilmesi için ana bileşendir. Sanallaştırmanın eniyileştirilmesi, bulut bilişim sisteminin genel performansı üzerinde önemli bir etkiye sahiptir ve bu da sanal makinelerin, fiziksel makinelere etkili ve verimli bir şekilde yerleştirilmesini gerektirir. Bu durum birden fazla kısıtlama içeren bir eniyileştirme problemi olduğundan dolayı, sanal makineleri yerleştirmek için genetik algoritma temelli bir yöntem önermekteyiz. Makinelerin kullanım oranlarını ve düğümler arası uzaklıkları dikkate alan yöntemimiz; kaynak israfını, ağ yükünü ve enerji tüketimini aynı anda düşürmeyi hedeflemektedir. Yöntemimiz diğer birkaç yöntem ile; gerçekleştirilen kullanım oranı, tüketilen ağ band genişliği ve sebep olunan enerji masrafı kıstasları üzerinden kullanıma açık CloudSim simulasyon platformu kullanılarak karşılaştırılmıştır. Sonuçlar, yaklaşımımızın, karşılaştırılan diğer benzer yaklaşımlara göre daha gelişmiş bir performans sağladığını göstermiştir.

*Anahtar sözcükler*: Bulut bilişim, Sanallaştırma, genetik algoritma, sanal makine yerleştirimi.

# Acknowledgement

First and foremost, I would like to thank my supervisors, Prof. Özgür Ulusoy and Assoc. Prof. İbrahim Körpeoğlu, for their guidance during my study. Their support helped me to complete this thesis.

I would like to thank Prof. Ahmet Coşar and Assist. Prof. Mustafa Özdal for evaluating this thesis.

Last but not the least, I would like to thank my family for their endless support and love. This thesis is dedicated to you.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Cloud computing has become a pervasive technology for providing computing, storage and software services on Internet. This is reasoned by the fact that cloud systems can provide nearly any type of service customers may demand and relieve customers from building their own physical infrastructures. It can also offer services with pay-as-you-go charging model where customers pay according to the amount of the services they use, which makes it even more attractive.

Due to increasing demand for cloud services, optimization of resource consumption becomes even more essential to increase performance, availability and reliability of the cloud systems. Virtualization technologies have been proven to be very useful to meet these requirements. Virtualization enables users and applications to share physical cloud resources in an efficient, effective and isolated manner. With server virtualization multiple virtual machines can run in a single physical machine in a totally isolated manner. In this way cloud providers can serve more number of customers in a flexible and efficient manner.

Different virtual machines requested by users may have different processing, memory, I/O and networking requirements. Physical servers can also have different capacities. This leads to an optimization problem which is known as virtual machine placement problem. A solution to this problem may aim to increase

utilization of physical machines to reduce costs and energy consumption. Due to increasing user demand for cloud computing mentioned earlier, optimization of resource usage becomes a very essential issue to save more energy, reduce costs and meet customer service level agreements (SLAs).

The virtual machine placement problem is a multi-objective constrained NP-hard problem [40]. Genetic algorithm based approaches have been one of the most widely used methods to solve this type of complex problems. A genetic algorithm mimics the natural selection process and in this way tries to find a close to optimal solution to a given complex problem.

In this thesis we propose a genetic algorithm based solution to the virtual machine placement problem, which uses a novel fitness function and chromosome structure and considers resource utilization, network bandwidth usage, and energy costs at the same time. Our method is called Utilization Based Genetic Algorithm (UBGA) and uses genetic algorithm-based heuristic approach to find a close-to-optimum solution. A genetic algorithm requires specification of the chromosome structure and we designed our chromosome structure as a tree representing a datacenter network topology. Leaves of the tree represents the physical machines and each leaf node has a pointer to a list of virtual machines. The fitness function of our genetic algorithm is also designed uniquely with the aim of reducing resource waste and network bandwidth consumption.

Additionally, we developed a utilization based virtual machine placement algorithm that uses Simulated Annealing meta-heuristic approach. We call this algorithm as UBSA. We did this to see which meta-heuristic, genetic algorithm or simulated annealing, would yield better results. The results show that even though they have similar performance, UBGA performs slightly better than UBSA. Therefore UBGA has been our choice for comparing with other approaches from literature.

We integrated our UBGA algorithm into the publicly available CloudSim [6] cloud computing simulator and conducted extensive simulation experiments to evaluate it and compare it with other methods. We compared UBGA with a

random strategy, with the default allocation method of CloudSim simulator, with First Fit Decreasing (FFD) method, and with the VMPGGA algorithm from literature [40]. Our experiments consider homogeneous, uniform and random distribution of resource capacities and demands. The results show that our method is completely superior to random allocation and default CloudSim allocation method, and achieves better performance than FFD and VMPGGA for most scenarios.

Finally, we also compared UBGA with a linear programming method to see how close the solutions it produces are to the optimum values.

The organization of this thesis is as follows. In Chapter 2, we give background information about genetic algorithm concept and cloud computing. In Chapter 3, we give information about the related work that use both evolutionary and non-evolutionary approaches for solving virtual machine placement problem. In Chapter 4, we describe our proposed method in a detailed manner. Chapter 5 describes our approach for virtual machine placement that uses simulated annealing concept. In Chapter 6, we present results of the experiments that evaluate our approach, and finally in Chapter 7, we conclude the paper.

# Chapter 2

# Background and Related Work

In this chapter, we give brief information about genetic algorithms and cloud computing. Then, we show previous works having both evolutionary and non-evolutionary approaches in the field of virtual machine placement.

## 2.1 Genetic Algorithms

A genetic algorithm is a search meta-heuristic that mimics natural selection process of real-life and is commonly used to find close-to-optimal solutions for complex optimization problems.

Genetic Algorithms were invented and developed by John Holland and his students at University of Michigan in 1970s [35]. Their initial aim was to study how adaption of natural selection would work instead of creating new algorithms for certain problems. Holland presented genetic algorithms as an "abstraction of biological evolution" in his book *Adaptation in Natural and Artificial Systems* [18].

Genetic algorithms are even useful for problems with very large search space and large number of variables. In a search space, a genetic algorithm tends to look for selected and better solutions instead of scanning the whole space. Thanks to

mutation feature, it has less probability to get local maximum value rather than global maximum.

A standard genetic algorithm is defined with:

1) A *genetic representation* of solutions

2) A *fitness function* to evaluate candidate solutions

A genetic representation is mostly an array of bits but can be any data structure depending on the on the problem (e.g., a tree in this thesis). Since any property of the selected genetic representation form is fixed (i.e., size of array, height and number of leaves of tree) for each individual of the population, they can be easily changed, and this leads to simple crossover operations. The fitness function is a figure of merit that shows how close a solution is to desired objectives.

When the genetic representation and fitness function are determined, a genetic algorithm continues to generate an initial population (a random set of solutions) and then try to improve it by repetitive application of selection, crossover and mutation.

The initialization phase includes creation of population with several hundreds or thousands of random solutions according to the nature of the problem. These solutions may be impossible to apply to the problem. In some cases, solutions may be directed to become a more optimal one but this is inadvisable since this decreases variety among solution that may lead being stuck at the local maximum.

After the initial population is created, selection, crossover and mutation operations are realized repetitively. Each repetition is called a generation. Selection operation is the process of deciding which individual is chosen from the population. There are some selection algorithms such as tournament selection, which chooses the best individual from randomly chosen subset of solutions, and truncation selection which selects an individual from the best half of any other proportion of the individuals. There are algorithms that only consider solutions

with fitness values higher than a given threshold, however, this, again, may lead to decreased variety.

Crossover operation is used for creating individuals for the next generation of population. After pairs of individuals are determined with the selection phase, selected parents' genes are used to create their offsprings. There are several crossover operations most common of which is uniform crossover. It selects genes from either parent by using a fixed ratio between two parents. If that ratio is 0.5, then the offspring has approximately half of its genes from one parent and other half from the other parent. If that ratio has some other value, then, the crossover operation is biased (probably favors the one with higher fitness value).

Mutation is the genetic operation to maintain genetic diversity among population. Therefore, the genetic algorithm is more likely to avoid the local maximum value by preventing individuals in the population becoming too similar which may lead to stopping evolution. The mutation operation is generally realized by changing the value of a randomly chosen gene.

The genetic algorithm continues until the termination condition is reached. The termination condition can usually be reaching a fixed number of generations, or having no more change for the best individual during a certain number of generations, or finding a solution that satisfies a certain threshold of fitness value. When the algorithm terminates, it outputs best individual as the solution.

## 2.2 Cloud Computing

Cloud computing is an Internet based service that provides shared resources and data to users in an on-demand basis. It enables users to store their data in datacenters. Cloud computing allows companies to run their applications faster with easy management and maintenance. Providers of cloud systems use a "pay as you go" model that allows users to pay for cloud services proportionally to the time they use the services.

Cloud computing has been enabled recently due to low cost of computers and high capacity networks that are easily accessible. Cloud systems have two types: Public and private clouds. Public clouds are open to public use while private clouds only belong to some company or organization.

There are different service models of cloud systems: Infrastructure as a service, platform as a service and software as a service.

Infrastructure as a service (IaaS) provides online resources such as virtual machines and storage to its subscribed customers.

Platform as a service (PaaS) provides a development environment to its customers who are application developers. PaaS providers offer a computing platform that contains a web server, a database, an operating system and a programming language execution environment. Most famous ones are Microsoft Azure and Google App Engine.

Software as a service (SaaS) lets users gain access to software. Cloud providers install a software in their system and users access that software with their clients. Games, and email are examples of this type of service in cloud computing.

## 2.3 Related Work

### 2.3.1 Non-evolutionary Algorithms

There are various types of algorithms to solve the virtual machine placement problem. The algorithms introduced in [22] for placement of virtual machines in cloud data centers have the objective of maximizing a new metric named satisfaction, which reflects the relative suitability of a physical machine for any virtual machine to assign to it. Huang et al. [21] aim to reduce the placement cost by consolidating virtual machine workload in physical machines while preserving service level agreements. Fang et al. [14] propose an approach called VMPlanner

to optimize network elements in the cloud to reduce cost. Goudrazi et al. [17] describe a solution for reducing network cost by replicating virtual machines on different servers. Chaisiri et al. [10] provide an algorithm called OVMP that minimizes cost spending for virtual machine hosting in multiple clouds by using stochastic integer programming.

In [8], a heuristic framework is proposed that establishes an SLA violation decision algorithm for finding overloaded hosts in the datacenter. The algorithm of Bin et al. [4] is for finding an optimal solution of relocation of virtual machines to a non-failed host when its own host has failed, while Anand et al. [3] propose an intelligent algorithm that considers virtual machine usage and minimizes the performance loss during the migration process of the virtual machines. Calcavecchia et al. [5] propose a solution called Backward Speculation Placement which has two parts. The first part deals with the stream of deployment process and the second part optimizes the placement periodically in case of dynamic demands.

In [32], a method is proposed that aims at reducing virtual machine interference as well as organizing resources of physical machines, while in [9] Caron et al. propose a security algorithm specific to cloud systems to place virtual machines. The virtual machine placement method described by Jin et al. in [25] deals with dynamic resource needs of virtual machines with an algorithm that tries to maximize the minimum utilization ratio of physical machines. In [39], Singh et al. aim to reduce energy cost by migrating virtual machines from less loaded physical machines to more loaded ones. In this way, they try to increase the number of idle machines which can be shut down. Alicherry et al. [2] provide methods that place virtual machines by considering inter-VM and intra-VM latency.

The algorithm proposed by Chen et al. [11] is a cost-aware two-phase meta-heuristic called Cut-and-Search that finds an approximate way to find a trade-off point between two cost terms. In [13], the authors propose a power efficient solution that reduces power consumption and brings communicating groups closer in fat-tree topology networks. In [16], two algorithms are evaluated one of which places virtual machines to most demanding virtual link with a greedy approach and the other approach finds potential neighborhood of physical machines.

Hong et al. [19] propose a method that maximizes total profit of cloud game providers with exponential running time. In [20], an algorithm based on a multi-dimensional space partition model is presented that can balance the utilization of resources of physical machines in use. Jayasinghe et al. [23] design a hierarchical placement approach that solves structural constraint aware virtual machine placement problem. The algorithm in [24] is an extension of Markov approximation to optimize dynamic cloud environment.

Kantarci et al. [27] propose a mixed integer linear programming method that places VMs in data centers with minimum power consumption. In [29], an approach is presented that uses Analytic Hierarchy Process to make a trade-off between SLA and low energy consumption. In [30], Le et al. propose a dynamic load distribution policy that considers impact of load placement policies to reduce electricity cost. Li et al. [31] propose a new concept, elasticity, which specifies how the state of the datacenter satisfies growth of VM demands, and a hierarchical VM placement algorithm is proposed to optimize that. In [34], authors design an approximate algorithm that solves traffic aware virtual machine placement to improve scalability of the network.

Moreno et al. [36] introduce an approach to the allocation problem which improves energy efficiency in the cloud by considering workload heterogeneity of the datacenters. In [37], Piao et al. try to minimize data transfer time when a migration occurs. Shi et al. [38] describe a method which uses a first fit heuristic to maximize the profit under SLA and power budget constraints. The method described in [43] has two objectives: minimize required number of PMs and maximize the utilization of used PMs. They define VM request and VM placement matrices to find an optimum solution.

