OPTIMIZING ENERGY CONSUMPTION OF CLOUD COMPUTING IaaS

by

Huda Ibrahim Mohamed

A Thesis presented to the Faculty of the
American University of Sharjah
College of Engineering
In Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Computer Engineering

Sharjah, United Arab Emirates

May 2017

Approval Signatures

We, the undersigned, approve the Master's Thesis of Huda Ibrahim Mohamed.

Thesis Title: Optimizing Energy Consumption of Cloud Computing IaaS.

Signature	**Date of Signature** (dd/mm/yyyy)
_____________________________ Dr. Raafat Aburukba Assistant Professor, Department of Computer Science and Engineering Thesis Advisor	_______________
_____________________________ Dr. Khaled El-Fakih Associate Professor, Department of Computer Science and Engineering Thesis Co-Advisor	_______________
_____________________________ Dr. Rana Ahmed Associate Professor, Department of Computer Science and Engineering Thesis Committee Member	_______________
____________________________ Dr. Mostafa Farouk Shaaban Assistant Professor, Department of Electrical Engineering Thesis Committee Member	_______________
_____________________________ Dr. Fadi Ahmed Aloul Head, Department of Computer Science and Engineering	_______________
_____________________________ Dr. Ghaleb Husseini Associate Dean for Graduate Affairs and Research College of Engineering	_______________
_____________________________ Dr. Richard Schoephoerster Dean, College of Engineering	_______________
_____________________________ Dr. Mohamed El-Tarhuni Vice Provost for Graduate Studies	_______________

Acknowledgement

Foremost, I would like to thank Allah for the opportunities and blessing I was given.

I would like to express my gratitude to my advisor Professor Raafat Aburukba for his guidance, support, and motivation. The enthusiasm he has for research, professionalism, and sharing knowledge has made this research a joyful and full of expertise experience.

I want to thank my co-advisor Professor Khaled El-Fakih for providing knowledge, support, and for being there whenever I needed help. I’m deeply beholden for his worthy discussion and suggestions.

I must express my very profound gratitude to my mother (Dr. Safia Mahgoub Karar) and my father (Dr. Bukhari Ibrahim), my sisters (Dr. Maisoon and Ms. Eman), and my brothers for providing me with unfailing support and continuous encouragement throughout the years. This accomplishment would not have been possible otherwise.

Finally, I would like to thank the American University of Sharjah and the Department of Computer Science and Engineering for granting me the Graduate Teaching Assistantship.

Dedication

To those who discouraged me and once said you cannot do it; I DID IT.

Abstract

Cloud computing infrastructures are designed to support the accessibility and availability of various services to consumers over the Internet. Datacenters hosting Cloud applications consume massive amounts of power, contributing to high carbon footprints to the environment. Hence, Green Cloud computing solutions are needed within the Cloud datacenters that optimize the energy consumption. The main objective of this thesis is to address the problem of power and carbon efficient resource management in a Cloud datacenter. This work focuses on the development of a dynamic task scheduling algorithm to enhance the datacenter power efficiency over time. To achieve this objective, a formal optimization model is proposed using Integer Linear Programming (ILP) that minimizes the energy consumption in a Cloud datacenter. The model is verified using exact techniques and the Genetics Algorithm (GA) heuristic-based technique. Furthermore, an adaptive GA is proposed to reflect the dynamic nature of the Cloud computing environment. The proposed adaptive GA is validated by simulating an IaaS Cloud environment and conducting a set of performance and quality evaluation study in this environment. The results demonstrate that the proposed solution offers performance gains with regards to response time and in reducing the power consumption in the Cloud datacenter.

Keywords: *Cloud computing; task scheduling; optimization; integer linear programming; power consumption; genetic algorithms.*

Table of Contents

Abstract 6

List of Figures 9

List of Tables 10

List of Algorithms 11

List of Abbreviations 12

Chapter 1. Introduction 13

1.1. Cloud Computing 13
1.2. Scheduling in Cloud Computing 16
1.3. Thesis Objective 18
1.4. Research Contribution 19
1.5. Thesis Organization 19

Chapter 2. Background and Literature Review 21

2.1. Scheduling Problem 21
2.2. Scheduling in Cloud Computing using Heuristics 23
2.3. Scheduling for Minimizing Power Consumption in Cloud Computing 25

Chapter 3. Methodology 33

3.1. Cloud Computing Environment 33
3.1.1. Cloud Provider. 33
3.1.2. Cloud Consumer. 34
3.1.3. Cloud Environment Setting. 35
3.2. Optimization Problem Formulation 36
3.3. Model Validation 40

Chapter 4. Genetic Algorithms for Solving the ILP Model 43

4.1. Genetic Algorithm 43
4.1.1. Encoding of Chromosomes 44
4.1.2. Initial Population. 44
4.1.3. Fitness Function. 45
4.1.4. Selection. 46
4.1.5. Crossover. 46
4.1.6. Feasibility Check and Penalization. 48
4.1.7. Elitism. 48
4.1.8. Termination Criteria. 49
4.1.9. GA Experimentation. 50
4.2. Comparison between Lingo™ and the Genetic Algorithm 51
4.3. Adaptive Genetic Algorithm 53
4.4. Adaptive Genetic Algorithm Workflow 57

Chapter 5. Experimentation and Results 61

5.1. Simulation Environment 61
5.2. Simulation Results 62
5.3. Performance Evaluation 65
5.4. Sensitivity Analysis 68

Chapter 6. Conclusion and Future Work 72
References 74
Appendix A: Experimentation Data 79
A.1. Tasks Requirements 79
A.2. Tasks Deadline 79
A.3. Resources Capabilities 80
A.3.1. Computational Resources 80
A.3.2. Storage Resources 80
A.3.2. Network Resources 80
A.4. Tasks Execution Time 81
A.5. Resources Power Consumption 81
Vita 82

List of Figures

Figure 1.1: Cloud Logical Architecture 17
Figure 3.1: Energy Aware Scheduling 36
Figure 3.2: Total Power Consumption, Lingo™ 42
Figure 3.3: Elapsed Time, Lingo™ 42
Figure 4.1: Crossover Example 47
Figure 4.2: Fittest Chromosome 49
Figure 4.3: Total Power Consumption using GA 51
Figure 4.4: Elapsed Time, GA 51
Figure 4.5: Total Power Consumption, Lingo™ vs. GA 52
Figure 4.6: Elapsed Time, Lingo™ vs. GA 52
Figure 4.7: Adaptive GA Workflow 57
Figure 4.8: GA, Population Size vs. Fitness (Power Consumption) 58
Figure 4.9: Adaptive GA, Population Size vs. Fitness (Power Consumption) 59
Figure 4.10: GA, Number of Generations vs. Fitness 60
Figure 4.11: Adaptive GA, Number of Generations vs. Fitness 60
Figure 5.1: Total Power Consumption, Adaptive GA 63
Figure 5.2: Total Elapsed Time, Adaptive GA 63
Figure 5.3: Problem Size vs. Elapsed Time per Tasks Batch 64
Figure 5.4: Adaptive GA vs. GA 66
Figure 5.5: Power Consumption, Adaptive GA vs. FCFS 67
Figure 5.6: Total Elapsed Time, Adaptive GA vs. FCFS 68
Figure 5.7: Power Consumption for Different Tasks 69
Figure 5.8: Total Scheduling Time for Different Tasks 69
Figure 5.9: Power Consumption for Different Resources 70
Figure 5.10: Total Scheduling Time for Different Resource 71
Figure 5.11: Power Consumption, All Resources vs. Only Compute Resources 71