### 2.3.2 Evolutionary Algorithms

Genetic algorithms can be used in solving resource allocation and assignment problems in cloud computing as in [12], which proposes an algorithm for data replica placement both within a data center and among a number of data centers.

There are evolutionary approaches to solve virtual machine placement problem as well. Various methods have been proposed to find an optimal solution considering a number of metrics. Wu et al. [44] describe a solution that only considers energy consumption of a cloud system. They propose a basic traditional genetic algorithm that uses lists as chromosomes. The length of the list is the number of physical machines. They use a fat-tree topology in their experiments and calculate the network cost of a connection between two virtual machines based on this topology. In [41], an Ant colony algorithm is presented, which deals with the mapping problem in permutation form from virtual machines to physical ones. The aim of the algorithm is to propose a solution that can reduce the energy consumption and resource waste simultaneously. In their algorithm, virtual machines are mapped to physical machines based on the probability of their movement.

In [42], an algorithm for dynamic resource allocation is presented, which applies a threshold based method to optimize the process. The algorithm replaces virtual machines dynamically based on workload changes in cloud applications. Joseph et al. [26] propose a family genetic algorithm approach that is integrated into CloudSim. The algorithm divides the process among different families running in parallel to assign virtual machines to physical machines. They made the mutation probability dynamic depending on some parameters and they compare their algorithm with CloudSim's existing allocation policies. Sookhtsaraei et al. [40] describe a multi-purpose genetic algorithm, which is called Group GA, that considers virtual machine placement as a bin-packing problem. They use Falkenauer's idea, described in [15], which is using genetic algorithms for grouping problems. Madhusudhan et al. [33] propose a genetic algorithm in which solutions are represented as trees. In their work, a global resource manager is the root of the tree, physical machines are next level nodes, and leaves are virtual machines. They also use CloudSim toolkit to evaluate their algorithm.

Adamuthe et al. [1] compare the results of non-dominated sorting genetic algorithms whose objective is to maximize profit and balance load in the cloud.

Our proposed method differs from all the methods explained above by using a novel genetic algorithm for finding a close-to-optimal solution, which considers

the utilization of physical machines, network load, and energy cost, all at the same time.

# Chapter 3

# Utilization Based Genetic Algorithm - UBGA

In cloud systems, computing and storage resources of service providers are totally virtualized. Optimization of virtual machine placement is essential for a cloud owner for satisfaction of their customers because more optimized cloud environments provide better performance, availability and reliability.

As mentioned in the previous section, virtual machine placement problem can be considered as a bin packing problem [40]. Virtual machines can be thought as objects and physical machines are bins. The difference of virtual machine placement problem is that there are multiple constraints rather than only one constraint as in the bin packing problem.

## 3.1 Offline Virtual Machine Placement

We propose a genetic algorithm based approach, called Utility Based Genetic Algorithm (UBGA), to optimize virtual machine placement considering utilization of physical machines. Our approach aims to reduce total wasted resources of

physical machines in the cloud. We define unused amount of resources of a physical machine as wasted if that physical machine has at least one virtual machine so that it cannot be shut down. We do not consider the resources of idle machines as wasted since they can be shut down. We also integrated our algorithm into CloudSim toolkit which is an open source cloud computing simulator allowing creation and simulation of datacenters, physical and virtual machines in an easy way.

Since genetic algorithms mimic the natural selection process of real life, which suggests that better individuals have more chance to survive in a population according to evolution theory, it can be used to find a close-to optimum solution for a lot of complex problems. As evolution in real life progresses, species have populations with better individuals after some number of generations. Similarly, a genetic algorithm starts with a population containing random individuals each of which represents a solution to the given problem and improves these individuals at each generation to have better solutions. Therefore, a genetic algorithm produces a more optimum solution to the given problem from one generation to another according to a given criterion, an example of which is called *fitness* function in this paper. At the end, we reach a solution that is good enough and approximates the optimal solution to the problem.

The only setback of genetic algorithms can be thought as their running time complexity since a genetic algorithm may take too much time depending on the specified evolution termination condition and the number of individuals in the population in each generation. However, since the scenario that we focus on in this paper is not a highly dynamic one, time to find a good enough solution is not the most prior concern and therefore does not prevent us from using a genetic algorithm.

### 3.1.1 Problem Description

The following parameters are used for a formal description of our problem.

$\boldsymbol{V}$: a set of virtual machines

$\boldsymbol{P}$: a set of physical machines

$P_{all}$: set of allocated physical machines

$v_i$: virtual machine $i$

$v_i^{cpu}$: CPU requirement of $v_i$

$v_i^{mem}$: memory requirement of $v_i$

$v_i^{bw}$: bandwidth requirement of $v_i$

$p_j$: physical machine $j$

$p_j^{cpu}$: CPU capacity of $p_j$

$p_j^{mem}$: memory capacity of $p_j$

$p_j^{bw}$: network bandwidth capacity of $p_j$

$V_j$: list of virtual machines assigned to $p_j$

$w_j^{cpu}$: wasted CPU percentage of $p_j$

$w_j^{mem}$: wasted memory percentage of $p_j$

$w_j^{bw}$: wasted bandwidth percentage of $p_j$

$S_{ij}$: number of switches between $v_i$ and $v_j$

$F_{ij}$: data flow demand between $v_i$ and $v_j$

$N_{ij}$: network cost of connection between $v_i$ and $v_j$

$E_j$: total energy consumed by $p_j$

$E_j^{idle}$: energy consumed by $p_j$ in idle state

$E_j^{max}$: energy consumed by $p_j$ when it is fully utilized (100% CPU utilization)

$U_j$: utilization of $p_j$

We aim to minimize the wasted resources (CPU, memory and bandwidth) of physical machines, minimize the energy consumed by physical machines, and minimize the forwarding load on network switches due to communication among virtual machines. We can state our virtual machine placement problem as follows:

$$\sum_{p_j \in P} E_j \tag{3.1}$$

$$\sum_{p_j \in P} (w_j^{cpu} + w_j^{mem} + w_j^{bw}) \tag{3.2}$$

$$\sum_{v_i \in V} \sum_{v_j \in V} N_{ij},\ where\ i < j \tag{3.3}$$

are minimized as much as possible subject to

$$\boldsymbol{V} = \bigcup_{p_j \in P} V_j \tag{3.4}$$

$$p_j^{bw} \geq \sum_{v_i \in V_j} v_i^{bw} \tag{3.5}$$

$$p_j^{mem} \geq \sum_{v_i \in V_j} v_i^{mem} \tag{3.6}$$

$$p_j^{cpu} \geq \sum_{v_i \in V_j} v_i^{cpu} \tag{3.7}$$

$$V_i \cap V_j = \emptyset,\ if\ \ i \neq j \tag{3.8}$$

$$U_j = 1 - w_j^{cpu} \tag{3.9}$$

$$E_j = E_j^{idle} + (E_j^{max} - E_j^{idle}) * U_j \tag{3.10}$$

$$N_{ij} = S_{ij} * F_{ij} \tag{3.11}$$

Eq. 3.4 states that every virtual machine is assigned to one physical machine, while Eq. 3.8 specifies that each virtual machine is assigned to only one physical machine.

Equations 3.5, 3.6, and 3.7 state that the total demand by virtual machines for each resource (CPU, memory and bandwidth) does not exceed the amount of the corresponding resource of the physical machine to which those virtual machines are assigned. As stated in Eq. 9, we define the utilization of a physical machine as percentage of CPU usage, as in [44]. Energy consumption of a machine is calculated based on the utilization of the machine, as shown in Eq. 3.10. An idle machine also consumes energy. Eq. 3.11 shows the calculation of network cost. A virtual machine may communicate with another virtual machine and the cost of this communication is defined to be the number of switches between the physical machines hosting the virtual machines *times* the data flow demand between the virtual machines. We created a traffic demand matrix size of $|V|$x$|V|$ that shows desired data flow demands between virtual machines. Both energy consumption and network cost calculations are inspired from [44].

We next give the components of our genetic algorithm for virtual machine placement.

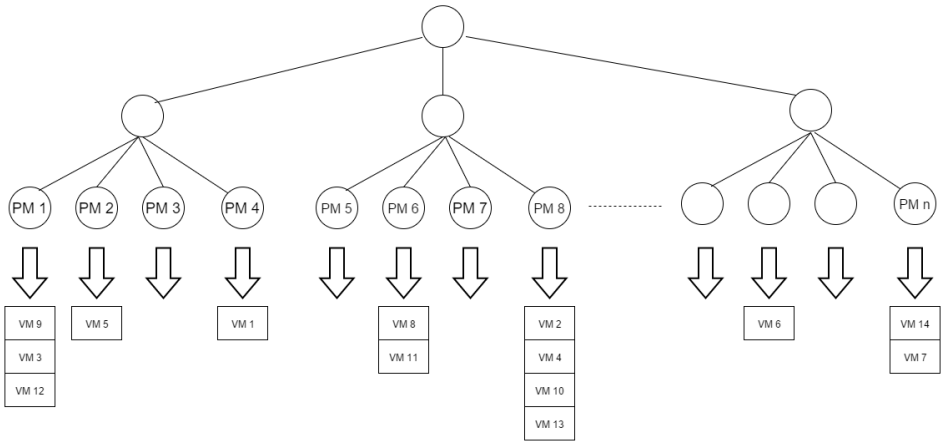


Figure 3.1: The chromosome structure of our genetic algorithm.

### 3.1.2 Chromosome Structure

A chromosome in our proposed genetic algorithm is a tree data structure whose leaves represent the physical machines and non-leaf nodes represent switches that connect these physical machines. Each leaf has a pointer to a list, which keeps the virtual machines that are assigned to the physical machine represented by the leaf. Hence, we have $|P|$ number of lists. Obviously, some of these lists may be empty when no virtual machine is assigned to the corresponding physical machine. The proposed chromosome structure has $|V|$ genes each one representing a single virtual machine. An example of chromosome can be seen in Figure 3.1.

With this tree structure it is easier and faster to determine what percentage of resources of a physical machine is wasted. We also make use of this structure to calculate communication cost of between two virtual machines.

A *particular placement* of virtual machines (feasible or unfeasible) to physical machines is represented by such a tree and its lists. This is called an *individual*.

### 3.1.3 Fitness Function

In the problem decription above, there are 3 objectives to minimize: 1) energy, 2) resource waste, and 3) communication. We may not achive all together. We combine these 3 objectives into a single fitness value to be used in our genetic algorithm, so that at the end all these three objectivies are considered together and in this way are tried to be minimized in a satisfactory (close to optimal) solution found by our genetic algorithm.

We define the *fitness* value of an individual of the population who has $k$ number of over-demanded physical machines as in Eq. 3.12. Here, we have $0 \leq k \leq n$, where $n$ is the number of physical machines in the datacenter.

$$fitness(i) = (k+1) * (\frac{1}{T_{cpu}} * \sum_{p_j \in P_{all}} (-w_j^{cpu} * p_j^{cpu}) + \frac{1}{T_{mem}} * \sum_{p_j \in P_{all}} (-w_j^{mem} * p_j^{mem}) + \frac{1}{T_{bw}} * \sum_{p_j \in P_{all}} (-w_j^{bw} * p_j^{bw}) - \frac{\sum_{v_i \in V} \sum_{v_j \in V} N_{ij}}{N_{max}}) \quad (3.12)$$

where;

$$T_{cpu} = \sum_{p_j \in P_{all}} p_j^{cpu} \quad (3.13)$$

$$T_{mem} = \sum_{p_j \in P_{all}} p_j^{mem} \quad (3.14)$$

$$T_{bw} = \sum_{p_j \in P_{all}} p_j^{bw} \quad (3.15)$$

$$N_{max} = H * \sum_{v_i \in V} \sum_{v_j \in V} F_{ij},\ where\ i < j \quad (3.16)$$

The fitness function, whose value is negative, indicates how well a placement is. A bad placement will have a very low fitness value (a very large negative value) and this is obtained by punishing such bad placements as much as possible. The fitness function punishes over-demand by virtual machines the most, since this is the most undesired case where capacity constraints of physical machines are violated. That is to say, for one of the resources, if the total demand by virtual machines exceeds the capacity of the assigned physical machine, we reduce that individual's fitness score by multiplying the inside sum (which calculates the sum of wasted resources' percentage and normalized network cost) in Eq. 3.12 with the number of over-demanded physical machines. Since the inside sum is negative, we severely penalize over-demand. With this big penalty, we try to hinder the existence of over-demanded physical machines in the final allocation.

On the other hand, wasted resources and network cost are more acceptable than over-demand. Therefore, they have less penalty, which is the result of the inside sum. The inside sum is allowed to increase at most to -4, which is the sum of the total wasted percentage of CPU, memory and bandwidth of all used physical machines and normalized network cost of the virtual machines in the cloud. Therefore, we aim directly to reduce the total wasted percentage of resources and network cost in the whole cloud.

Network cost is normalized in the fitness function by dividing it by maximum possible network cost. Therefore, it has a value between 0 and 1. As a result, resource wastage and network cost have equal impact on fitness score of individuals. Maximum possible network cost is formulated in Eq. 3.16. We multiply the flow demand between two virtual machines with $H$, where $H$ is the maximum number of switches between two virtual machines.