List of Tables

Table 3.1: Proposed Model Notations 37
Table 3.2: No. of Tasks and Resources 41
Table 3.3: Lingo™ Objective Value and Elapsed Time... 41
Table 4.1: GA Fitness and Elapsed Time 50
Table 5.1: Physical Resources Capabilities 61
Table 5.2: Number of Tasks and Resources 62
Table 5.3: Total Number of Tasks and Resources, Different Tasks............................ 69
Table 5.4: Total Number of Tasks and Resources, Different Resources..................... 70
Table A.1: Tasks, Requested Capabilities 79
Table A.2: Tasks, Deadline 79
Table A.3: Computational Resources, Provided Capabilities 80
Table A.4: Storage Resources, Provided Capabilities 80
Table A.5: Network Resources, Provided Capabilities 80
Table A.6: Execution Time of Tasks in Capable Resources 81
Table A.7.a: Power Consumption of Resources 81
Table A.7.b: Power Consumption of Resources……………………………………..81

List of Algorithms

Algorithm 4.1: Genetic Algorithm 44
Algorithm 4.2: Initialization of Population 45
Algorithm 4.3: Fitness Function 45
Algorithm 4.4: Selection Based on Roulette Wheel Method 46
Algorithm 4.5: Crossover 47
Algorithm 4.6: Feasibility Check and Penalization, GA 48
Algorithm 4.7: Elitism 49
Algorithm 4.8: Adaptive GA 54
Algorithm 4.9: Feasibility Check and Penalization, Adaptive GA 56

List of Abbreviations

API	Application Programming Interface
BoT	Bag of Tasks
COP	Coefficient of Performance
CPU	Central Processing Unit
DC	Data Center
DVFS	Dynamic Voltage and Frequency Scaling
DVS	Dynamic Voltage Scaling
FCFS	First Come First Serve
GA	Genetic Algorithm
HPC	High Performance Computing
IaaS	Infrastructure as a Service
ILP	Integer Linear Programming
MI	Million Instructions
MIPS	Million Instructions Per Second
NIST	National Institute of Standards and Technology
QoS	Quality of Service
POP size	Population size
PaaS	Platform as a Service
SaaS	Software as a Service
SLA	Service Level Agreement
UPS	Uninterruptable Power Supply
VM	Virtual Machine
WAN	Wide Area Network

Chapter 1. Introduction

In recent years, Cloud computing has become one of the most talked about technologies and has captured the attention of service providers due to the opportunities it offers. It presents a different way to remotely use and manage various resources and services. This chapter illustrates the fundamental concepts of Cloud computing and the scheduling problem in the Cloud environment. Thereafter, a discussion on the problem investigated, the thesis contribution, and the thesis organization are introduced.

1.1. Cloud Computing

Cloud computing has been emerging as a successful paradigm for providing computing services to consumers. The availability of low cost computers, servers, storage devices, and high capacity networks motivated Cloud providers to expose underutilized resources as a utility to consumers over the Internet in a pay as you go manner. Those Cloud providers ensure the ultimate use of the provided services while guaranteeing high quality of service and customer satisfaction. There are various definitions to cloud computing. Buyya [1] defined Cloud computing as “A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers”. This definition specifies the following characteristics of Cloud computing [1]:

- The increase of resource utilization through virtualization by creating multiple tenants from the physical resources.
- Dynamic creation and provisioning of services based on consumers’ needs and without human intervention from the service provider side.
- Isolation of the infrastructure from the users by presenting the underlying physical and virtualized resources as unlimited resources to consumers.
- Quality of service through the service-level agreements between the service provider and the consumer in order to ensure consumer’s satisfaction.

Another definition was proposed by The National Institute of Standards and Technology (NIST) that defined Cloud computing as “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" [2]. NIST states a set of characteristics that can be achieved by the proper planning and configuration of the physical resources and can be summarized as follows [2]:

- On demand self-service: users can provision resources without human intervention from the service provider side.
- Broad Network Access: authorized access and use of the resources hosted by the Cloud from heterogeneous devices.
- Resource Pooling: the process of increasing resource utilization through aggregating the physical resources in groups (pools) to create multiple tenants that serve multiple consumers at the same time. To attain resource pooling the underlying hardware should support virtualization which provides the abstraction of the physical layer by making pools from the physical resources.
- Rapid Elasticity: allows allocation and release of resources when needed based on the consumers' requirements. Elasticity is possible because of resource pooling and on demand self-service.
- Measured Services: services offered by the Cloud are monitored and measured by the service providers for various purposes such as billing, effective use of resources, and capacity planning. Different service providers have different pricing models such as pay as you go, prepaid subscriptions, through resellers or enterprises agreements.