Moreover, we do not penalize individuals for having totally idle physical machines (machines with no VM assigned to them), since idle machines can be shut down. We want to utilize physical machines as much as possible and to realize that we try to increase the number of idle machines as much as possible. Hence, more idle machines in a placement means less energy consumption, as can be seen in Eq. 3.10, which we aimed in Eq. 3.1.

### 3.1.4 Crossover Operation

The crossover operation determines which genes are inherited from which parents, as described in Algorithm 1. The $uniformRate$ in the algorithm is to determine which parent is selected for inheritance. In our experiments, we decided to set $uniformRate$ to 0.5 to have an unbiased gene selection from either parents to improve variety of newly created individuals. According to the generated random value, a parent is selected, and a virtual machine is assigned to the same physical machine in the offspring's chromosome, that it is as assigned in the selected parent's chromosome. We do not favor the parent with higher fitness score to increase variety in the offspring.

**Algorithm 1** Crossover Operation

**Require:** Two parent chromosomes: $C_1$ and $C_2$
**Ensure:** One offspring chromosome: $C_{new}$

```
1: procedure CROSSOVER
2:     for i = 1 to |V| do
3:         randomly create a value between 0 and 1, r;
4:         if r ≤ uniformRate then
5:             set gene i of C_new as gene i of C_1
6:         else
7:             set gene i of C_new as gene i of C_2
8:         end if
9:     end for
10: end procedure
```

### 3.1.5 Selection Operation

We decided to use Tournament Selection operation to pair up individuals and then perform crossover operations for pairs. Our selection operation which is described in Algorithm 2 creates a sub-population with randomly selected individuals from the overall population. Then, it returns the individual with the best fitness value. In the experiments, we set the size of this sub-population to be 5% of overall population. We decided to use this value because it is large enough to have better pairs and it is small enough not to always bring the best individual, which

is required for increasing variety among individuals, as we see in the experiments.

**Algorithm 2** Selection Operation

**Require:** Size of tournament population, $SIZE$ and population, $Pop$
**Ensure:** an individual chromosome

```
1: procedure SELECTION
2:     bestScore ← 0
3:     bestInd ← Null
4:     for i = 1 to SIZE do
5:         randomly select an individual chromosome from Pop, newInd
6:         calculate its fitness score, newScore
7:         if bestScore ≤ newScore then
8:             bestScore ← newScore
9:             bestInd ← newInd
10:        end if
11:    end for
       return bestInd
12: end procedure
```

### 3.1.6 Mutation Operation

The mutation operation randomly selects a virtual machine and a physical machine, removes that virtual machine from its current physical machine, and places it to the newly selected physical machine. If this migration would cause overdemand on the newly selected physical machine, then the mutation operation is cancelled. Algorithm 3 describes in detail how mutation works. Mutation operation is essential for the methods that use genetic algorithms as it increases variety among offspring population.

### 3.1.7 The Genetic Algorithm

The proposed genetic algorithm works very similar to the traditional genetic algorithm. The flowchart of the algorithm can be seen in Figure 3.2. We start with an initial population. Then, inside the generation loop, we start with calculating fitness value of each individual followed by selecting another individual for each one

**Algorithm 3** Mutation Operation

**Require:** One chromosome: $C_1$
**Ensure:** One mutated chromosome: $C_2$
1: **procedure** MUTATION
2: $C_2 \leftarrow C_1$
3: randomly select a virtual machine $v$; where $1 \leq v \leq |V|$
4: randomly select a physical machine $p$; where $1 \leq p \leq |P|$
5: remove $v$ from its current physical machine
6: add $v$ to $p$
**return** $C_2$
7: **end procedure**

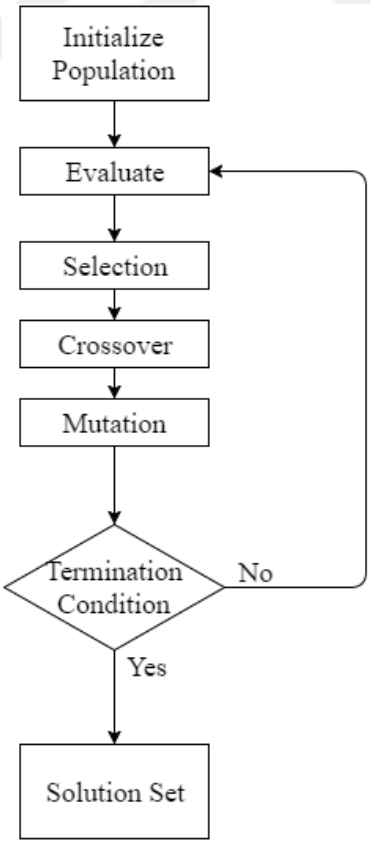


Figure 3.2: Flowchart of genetic algorithm.

to pair up to produce a new offspring. After that, we apply mutation operation if the randomly produced mutation value is less than the desired mutation rate. Moreover, we find the best individual among the new population and compare it with the current best one. If the new population's best individual is better than the current best individual, we set the new one as the current best. For the next iteration of the generation loop we continue with the new generation and discard the old one since the new generation consists of better individuals than the old population. The pseudo-code of our main algorithm is given in Algorithm 4.

**Algorithm 4** Utilization Based Genetic Algorithm - UBGA

**Require:** VM demands, PM resources, VM Network Matrix
**Ensure:** Assignment of VMs to PMs

```
1: procedure The GA
       generate a population of POPSIZE number of random individuals, POP;
2:     while THE TERMINATION CONDITION is not true do
3:         for each individual i in POP do
4:             calculate its fitness value f(i)
5:         end for
6:         for each Individual i in POP do
7:             invoke the Selection Operation, that is using tournament selection technique to select another individual to pair
8:         end for
9:         for each pair of parents do
10:            use Uniform Crossover Operation to produce an offspring
11:        end for
12:        for each offspring do
13:            apply Mutation Operation according to mutation rate
14:        end for
15:        find the best individual among offsprings, newBest
16:        if newBest is better than current best individual then
17:            replace current best individual with newBest
18:        end if
19:    end while
       return best individual
20: end procedure
```

## 3.2 Online Virtual Machine Placement

We also propose a dynamic virtual machine placement algorithm called DUBGA (Dynamic UBGA) based on UBGA that handles the case in which the number of virtual machines is not fixed but increases in time. Therefore, the placement algorithm has to place newly arrived virtual machines in a way as optimum as possible.

Our proposed algorithm, DUBGA, is an extension of UBGA. We use the same genetic algorithm idea described in Algorithms 1, 2, 3, and 4 to find a close-to optimal placement.

In the dynamic virtual machine placement, we have set of virtual machines that are already placed and a set of virtual machines that are to be placed. Resources of physical machines we have are wasted parts of physical machines in use and idle machines. Therefore, we only consider these resources when new virtual machines arrive to be placed. For example, suppose that we have 3 physical machines with 5 GB memory each and one of them has virtual machines using 2 GB, other has virtual machines using 4 GB and the last one is idle. When a set of virtual machines arrive, we consider that those physical machines have 3 GB, 1 GB and 5 GB memory, respectively. This approach is valid for the other resource types, bandwidth and CPU.

To ensure that newly arrived virtual machines cause the least number of physical machines to start (i.e., change its state from idle to in use), we modified our fitness function for dynamic allocation as in Eq. 3.17. We have a new value $s$, that is the number of physical machines in use which was idle before new virtual machines arrived.

The values in Eqs. 3.17, 3.18, 3.19, and 3.20 are for demands of new virtual machines and available resources of physical machines.

Since the dynamic allocation has to be realized fast in real life, we decreased some values of the genetic algorithm. The number of generations, and population

size are very small compared to the values they have in the offline placement.

$$fitness(i) = (k + s + 1) * (\frac{1}{T_{cpu}} * \sum_{p_j \in P_{all}} (-w_j^{cpu} * p_j^{cpu}) + \frac{1}{T_{mem}} * \sum_{p_j \in P_{all}} (-w_j^{mem} * p_j^{mem}) + \frac{1}{T_{bw}} * \sum_{p_j \in P_{all}} (-w_j^{bw} * p_j^{bw})) \quad (3.17)$$

where;

$$T_{cpu} = \sum_{p_j \in P_{all}} p_j^{cpu} \quad (3.18)$$

$$T_{mem} = \sum_{p_j \in P_{all}} p_j^{mem} \quad (3.19)$$

$$T_{bw} = \sum_{p_j \in P_{all}} p_j^{bw} \quad (3.20)$$

## 3.3 Chapter Summary

In this chapter, we represented our problem with a formal description. We showed our algorithm's aims and constraints of the problem. Then, we described our algorithm by showing its chromosome structure and fitness function. We defined selection, crossover and mutation operations of our algorithm. We showed how the overall genetic algorithm works. Finally, we extended offline virtual machine algorithm to handle online virtual machine placement.

# Chapter 4

# Utilization Based Simulated Annealing - UBSA

As stated in the previous chapter, optimization in cloud systems is very important for both cloud owners and customers as more optimized systems consume less energy, provide better quality of service, and support more customers. Virtual machine placement is the key component of optimization in cloud systems.

In this chapter, we propose a new algorithm for virtual machine placement problem, that uses simulated annealing metaheuristic algorithm. Similar to UBGA, this algorithm aims to reduce resource waste, energy consumption and network cost in datacenters. Wasted resource definition is the same as that in the previous chapter, i.e., resources of idle physical machines are not considered as wasted.

We have the same problem description as in Section 4.1.1. Therefore, constraints in that section are still valid.

## 4.1 Candidate Solution Representation

A particular assignment of virtual machines to physical machines is considered a possible solution and is referred as a system *state* or *configuration*. The topology of the network connecting physical machines is assumed to be a tree. We decided to use a tree data structure to represent such an assignment. Internal nodes of the tree represents network switches and leaves of the tree represent physical machines. There is a single pointer from each leaf to a list of virtual machines which are assigned to the physical machines represented by that leaf, as can be seen in Figure 3.1.

## 4.2 Neighbor Generator

Simulated annealing algorithm tries to find better configurations for the given problem. Therefore, we need a neighbor generator method that finds those better configurations to avoid searching the whole solution space. A new neighbor state is defined as a random change of a random number (between 1 and 5 in our proposed method) of virtual machines and their assigned physical machines. This change considers whether the new configuration satisfies resource limitations of physical machines, i.e., assigned virtual machines cannot demand more than what physical machines provides. If this is the case, we assign newly migrated virtual machine to a different physical machine. Therefore, we always satisfy resource limitations and avoid impossible placements.

## 4.3 Objective Function

For the algorithm, we need a function that evaluates how good a configuration is. In simulated annealing metaheuristic algorithm, this objective function calculates *energy* of the configuration (different from the specific *energy* definition,we use in our problem statement in Section 4.1). To avoid the confusion, we call the

output of the objective function as *cost* instead of as *energy*.

Our objective function is very similar to the fitness function in the previous chapter and can be seen in Eq. 4.1. Here, we have $0 \leq k \leq n$, where $n$ is the number of physical machines in the datacenter and $k$ is the number of over-demanded physical machines.

$$cost = (k+1) * (\frac{1}{T_{cpu}} * \sum_{p_j \in P_{all}} (w_j^{cpu} * p_j^{cpu}) + \frac{1}{T_{mem}} * \sum_{p_j \in P_{all}} (w_j^{mem} * p_j^{mem}) + \frac{1}{T_{bw}} * \sum_{p_j \in P_{all}} (w_j^{bw} * p_j^{bw}) + \frac{\sum_{v_i \in V} \sum_{v_j \in V} N_{ij}}{N_{max}}) \quad (4.1)$$

where;

$$T_{cpu} = \sum_{p_j \in P_{all}} p_j^{cpu} \quad (4.2)$$

$$T_{mem} = \sum_{p_j \in P_{all}} p_j^{mem} \quad (4.3)$$

$$T_{bw} = \sum_{p_j \in P_{all}} p_j^{bw} \quad (4.4)$$

$$N_{max} = H * \sum_{v_i \in V} \sum_{v_j \in V} F_{ij},\ where\ i < j \quad (4.5)$$

As can be seen in Eqs. 4.1 through 4.5, we do not penalize configurations for idle machines. We already avoid over-demand in the neighbor generator as explained in the previous section, however, the objective function also considers this in any case. $H$ in Eq. 4.5 denotes the maximum possible number of switches between two virtual machines.

## 4.4 Acceptance Criterion

While generating a new configuration for the placement problem, we need an acceptance criterion whether a new configuration is created after the neighbor generator operation or the currently selected one should be selected as the next state of the algorithm. Algorithm 5 shows the acceptance criterion. If the cost of new configuration is less than the current one, we pass to the new configuration. Otherwise, it depends on the given value in the general simulated annealing method.

**Algorithm 5** Acceptance Criteria

**Require:** Current Best Cost: $S_{cur}$, New Cost: $S_{new}$, Temperature: $T$
**Ensure:** Acceptance Probability: $P$

```
1: procedure ACCEPTANCE
2:     if S_new < S_cur then
3:         P ← 1.0
4:     else
5:         P ← e^((S_cur − S_new)/T)
6:     end if
       return P
7: end procedure
```

## 4.5 Temperature Scheduling

The temperature scheduling is a key aspect of the simulated annealing algorithm. It basically determines how many iterations the metaheuristic algorithm will have. How this operation is realized is important because an unsuitable way may lead to quick quenching. [28]. Our temperature value starts from 1000 degrees and we decrease the temperature in each iteration by an integer random value between 0 and 5 degrees, as in [45].

## 4.6 Simulated Annealing

The simulated annealing algorithm we designed can be seen in Algorithm 6. The algorithm creates an initial placement configuration. Then, until the temperature decreases to 0, it creates neighbor configurations and compares them with the current solution. *Acceptance* function in the algorithm is explained in Section 5.4 and *DecreaseTemperature* method is explained in Section 5.5.

**Algorithm 6** Utilization Based Simulated Annealing - UBSA

**Require:** VM demands, PM resources, VM Network Matrix
**Ensure:** Assignment of VMs to PMs

```
1:  procedure SIMULATED ANNEALING
2:      Generate a Initial Random Assignment, Init
3:      Best ← Init
4:      Calculate best cost, BestCost
5:      Temperature ← 1000
6:      while Temperature > 0 do
7:          Curr ← Best
8:          Generate a new neighbor solution, New
9:          Calculate current cost, CurrCost
10:         Calculate new cost, NewCost
11:         r ← Random value between 0 and 1
12:         if Acceptance(CurrCost, NewCost, Temperature) > r then
13:             Curr ← New
14:             CurrCost ← NewCost
15:         end if
16:         if CurrCost < BestCost then
17:             Best ← Curr
18:             BestCost ← CurrCost
19:         end if
20:         DecreaseTempereature
21:     end while
        return Best
22: end procedure
```

## 4.7 Chapter Summary

In this chapter, we proposed a simulated annealing method for virtual machine placement. We started with defining a candidate solution representation for our algorithm which is the same as genetic representation in the previous chapter. Then, we defined neighbor generator and objective function of our algorithm. We described the acceptance criterion that determines whether the algorithm should pass to new state. We showed temperature scheduling algorithm and we conclude with overall simulated annealing method.

# Chapter 5

# Evaluation

To evaluate algorithms, we conducted extensive simulation experiments by using CloudSim v.3.0.3, which is an open-source cloud environment simulation framework implemented in Java. CloudSim has its own default virtual machine allocation policy. We extended this default policy to use the genetic algorithm library we created. Our library implements our proposed method.

## 5.1 CloudSim Integration

The integration of our method to CloudSim involves extending CloudSim's own virtual machine allocation method with our algorithm. We implemented our genetic algorithm in a library. Our algorithm uses tree data structure to represent a chromosome which has one-to-one mapping with the network and server interconnection topology of a datacenter where virtual machines are placed.

## 5.2 Offline Virtual Machine Placement

We compared our method, Utilization Based Genetic Algorithm (UBGA), with CloudSim's default virtual machine placement policy (which we call CloudSim in figures and in the rest of the paper), random allocation method (RA), First Fit Decreasing method (FFD), and the VMPGGA (Virtual Machine Placement Group Genetic Algorithm) proposed by Sookhtsaraei et al. [40]. We did comparisons by using the following metrics:

1. Wasted CPU
2. Wasted memory
3. Wasted bandwidth
4. Number of physical machines used
5. Energy consumed by physical machines
6. Network cost
7. Number of over-demanded physical machines

We created test cases to compare our UBGA algorithm with these methods. Information about the physical machine resource capacity values and virtual machine demand values for those test cases can be seen in Tables I, II and III.

In all our test cases, the number of virtual machines to be allocated is varied between 200 and 1000 with an increment of 200. The number of physical machines is fixed at 200.

In our genetic algorithm, the number of generations is set to 20 and the population size for each generation is kept constant and is set to 100. We chose these values because we observed in our experiments that any value greater than these does not significantly increase the fitness score of the best individual. That

means increasing generation count and population size does not improve the optimal solution much, and therefore we did not set the values of these parameters too large to reduce the running time of our proposed method.

The parameter values we used for the components of our algorithm described in the previous section are set as follows:

- In the crossover operation, we set $uniformRate$ to be 0.5. This means an offspring has equal chance for inheriting from its parents. As stated above, this value is selected for unbiased gene selection from parents to improve variety.
- In the selection operation, we set tournament population size to 5, which is 5% of the population size.
- In the genetic algorithm, the termination condition is reaching the generation 20. We see in the experiments that this value is large enough, since use of higher number of generations does not significantly change the fitness score of the best individual.
- In the genetic algorithm, mutation rate is set to be 0.02. Typical values used for mutation rate in literature are below 0.05, as can be seen in [40] and [44].

### 5.2.1 Homogeneous Distribution

In homogeneous distribution case, we set the capacities of physical machines and demands of virtual machines as in Table 5.3. Each physical machine has the same initial amount of resources and each virtual machine has the same amount of demand for each type of resource.

When compared in terms of wasted resource amount, UBGA and FFD algorithms provide similar results and are the best ones. VMPGGA is slightly worse than these two. Random allocation and CloudSim's default allocation policy are

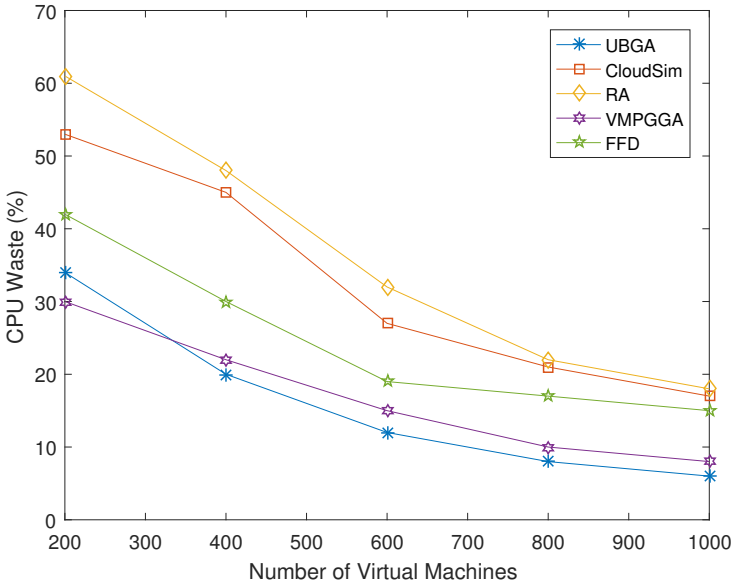


Figure 5.1: The result of CPU wastage with uniform distribution.

far behind. The reason of superior performance of FFD is that this case is the optimum scenario for that method. Both UBGA and FFD achieve placing virtual machines to the least number of physical machines. Of the $m$ physical machines that might be used for the given set of VMs, the first $m-1$ machines are fully utilized and the last machine is partially utilized and is the only one that contributes to the waste of a particular resource type.

### 5.2.2 Uniform Distribution

In uniform distribution case, resources demands of virtual machines are uniformly distributed within the value range shown in Table 5.1 and resource capacities provided by physical machines are uniformly distributed within the value range shown in Table 5.2.

The results of the uniform distribution case are shown in Figures 5.3 through 5.6. When we consider resource waste, we can say that UBGA is clearly superior to CloudSim's method, random allocation method and FFD method, as

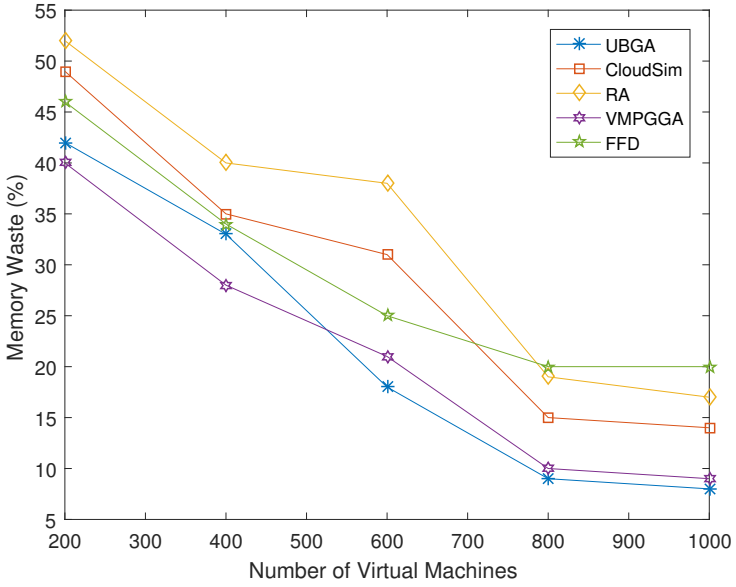


Figure 5.2: The result of memory wastage with uniform distribution.

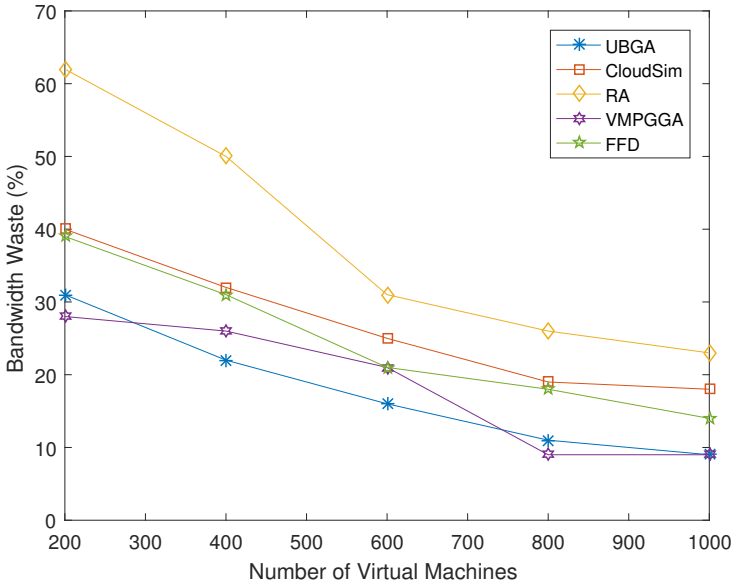


Figure 5.3: The result of bandwidth wastage with uniform distribution.

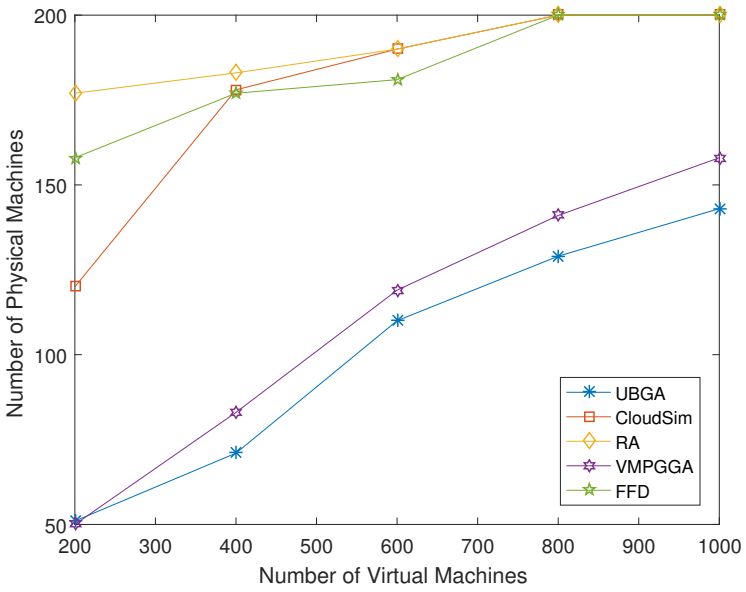


Figure 5.4: Number of PMs used with uniform distribution.

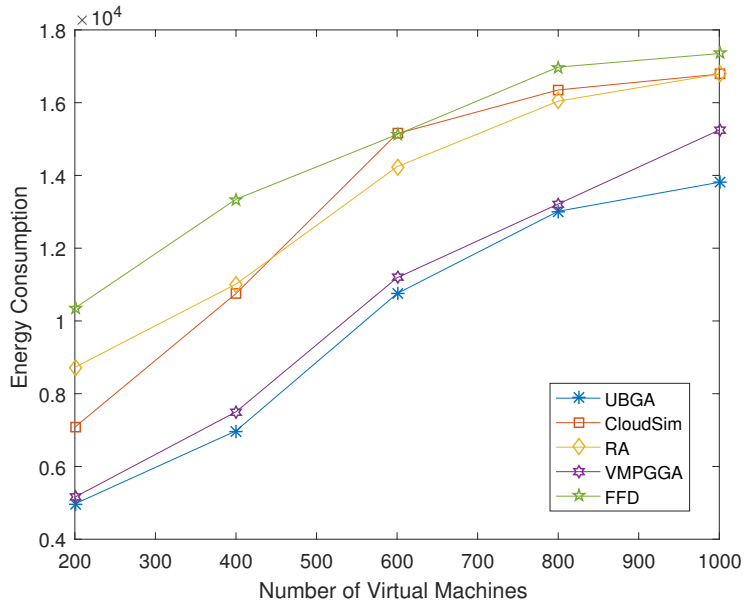


Figure 5.5: Energy consumption by PMs used with uniform distribution.