Both definitions consider the fact that Cloud computing environment should support virtualization of resources. These resources appear as unlimited resources to consumers using resource pooling and multi-tenancy. Also, the two definitions emphasized that users can lease and release resources based on their requirements without service providers' involvement.

The services provided by the Cloud can be categorized to [2] Infrastructure as a Service (IaaS): in which consumers can rent and have access to the Cloud physical resources (compute, storage and network resources), Platform as a Service (PaaS) where users are provided with a framework to build their own applications and manage these applications, whereas in Software as a Service (SaaS) consumers have access to applications hosted in the Cloud through web browsers or software interfaces. All the above mentioned characteristics and services can be deployed either on a public Cloud, private Cloud, community Cloud, or a hybrid Cloud [2]. On a public Cloud resources are made available to the public users over the Internet while the resources on a private Cloud are made available to a specific user or organization exclusively. On the other hand, serving a group of consumers that share the same interest is done by deploying a community Cloud. Any of the previously mentioned deployment models can be mixed to make a hybrid Cloud.

The Cloud infrastructure is made of server farms or datacenters to support accessibility and high availability of services to consumers. This work focuses on providing infrastructure services in a way that will contribute in minimizing power consumption in a datacenter. The Cloud services are usually deployed in datacenters that contain thousands of physical resources. However, to meet the growing demand for services, Cloud computing provides high performance resources that result in a notable increase in energy consumption. Hence, solutions are required to optimize energy in Cloud datacenter.

Cloud datacenter resources (computation, storage, and network) are heterogeneous. Computation resources are the collection of physical machines (servers and hosts) that are placed in data centers to provide processing capabilities and to efficiently manage the Cloud infrastructure, execute software to deliver software and platform services, or leased to consumers. Storage systems on the other hand are used to store and retrieve Cloud data, applications data, and consumer data. Network resources are used to connect the compute resources to each other, connect compute resources to the storage resources and connect Clouds (via WAN).

The major resources that consume massive power in datacenters are compute servers, storage servers, interconnecting networks (switches/routers), cooling systems, and electrical devices (UPSs) [3, 4]. Along with these resources, service providers should keep in mind a set of factors that can result in an increase in power usage such as [4]: the design and implementation of software applications that might result in higher power consumption than required due to the high CPU and memory usage, the tradeoffs between QoS and power consumption (SLAs between providers and consumers that states extra resources) and the proper use of network bandwidth.

Cloud computing make use of technologies that are not available in traditional computing to achieve better utilization and performance. One of these technologies is virtualization which enables the consolidation of underutilized servers to a set of virtual machines running on the same physical machine. Other techniques such as dynamic provisioning of resources during high demand periods and multi-tenancy management. Furthermore, VM consolidation and VM migration through the use of virtualization, demand projection, heat management and temperature-aware allocation, load balancing and task scheduling [4] are used as basic techniques for minimizing energy consumption in Cloud environments. This work focuses on task scheduling that optimizes the energy consumption in the datacenter.

1.2. Scheduling in Cloud Computing

Generally, schedulers allocate tasks to resources based on some constraints and objectives. The Cloud infrastructure logical layered architecture is depicted in Figure 1.1, where each layer depends on the services provided by the lower layer. The Cloud infrastructure layers are as follows:

- **The Physical layer** consists of Cloud datacenter resources such as compute systems, storage systems, and network resources.
- **Virtualization layer** aims on abstracting the physical layer by making multiple virtual instances and pools the physical resources. This is achieved by deploying the virtualization software on the physical resources.
- **The Control layer** is responsible for configuration, provisioning, and monitoring of resources. It is also responsible for task scheduling by taking the requests from the service and the orchestration layers and executing tasks based on the workflow defined at the orchestration layer.
- **The Service Orchestration layer** is responsible for orchestrating the execution of the requests coming from the Cloud consumers.
- **The Service Layer** provides functional and management interfaces in order to enable the creation of services by providers and the utilization of these offered services by consumers.

Scheduling in the Cloud infrastructure is done by the control layer. Scheduling is the allocation of resources (physical or virtual) to execute tasks at specific time to achieve quality of solution such as minimizing the energy consumption. Scheduling problems can be modeled as optimization problems that require finding the best solution among all the feasible solutions. This problem is classified as an NP-hard problem [5].

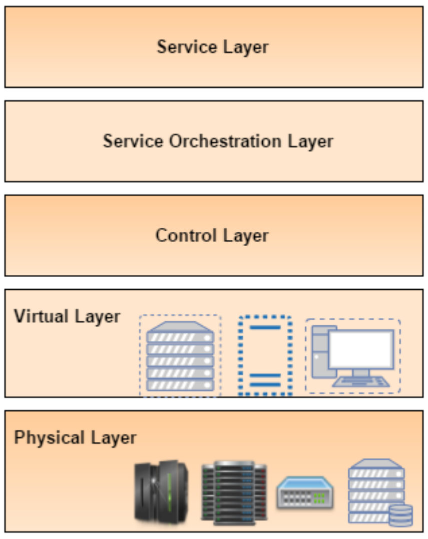


Figure 1.1: Cloud Logical Architecture

The scheduling problem can be classified to three categories based on the nature of the environment:

- Centralized scheduling [6], a single scheduler is responsible for decision making and for mapping tasks to the available resources. In this scheduling scheme, the knowledge about the resources and tasks and the control are centralized.
- Distributed scheduling [6], is more complex than the centralized scheduling where the knowledge is distributed and the control is centralized.
- Decentralized scheduling [7], is more complex than distributed scheduling, where the knowledge and control are distributed and each entity is responsible for making the scheduling decisions based on its policies and objectives.

Furthermore, scheduling problems can be categorized as either static scheduling [8] or dynamic scheduling [8] depending on the nature of the scheduling problem and can be solved by finding the optimal (exact) or near optimal solution through heuristic algorithms.

- In static scheduling tasks and resources are known in advance and a schedule is defined before the execution of the algorithm with less runtime overhead.
- In dynamic scheduling, tasks are not known in advance and resources can join and disjoin at any time. Dynamic strategies can be further classified to an online mode [9] in which the scheduler considers only the newly arrived task for scheduling (e.g. Most-Fit Algorithm) and a batch mode [9] where the newly arrived task will cause the scheduler to reschedule the new and the previously waiting tasks (e.g. Min-Min Algorithm[6]).

In the Cloud environment, task scheduling process is done in the control layer and can be generalized in three stages [10]: *(a) the resources discovery and filtering* in which the scheduler collects information about the available resources, *(b) the resource selection* for deciding the appropriate resources based on performance, availability and capacity, and *(c) the task submission* in which the incoming tasks are submitted to the selected resources for execution. The incoming tasks can be characterized by the requirements (computation, storage, and network) requested or by their priority, size, deadline, arrival time, waiting time, start time, or pre-emption.

1.3. Thesis Objective

Nowadays, most organizations depend on the Cloud infrastructure for the operation and development of their business without the need to worry about purchasing, configuring, and monitoring their own infrastructure. However, from service provider’s perspective, the growing demand of Cloud infrastructure services result in an increase in energy consumption and generates high carbon emissions. A recent article mentioned that the world’s data centers used about 416.2 terawatt hours of electricity in the last year [11]. Hence, there is a need for moving towards decreasing this level of usage while preserving the quality of service and meet the Service Level Agreement (SLA). Consolidation of VMs and VM migration, scheduling, demand projection, heat management and temperature-aware allocation and load balancing are some of the techniques used for minimizing the energy consumption in the Cloud infrastructure [4].