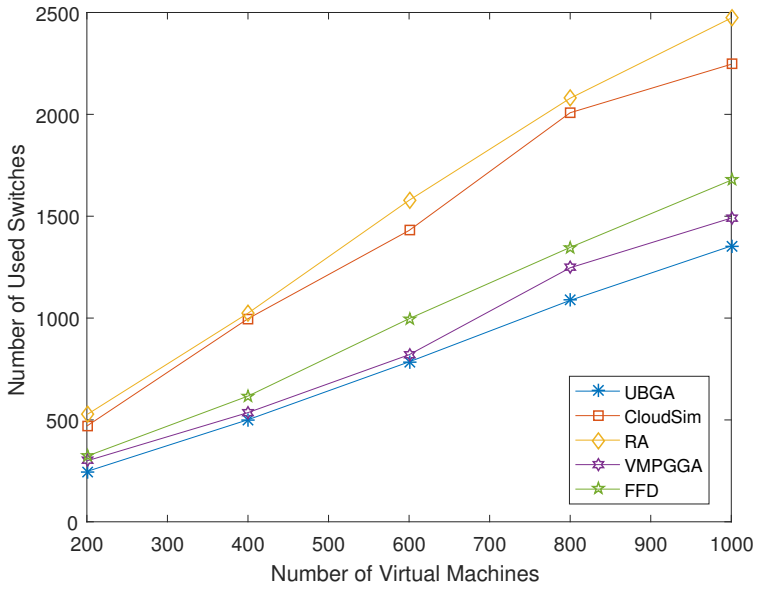


Figure 5.6: Network cost with uniform distribution.

they cause far more memory, CPU and bandwidth waste for five different virtual machine count cases, as shown in Figures 5.1 , 5.2, and 5.3, respectively. In comparison to VMPGGA [40], UBGA falls little bit behind for cases where virtual machine counts are 200 and 400. For the other three cases, on the average, UBGA is better than VMPGGA. The main reason behind this result is that UBGA is better optimized for large number of virtual machines.

Our UBGA method uses less number of physical machines for a given virtual machine set compared to the other four methods, as shown in Figure 5.4. UBGA tries to put virtual machines as close as possible in the network and since we do not penalize assignments for idle machines, UBGA tends to use least possible number of physical machines as well. However, the other four methods do not always try to produce solutions with the least number of physical machines.

For calculating energy consumption, we use the formula given in Eq. 3.10, where both utilization and number of idle machines are important. In this equation we use the values $E_j^{max} = 100$ and $E_j^{idle} = 10$, as they are CloudSim's default parameter values for energy calculation. To get a better result for this metric, our method utilizes as few physical machines as possible, leaving the others idle.

We have the advantage of using the least number of physical machines among the five methods for all five different virtual machine counts (Figure 5.4). Figure 5.5 shows the energy consumption results of the methods.

Network cost of a VM placement is considered as the total load incurred on the switches of the datacenter network due to communication happening among VMs. This total load is computed as follows. For each pair of VMs that communicate, the flow demand between these VMs is multiplied by the number of switches between the physical machines where these VMs are placed. In this way the load incurred by that pair of VMs is found. Then we sum the loads by all VM pairs and find out the total load incurred on the network, which we consider as the network cost of the placement. In our experiments, we assigned a random data flow demand between 0 and 4 traffic units to each virtual machine pair. We have $\binom{|V|}{2}$ pairs in total. In this scenario, the methods that allocate virtual machines having larger flow demand between them as close as possible in the cloud network topology produce better results. As can be seen in Figure 5.6, UBGA, VMPGGA, and FFD are clearly superior to the other two methods. There is a very small difference between the performance results of these top three methods. For small number of virtual machines to be placed, the three methods have very close results. For large number of virtual machines, UBGA performs slightly better than the others (Figure 5.6). The reason why UBGA produces the best result in terms of network cost is that it also incorporates network cost in the fitness function, while other methods do not.

### 5.2.3 Random Distribution

In the last case considered in our experiments, resource demands of virtual machines are randomly distributed within the value range shown in Table 5.1 and the resource capacities provided by physical machines are randomly distributed within the value range shown in Table 5.2.

The results for the random distribution of resource capacities and demands are shown in Figures 5.7 through 5.12. For resource waste metrics, random allocation

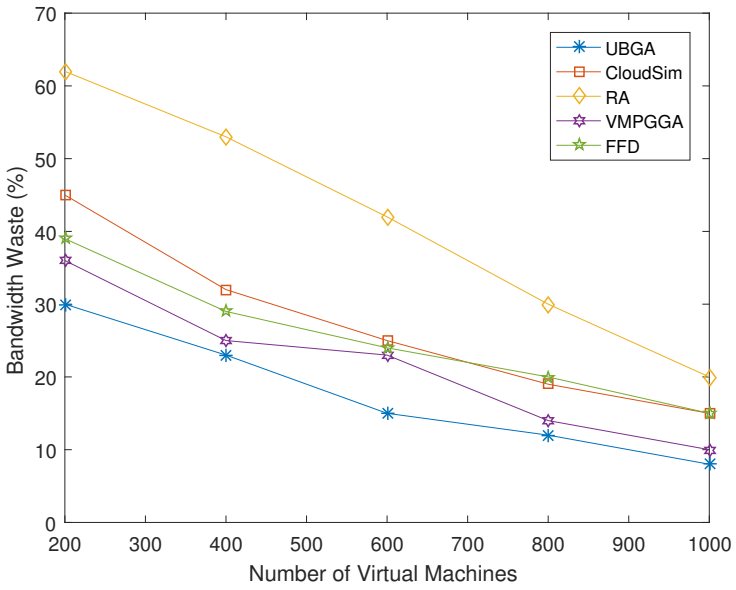


Figure 5.7: The result of bandwidth wastage with random distribution.

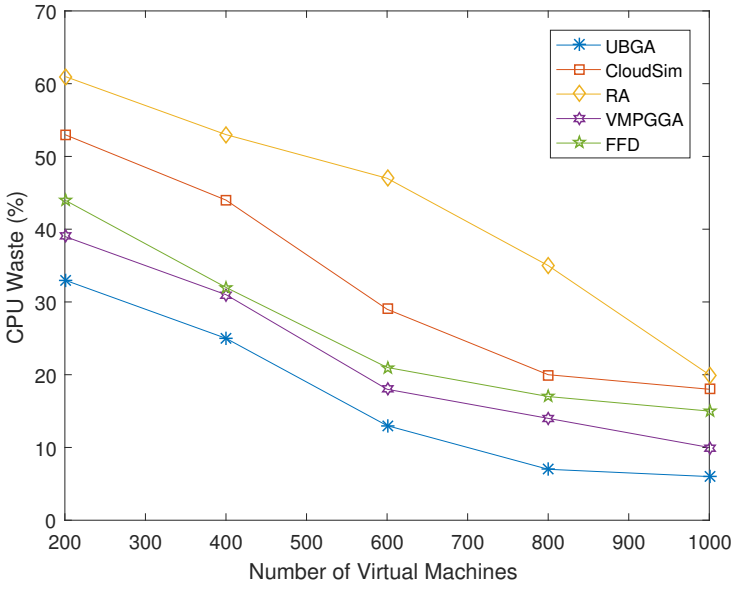


Figure 5.8: The result of CPU wastage with random distribution.

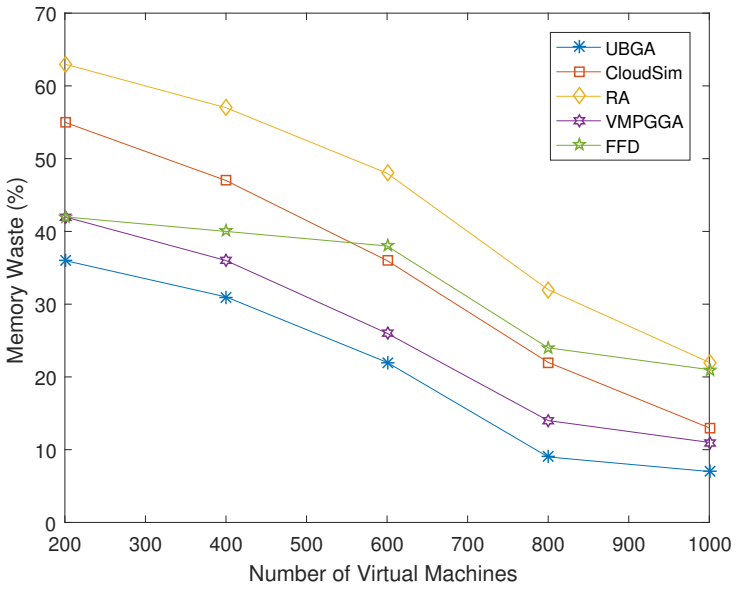


Figure 5.9: The result of memory wastage with random distribution.

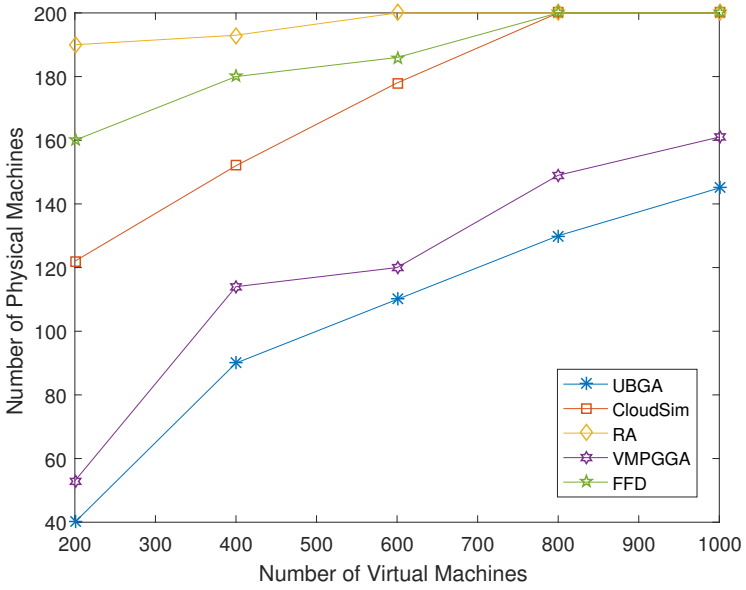


Figure 5.10: Number of PMs used with random distribution.

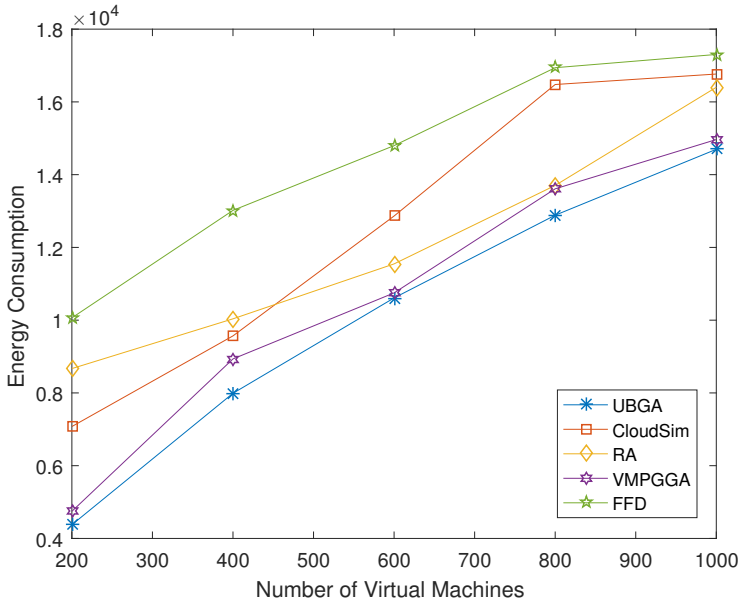


Figure 5.11: Energy consumption by PMs used with random distribution.

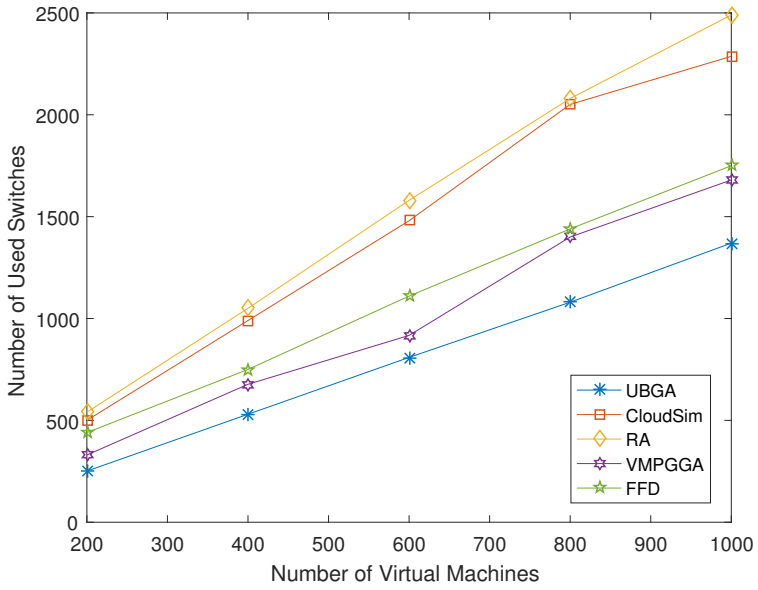


Figure 5.12: Network cost with random distribution.

method has the worst performance as in the previous case. CloudSim's default policy and FFD are slightly better than the random one, but still they do not provide the best solution. Performance of VMPGGA in terms of resources wasted slightly decreases with respect to the uniform distribution case, while UBGA keeps its performance as in the uniform case (Figures 5.7, 5.8, 5.9). This difference may be caused by the fact that VMPGGA is optimized for uniform resource distribution while we tried to optimize our method for all types of distributions.