Most of the research done for minimizing energy consumption in Cloud computing focused on one type of resources which is compute resources. In addition to compute resources, this work also considers storage, network resources, and cooling systems within a datacenter. Furthermore, with the world moving toward green environment, this work proposes a solution for optimizing power consumption in Cloud datacenters that will result in less carbon emission and hence saves energy.

This work aims at developing a model and an algorithm for optimizing energy consumption in an IaaS Cloud by scheduling the incoming tasks to resources in such a way that consumer demands are met and the Cloud datacenters energy is optimized. The focus will be on a Cloud that consists of one datacenter and owned by a single service provider. This datacenter consists of resources such as compute resources, storage, and network resources with constraints like processing power, memory capacity, and bandwidth. The scheduling process will take into consideration the requirements of the incoming tasks (e.g. resources requirements and deadline).

1.4. Research Contribution

This work contribution is summarized as follow:

1- Formulation of an integer linear programming model for optimizing energy consumption over time for Cloud computing IaaS.
2- Validation of the ILP model using optimization-modeling software (Lingo™) that is based on optimization exact solution method and a heuristic Algorithm (Genetic Algorithm) that gives near optimal solution.
3- Proposing an Adaptive Genetic Algorithm for dynamic task scheduling in the Cloud environment.
4- Simulation of Cloud IaaS and testing the proposed algorithm in the simulated environment.
5- Quality of Solution (QoS) result experimentation of the adaptive GA by comparing current approaches versus our proposed approach.

1.5. Thesis Organization

The thesis is organized as follows: Chapter 2 presents the literature review on task scheduling and optimization of energy consumption. Chapter 3 presents the formulation of the linear programming model and the validation of this model. Chapter 4 illustrates the proposed Adaptive Genetic Algorithm, its phases and workflow. Chapter 5 presents the experimentation and results.

Chapter 2. Background and Literature Review

Task scheduling is a key focus of Cloud computing and the way it is implemented has a direct impact on the Cloud performance and the quality of service. In general, scheduling is performed to optimize different objectives such as total completion time, total cost, total energy consumed, or any other objective depending on different users' constraints. This Chapter presents a review of the scheduling problem, the algorithms and techniques used to achieve different objectives, and illustrates what researchers did to optimize energy consumption.

2.1. Scheduling Problem

Most of the scheduling problems are subject to optimization objectives and feasibility constraints and can be solved by finding a solution within the feasible region using either exact or heuristic methods [12]. Exact algorithms are guaranteed to give the optimal/exact solution to the problem but with a dramatic increase in run-time. Some examples of exact algorithms include: Simplex Algorithm and Branch and Bound Algorithm.

- Simplex Algorithm:

It solves optimization problems that are in standard form. The Simplex method is an iterative method that starts from a vertex in the feasible region and tests the adjacent vertices of the feasible region until it reaches a vertex that gives the optimum solution [13].

- Branch and Bound Algorithm:

This algorithm searches the solution space (feasible region) to form a tree of possible solutions. It starts by initialization for finding a good initial solution that is counted as an upper bound. Then it branches the solution space into smaller sub problems (partial solutions) and it keeps on bounding these partial solutions to decide which one will be branched next and which will be discarded [14].

In this work, an optimization tool (Lingo™) will be used to find an exact solution to the proposed linear programming model. This tool uses branch and bound algorithm to find the exact solution.