In terms of the number physical machines used, energy consumption of physical machines and network cost, we have similar results as in the uniform distribution case. The methods having optimization concern provide better performance than the ones which do not have such a concern (Figures 5.10, 5.11, and 5.12). Again, for these three metrics UBGA maintains its performance, while VMPGGA's performance decreases slightly. The good performance of UBGA is again due to its aim for keeping more number of machines idle. Again, network cost is also targeted by our fitness function and that is the reason for why UBGA is the best method in the experiments.

### 5.2.4 Comparison with UBSA

We compared our genetic algorithm method (UBGA), also with our simulated annealing method (UBSA). We used the same test cases and parameter values: we have 200 physical machines and thenumber of virtual machines is between 200 and 1000 with an increment of 200; moreover, the demands of virtual machines and resources of physical machines are selected from Tables 5.1 and 5.2 for Uniform Distribution and Random Distribution, respectively.

#### 5.2.4.1 Uniform Distribution

In the uniform distribution case, UBGA and UBSA show very similar performance for all of the evaluation metrics. On average UBGA performs slightly better than UBSA. This similarity is caused by the similar design of two algorithms. Their

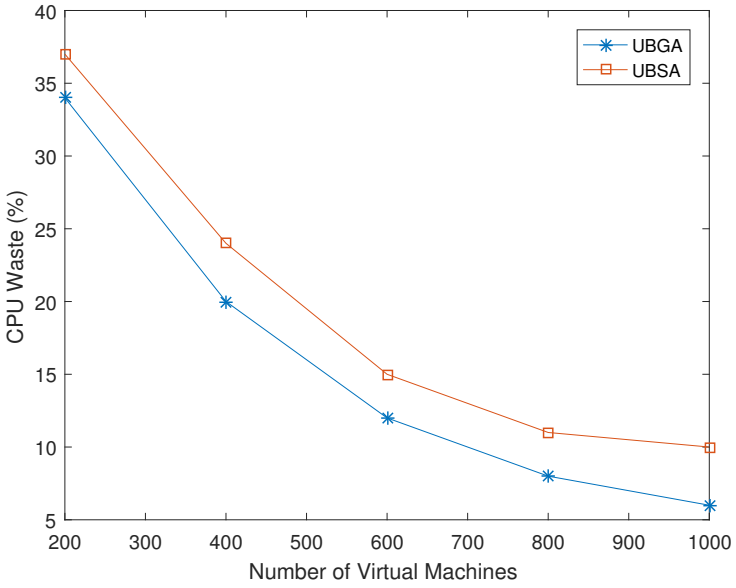


Figure 5.13: Comparison of UBGA and UBSA on CPU wastage with uniform distribution.

fitness and objective functions are almost the same except for one little detail. The summation part in the fitness function of the genetic algorithm has values between -4 and 0 while the values are between 0 and 4 in the objective function of the simulated annealing algorithm. Since we want larger outcome in the fitness function and smaller outcome in the objective function, the similarity in the results is understandable. Moreover, the configuration of the problem is handled via tree data structure in both algorithms. The results can be seen in Figures 5.13 through 5.18. The slightly better performance of UBGA may be caused by the fact that it creates more solutions at each iteration than UBSA. Hence, UBGA has more chance to find a better solution compared to UBSA.

#### 5.2.4.2 Random Distribution

In the random distribution case, again, our two algorithms have similar performance to each other. This time UBSA rarely has better performance than UBGA, which may be caused by randomly created different values for virtual machine demands and physical machine resources for each algorithm. The results is shown

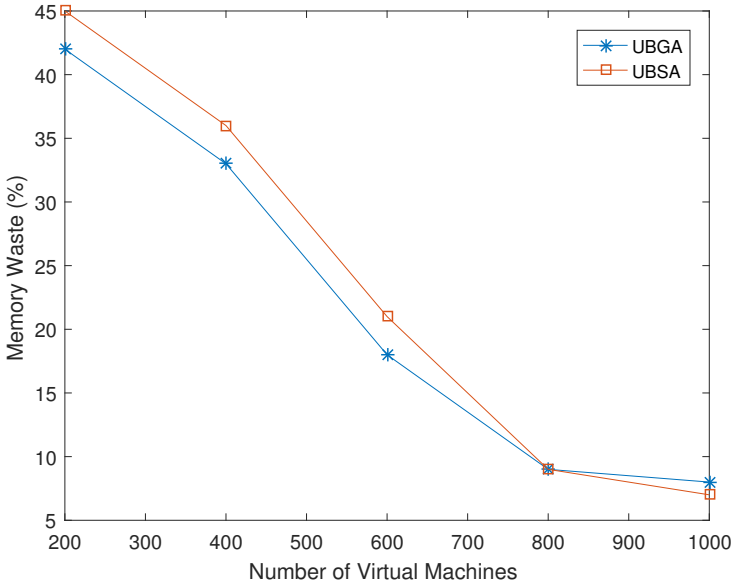


Figure 5.14: Comparison of UBGA and UBSA on memory wastage with uniform distribution.

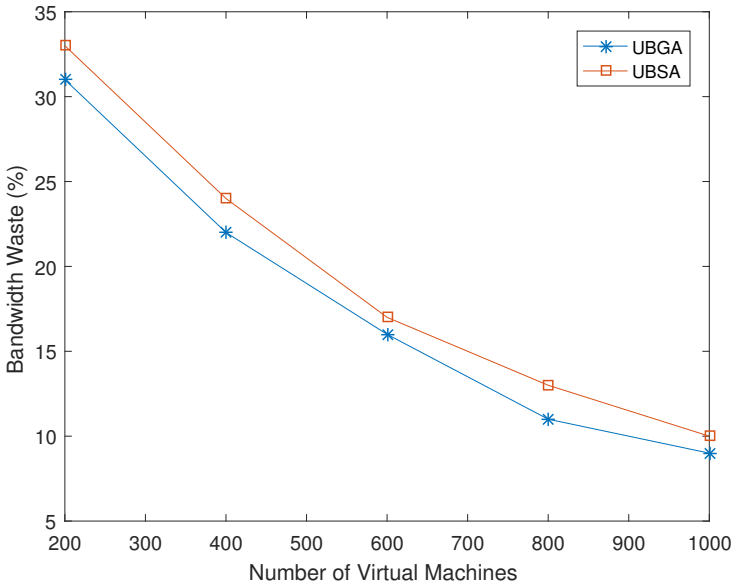


Figure 5.15: Comparison of UBGA and UBSA on bandwidth wastage with uniform distribution.

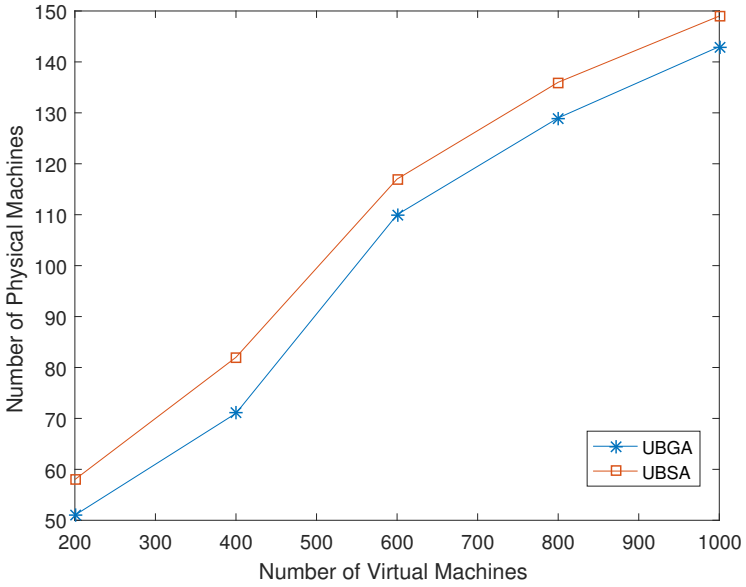


Figure 5.16: Comparison of UBGA and UBSA on number of PMs used with uniform distribution.

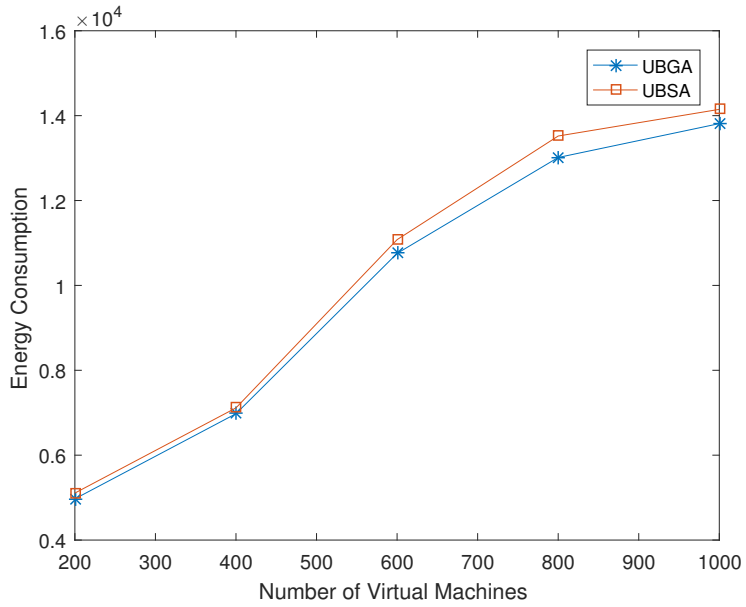


Figure 5.17: Comparison of UBGA and UBSA on energy consumption by PMs used with uniform distribution.

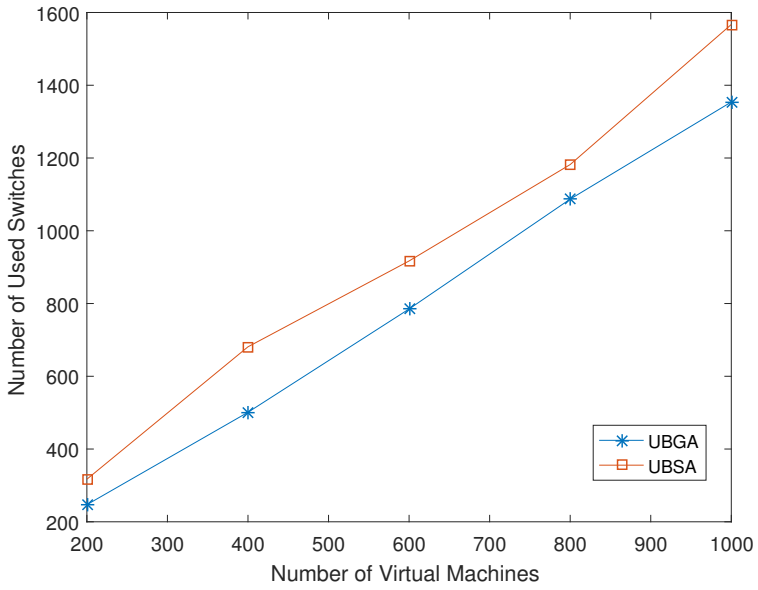


Figure 5.18: Comparison of UBGA and UBSA on network cost with uniform distribution.

in Figures 5.19 through 5.24.

## 5.3 Online Virtual Machine Placement

To evaluate the dynamic (online) virtual machine placement algorithm we proposed in Chapter 4.2, DUBGA, we only consider random distribution case of resource and demand values shown in Tables 5.1 and 5.2. We compared our dynamic allocation algorithm DUBGA, with our offline allocation algorithm UBGA, a random allocation method (RA) and First Fit Decreasing (FFD) method. While evaluating DUBGA against UBGA, when new virtual machines arrive, we allocate both new and old virtual machines together in UBGA. That is to say, we assume that every physical machine is idle and no virtual machine is allocated before we run UBGA again together with the new arrivals. We do this to see how close DUBGA is to the best allocation we can achieve with UBGA under the same conditions. For the random allocation (RA) method, if random placement of a newly arrived virtual machine causes over-demand, we randomly select a new

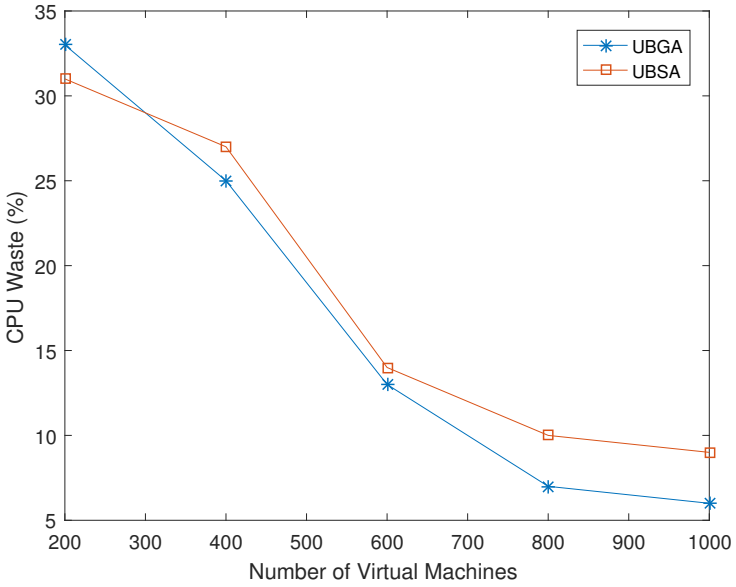


Figure 5.19: Comparison of UBGA and UBSA on CPU wastage with random distribution.

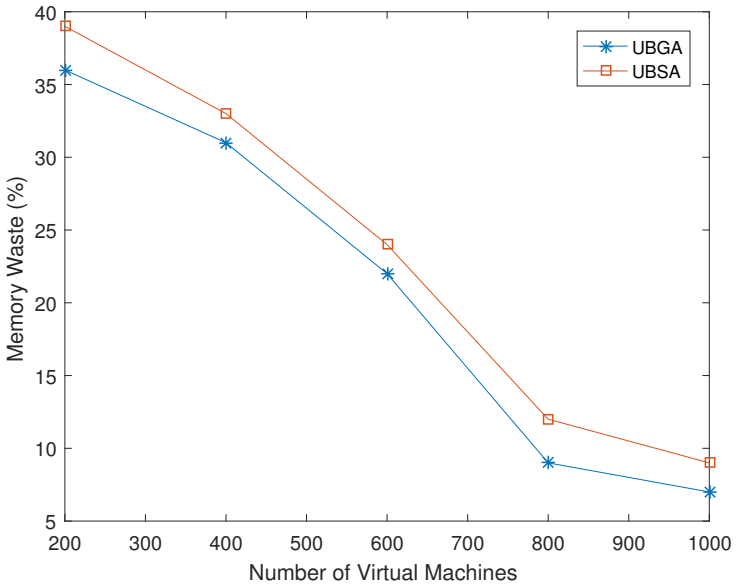


Figure 5.20: Comparison of UBGA and UBSA on memory wastage with random distribution.

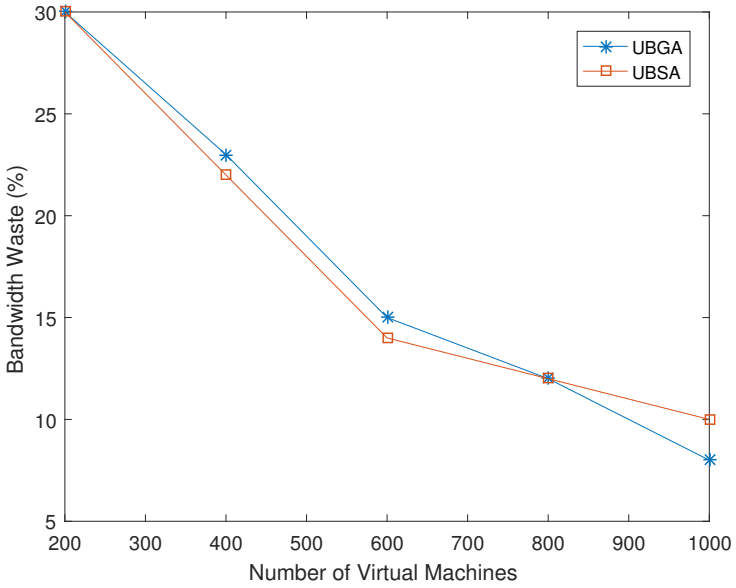


Figure 5.21: Comparison of UBGA and UBSA on bandwidth wastage with random distribution.

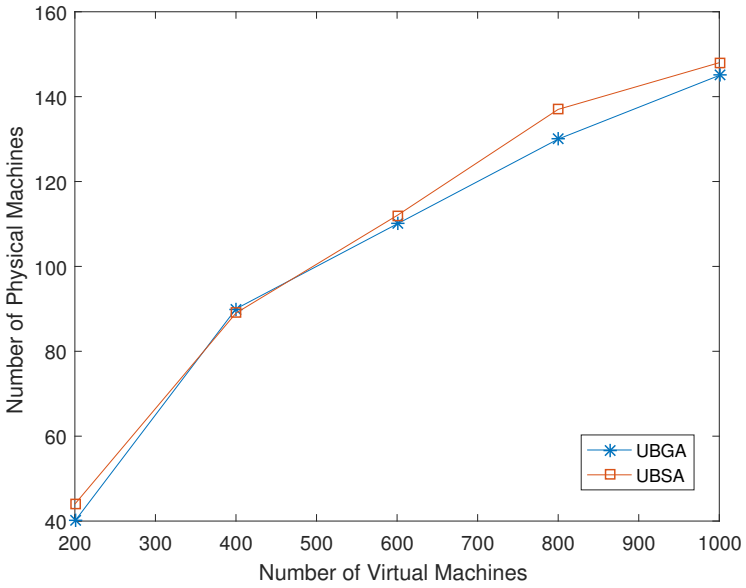


Figure 5.22: Comparison of UBGA and UBSA on number of PMs used with random distribution.

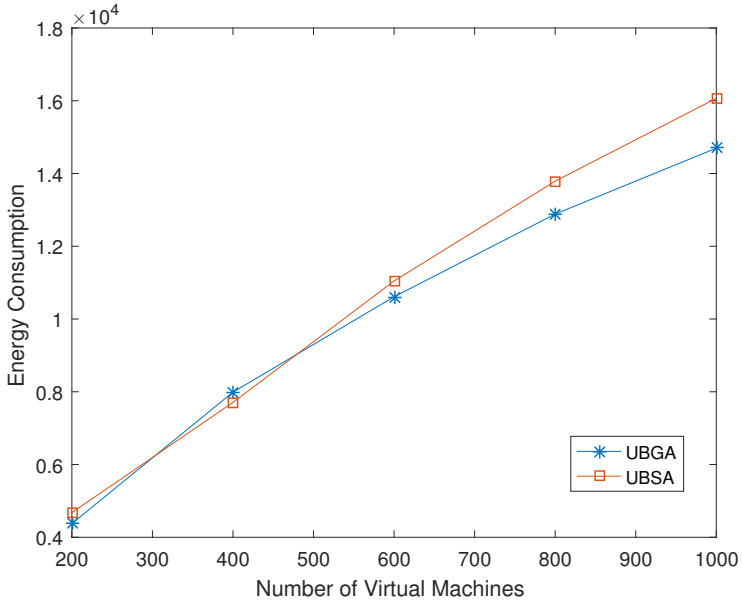


Figure 5.23: Comparison of UBGA and UBSA on energy consumption by PMs used with random distribution.

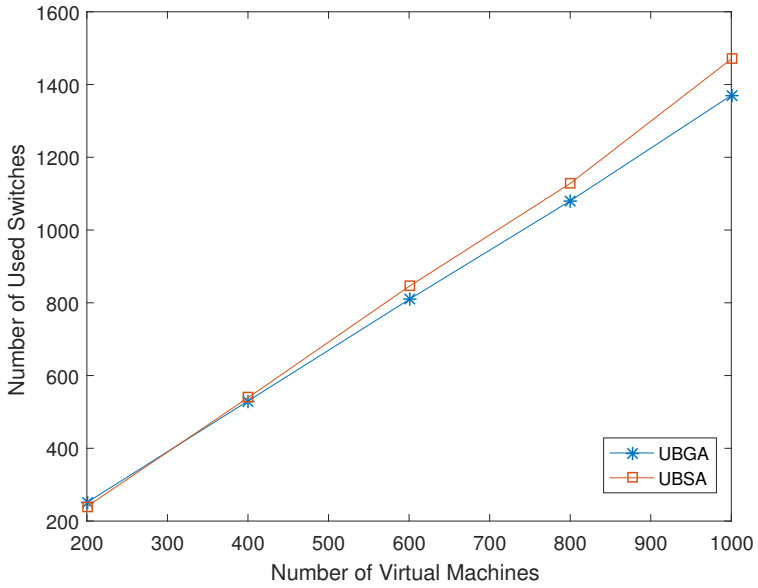


Figure 5.24: Comparison of UBGA and UBSA on network cost with random distribution.

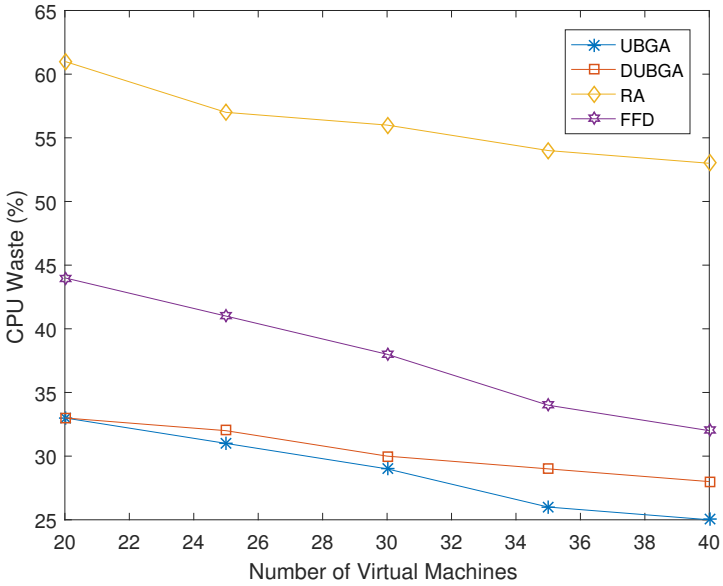


Figure 5.25: The result of CPU wastage in online distribution.

physical machine for that virtual machine again until we find a suitable one.

We created test cases with 20 physical machines. The number of virtual machines is initially 20. These 20 virtual machines are placed using UBGA, RA and FFD initially. Then we increase the number of virtual machines by 5 at each step until their number becomes 40.

The results provided in Figures 5.25, 5.26, and 5.27 shows that our dynamic allocation method (DUBGA) is superior to RA and FFD methods. The performance difference between UBGA and DUBGA shows how close DUBGA is to the most optimum placement solution we can have with our genetic algorithm UBGA. DUBGA performs slightly worse than UBGA and that performance difference does not decrease when new virtual machines arrive. The reason for small gap between our two methods is that we do not allow migration of the virtual machines since migration cost may be larger than the cost of extra wasted resources.

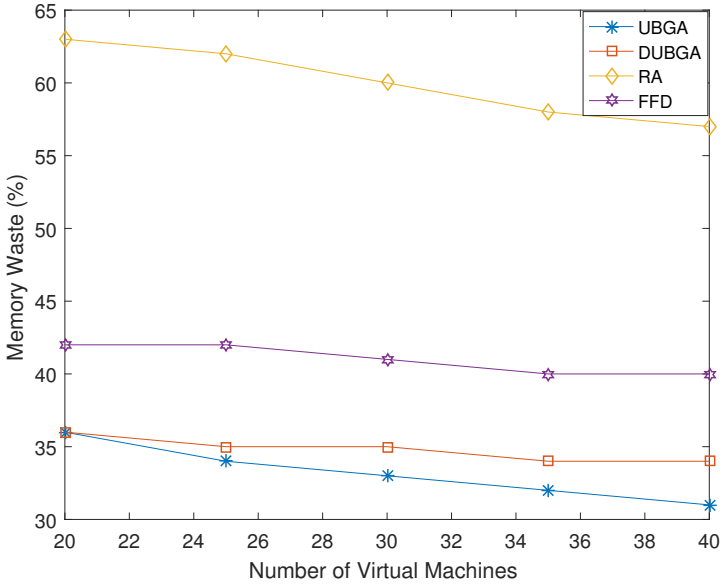


Figure 5.26: The result of memory wastage in online distribution.

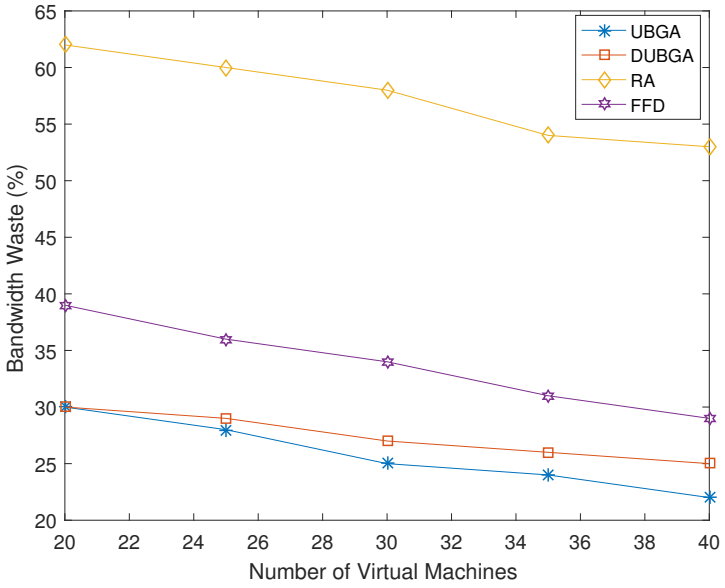


Figure 5.27: The result of bandwidth waste in online distribution.

## 5.4 Comparison with Linear Programming

To see how close our UBGA algorithm to an optimum solution is, we compared the results of UBGA with optimal solutions obtained via a linear programming model for each type of resource. To make use of linear programming we applied the method in [7] that optimizes bin packing problem for our case. Since our algorithm works better in random distribution case, we use that for evaluation.

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{m}(w_j * u_j) \\ \text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = 1, && i = 1, ..., n \\ & \sum_{i=1}^{n} d_i * x_{ij} \leq C_j && j = 1, ..., m \\ & x_{ij} \in \{0, 1\}, \qquad u_j \in \{0, 1\} \end{aligned} \tag{5.1}$$

Eq 5.1 shows of our linear programming design model. For each physical machine $j$, we have $w_j$ that shows wasted percentage of the corresponding resource and $u_j$ that shows whether physical machine $j$ is idle or not. A physical machine is idle when $u_j$ is 0. $x_{ij}$ is 1 when virtual machine $i$ is assigned to physical machine $j$. Therefore, the second line of the equation ensures that one virtual machine can only be assigned to one physical machine. $d_i$ is the demand of virtual machine $i$ and $C_j$ is the capacity of physical machine $j$ for the corresponding resource. $m$ is the number of physical machines and $n$ is the number of virtual machines to be assigned.

Since the run-time required to solve a linear program may be too long, we use small number of physical and virtual machines, i.e., *m = 20* and *n = 20, 40*. The results are shown in Table 5.4. The results show that UBGA produces very good performance since it considers all three resource constraints at the same time as well as the network cost of virtual machines, while the linear program takes the resources into account separately.

## 5.5 Summary

In this chapter, we compared two methods we created which are UBGA and UBSA with VMPGGA, a random allocation (RA) method and FFD method. Results show that our algorithm is superior to those methods in both uniform and random distribution of resources and demands. Moreover, we compared our dynamic allocation method (DUBGA) with UBGA, RA and FFD and we show that DUBGA is better than RA and FFD. We also compared UBGA with the results we have from a linear program model to show how close the solutions we find with UBGA to the optimum one and the results show that they are quite close to the optimum.

Table 5.1: Resource demand values of virtual machines.

| **Resource** | **Values** |
|---|---|
| CPU | 250 Mips - 1500 Mips |
| Memory | 500 MB - 5000 MB |
| Bandwidth | 500 Mbps - 1500 Mbps |

Table 5.2: Resources provided by physical machines.

| **Resource** | **Values** |
|---|---|
| CPU | 1000 Mips - 3000 Mips |
| Memory | 1 GB - 10 GB |
| Bandwidth | 1 Gbps - 5 Gbps |

Table 5.3: Resources and demands for homogeneous distribution case.

| **Resource** | **VM Demand** | **PM Resource** |
|---|---|---|
| CPU | 250 Mips | 2500 Mips |
| Memory | 500 MB | 5 GB |
| Bandwidth | 100 Mbps | 1 Gbps |

Table 5.4: Comparison of LP and UBGA

| **Resource** | **LP** | | **UBGA** | |
|---|---|---|---|---|
| **Waste (%)** | **#VM = 20** | **#VM = 40** | **#VM = 20** | **#VM = 40** |
| **CPU** | 19 | 14 | 33 | 25 |
| **Memory** | 17 | 13 | 36 | 31 |
| **Bandwidth** | 22 | 14 | 30 | 23 |

# Chapter 6

# Conclusion

How to place virtual machines in a datacenter considering multiple objectives and constraints is one of the important and challenging problems in virtualized cloud systems. In this paper we proposed a new genetic algorithm, called UBGA (Utilization Based Genetic Algorithm), that uses an innovative fitness function and chromosome structure, and that aims to increase server utilization, decrease resource waste, and reduce network overhead, all at the same time.

We evaluated the performance of our algorithm via extensive simulation experiments and compared it with four other approaches: Random allocation (RA), CloudSims default allocator (CloudSim), First fit decreasing method (FFD), and the VMPGGA algorithm from literature. Our results show that Random and CloudSim allocators perform worse. FFD provides an average performance for uniform and random capacity and demand distributions, and excellent performance for homogeneous distribution, which is its optimum case. For uniform case, VMPGGA performs better than our algorithm for small number of virtual machines. In all other scenarios, however, our algorithm performs better, especially in random distribution of capacity and demand values, in terms of all evaluation metrics.

We also compared UBGA with our simulated annealing method, UBSA. They

have very close performance since their designs are similar. We also used linear programming to determine how close the solutions UBGA produces to global optimum for each utilization metric. The results show that UBGA performs well.

For dynamic allocation case, we modified UBGA to create DUBGA and compared them with random allocation and First fit decreasing. DUBGA is superior to other two in terms of resource waste.

# Bibliography


[1] A. C. Adamuthe, E. M. Pandharpatte, and G. T. Thampi, *Multiobjective virtual machine placement in cloud environment*, Cloud and Ubiquitous Computing and Emerging Technologies (CUBE), pp. 8-13, 2013.

[2] M. Alicherry, and T Laksman, *Optimizing Data Access Latencies in Cloud Systems by Intelligent Virtual Machine Placement*, IEEE INFOCOM Conference, pp. 647-655, 2013.

[3] A. Anand, J. Lakshmi, and S. Nandy, *Virtual Machine Placement Optimization Supporting Performance SLAs*, The 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom), vol. 1, pp. 298-305, 2013.

[4] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, *Guaranteeing High Availability Goals for Virtual Machine Placement*, The 31st IEEE International Conference on Distributed Computing Systems (ICDCS), pp. 700-709, 2011.

[5] N. M. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti, *VM Placement Strategies for Cloud Scenarios*, The 5th IEEE International Conference on Cloud Computing (CLOUD), pp. 852-859, 2012.

[6] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Software-Practice and Experience, vol. 42, pp. 23-50, 2011.

[7] H. Cambazard, and B. O'Sullivan, *Propagating the bin packing constraint using linear programming*, The 16th International Conference of Principles Practice Constraint Programming, vol. 6308 pp. 129-141, 2010.

[8] Z. Cao, and S. Dong, *An energy-aware heuristic framework for virtual machine consolidation in cloud computing*, The Journal of Supercomputing, pp. 1-23, 2014.

[9] E. Caron, A. D. Le, A. Lefray, and C. Toinard, *Definition of Security Metrics for the Cloud Computing and Security-Aware Virtual Machine Placement Algorithms*, The 5th IEEE International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), pp. 125-131, 2013.

[10] S. Chaisiri, B.-S. Lee, and D. Niyato, *Optimal Virtual Machine Placement across Multiple Cloud Providers*, The 6th IEEE Asia-Pacific Conference on Services Computing, pp. 103-110, 2009.

[11] K. Chen, Y. Xu, K. Xi, and H. J. Chao, *Intelligent virtual machine placement for cost efficiency in geo-distributed cloud systems*, The 23rd IEEE International Conference on Communications (ICC), pp. 3498-3503, 2013.

[12] L. Cui, J. Zhang, L. Yue, Y. Shi, and D. Yuan, *A Genetic Algorithm Based Data Replica A Genetic Algorithm Based Data Replica Placement Strategy for Scientific Applications in Clouds*, IEEE Transactions on Services Computing, to appear, 2016.

[13] S. Fang, R. Kanagavelu, B.-S. Lee, C. H. Foh, and K. M. M. Aung, *Power-Efficient Virtual Machine Placement and Migration in Data Centers*, IEEE International Conference on Green Computing and Communications (GreenCom) and IEEE Internet of Things(iThings) and IEEE Cyber, Physical and Social Computing(CPSCom), pp. 1408-1413, 2013.

[14] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, *Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers*, Computer Networks, vol. 57, no. 1, pp. 179-196, 2013.

[15] 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.

[16] S. Georgiou, K. Tsakalozos, and A. Delis, *Exploiting network-topology awareness for vm placement in iaas clouds*, The 3rd IEEE International Conference on Cloud and Green Computing (CGC), pp. 151-158, 2013.

[17] H. Goudarzi, and M. Pedram, *Energy-efficient virtual machine replication and placement in a cloud computing system*, The 5th IEEE International Conference on Cloud Computing (CLOUD), pp. 750-757, 2012.

[18] J. H. Holland, *Adaptation in Natural and Artificial Systems*, First Edition, MIT Press.

[19] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, *QoE-Aware Virtual Machine Placement for Cloud Games*, The 12th IEEE Annual Workshop on Network and Systems Support for Games (NetGames), pp. 1-2, 2013.

[20] W. Huang, X. Li, and Z. Qian, *An Energy Efficient Virtual Machine Placement Algorithm with Balanced Resource Utilization*, The 7th IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 313-319, 2013.

[21] Z. Huang, D. H. K. Tsang, *SLA Guaranteed Virtual Machine Consolidation for Computing Clouds*, The 22nd IEEE International Conference on Communications (ICC), pp. 1314-1319, 2012.

[22] A.Ilkhechi, I. Korpeoglu, O. Ulusoy, *Network-Aware Virtual Machine Placement in Cloud Data Centers with Multiple Traffic-Intensive Components*, Computer Networks, vol. 91, pp. 508-527, 2015.

[23] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, *Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-Aware Virtual Machine Placement*, The 8th IEEE International Conference on Services Computing (SCC), pp. 72-79, 2011.

[24] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, *Joint VM placement and routing for data center traffic engineering*, IEEE INFOCOM Conference, pp. 2876-2880, 2012.

[25] H. Jin, D. Pan, J. Xu, and N. Pissinou, *Efficient vm placement with multiple deterministic and stochastic resources in data centers*, IEEE Global Communications Conference (GLOBECOM), pp. 2505-2510, 2012.

[26] C. T. Joseph, K. C. Sekaran, and R. Cyriac, *A Novel Family Genetic Approach for Virtual Machine Allocation*, International Conference on Information and Communication Technologies, vol. 46, pp. 558-565, 2015.

[27] B. Kantarci, L. Foschini, A. Corradi, and H. T. Mouftah, *Inter-and-intra data center VM-placement for energy-efficient large-Scale cloud systems*, IEEE Globecom Workshops, pp. 708-713, 2012.

[28] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science, vol. 220, p. 10, 1983.

[29] N. Kord and H. Haghighi, *An energy-efficient approach for virtual machine placement in cloud based data centers*, The 5th IEEE International Conference on Information and Knowledge Technology, pp. 44-49, 2013.

[30] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen, *Reducing electricity cost through virtual machine placement in high performance computing clouds*, ACM International Conference for High Performance Computing, Networking, Storage and Analysis, p. 22-33, 2011.

[31] K. Li, J. Wu, and A. Blaisse, *Elasticity-aware virtual machine placement for cloud datacenters*, The 2nd IEEE International Conference on Cloud Networking (CloudNet), pp. 99-107, 2013.

[32] J.-W. Lin and C.-H. Chen, *Interference-aware virtual machine placement in cloud computing systems*, IEEE International Conference on Computer and Information Science (ICCIS), vol. 2, pp. 598-603, 2011

[33] B. Madhusudhan, and K. C. Sekaran, *A Genetic Algorithm Approach for Virtual Machine Placement in Cloud*, International Conference on Emerging

Research in Computing, Information, Communication and Applications (ERCICA), pp. 115-122, 2013.

[34] X. Meng, V. Pappas, and L. Zhang, *Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement*, IEEE INFOCOM Conference, pp. 1-9, 2010.

[35] M. Mitchell, *An Introduction to Genetic Algorithms*. Fifth edition, MIT Press.

[36] I. S. Moreno, R. Yang, J. Xu, and T. Wo, *Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement*, The 11th IEEE International Symposium on Autonomous Decentralized Systems (ISADS), pp. 1-8, 2013.

[37] J. T. Piao and J. Yan, *A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing*, The 9th IEEE International Conference on Grid and Cooperative Computing, pp. 87-92, 2010.

[38] W. Shi and B. Hong, *Towards Profitable Virtual Machine Placement in the Data Center*, The 4th IEEE International Conference on Utility and Cloud Computing (UCC), pp. 138-145, 2011.

[39] N. A. Singh, and M. Hemalatha, *Reduce energy consumption through virtual machine placement in cloud data centre*, Mining Intelligence and Knowledge Exploration, Springer, pp. 466-474, 2013.

[40] R. Sookhtsaraei, M. Madani, A. Kavian, *A multi objective virtual machine placement method for reduce operational costs in cloud computing by genetic*, International Journal of Computer Networks and Communications Security, vol. 2, no. 8, pp. 250-259, 2012.

[41] B. Speitkamp, and M. Bichler, *A Mathematical Programming Approach for Server Consolidation Problems in Virtualized Data Centers*, IEEE Transactions on Services Computing, vol. 3, no. 4, pp. 266-278, 2010.

[42] S. Srikantaiah, A. Kansal, and F. Zhao, *Energy aware consolidation for cloud computing*, Workshop on Power Aware Computing and Systems, pp. 10-10, 2008.

[43] M. Sun, W. Gu, X. Zhang, H. Shi, and W. Zhang, *A Matrix Transformation Algorithm for Virtual Machine Placement in Cloud*, The 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 1778-1783, 2013.

[44] G. Wu, M. Tang, Y. Tian and W. Li, *Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm*, International Conference on Neural Information Processing, pp. 315-323, 2012.

[45] Y. Wu, M. Tang, and W. Fraser, *A Simulated Annealing Algorithm for Energy Efficient Virtual Machine Placement*, IEEE International Conference on Systems, Man, and Cybernetics, pp. 1245-1250, 2012.