On the contrary, heuristic algorithms do not guarantee finding the optimal solutions to decrease the run-time (converge fast) and only attempt to yield a good solution (near optimal). These algorithms include:

- First Come First Serve Algorithm (FCFS) [15]: in which tasks that come first in the queue will be scheduled and processed.
- Round Robin Algorithm [15]: where CPU time is divided equally between the tasks in a round robin fashion and will be paused if the time slice is expired and the task will be placed at the back of the ready queue.
- Min - Min Algorithm [15]: assigns small tasks to the resources that give the minimum completion time. Longer tasks have to wait till all the small tasks finish which may cause starvation and increase the total completion time.
- Max - Min Algorithm [15]: is similar to the Min – Min algorithm except that it executes large tasks first on resources with minimum completion time and the smaller tasks have to starve.
- Priority Scheduling Algorithm [8]: Each task is assigned a priority and scheduled accordingly. Equal priority tasks are scheduled in First Come First Serve order.
- Ant Colony Optimization [16] is a population based probabilistic algorithm that searches for optimal solution based on the behavior of ants seeking a path between their colony and a source of food. Ants lay down pheromone trails and shorter paths will have higher pheromone levels than longer ones.
- Honey Bee Foraging Algorithm [17] is another population based algorithm that mimics the food foraging behavior of honey bee colonies and it performs neighborhood search to find better solutions.
- Particle Swarm Optimization algorithm is inspired by social behavior of bird flocking and has been used in many areas: function optimization, artificial neural network training, fuzzy system control. It starts by having a population of candidate solutions and iteratively tries to improve these candidate solutions [18].
- Genetic Algorithm [19] begins its search with a random set of solutions. All solutions are assigned a fitness which is directly related to the objective function and it modifies the solution by applying the three operators similar to natural

genetic operators: reproduction, crossover, and mutation. This algorithm will be used to find a near optimal solution to the proposed ILP model.

2.2. Scheduling in Cloud Computing using Heuristics

Many researchers proposed algorithms for task scheduling in cloud computing with different parameters and characteristics to achieve a specific goal like minimization of makespan, minimization of response time, maximization of datacenter throughput, or increasing resource utilization.

In order to minimize the makespan, a task scheduling strategy was proposed in [20] and [21] with the assumption that multiple independent tasks form a single job. In the first strategy, the datacenter is composed of clusters of heterogeneous computation servers. The task scheduler is responsible for the proper assignment of resources to tasks based on the minimum completion time. The second proposed algorithm called A Group Leaders' Optimization Algorithm in which the problem space is divided into groups and each group has members (10 members) that consist of tasks and resource for each task which are chosen randomly in the initial population production. A fitness value is calculated for each member of each group so that the member with the best fitness value is selected as the group leader. For each old member, a new member is created based on the old member, the leader and a random value and is placed in the old member location if its fitness value is better than the old one. The previous steps are repeated according to a determined number of steps and the leader with the best fitness value is chosen as the problem solution. However, it is very hard to guarantee that tasks coming from cloud consumers are independent from each other.

In [22], an Artificial Bee Colony algorithm was used to minimize the makespan and waiting time. Each task has a number of operations and a known sequence among them and can be processed on one machine. The cloud nodes process one operation at a time. A random solution is generated for each bee in the initialization phase and then the bees search for a better solution around the randomly selected one till they find the best solution based on the fitness value. A work proposed in [23] took customer satisfaction into consideration and it used Ant Colony Optimization to minimize makespan, resources consumption and achieve high user satisfaction. The scheduling problem is represented by a graph that contains the VMs, the independent tasks and the edges that connect between them and ants are placed at the starting VMs randomly. Ants build solutions by moving from the first VM to another until all tasks are allocated and till they reach the maximum number of iterations. In every iteration, the length of the tour is calculated, the current optimal solution is updated with the best-founded solution and tasks are assigned to VMs for execution.

The work in [10] proposed a generalized priority based algorithm in which tasks and VM were prioritized based on their size (MI) and processing speed (MIPS) respectively. Tasks with higher sizes and VM with higher processing speed have higher priority. Their environment has multiple datacenters and a broker is responsible for scheduling tasks with higher priorities to VMs with higher MIPS. The results showed that the priority based algorithm has better response time than FCFS and Round Robin.

Resource utilization is another issue related to scheduling in cloud computing. In [24] authors used a dynamic scheduling algorithm in a client server environment - consists of one virtual cloud user and one virtual cloud server that contains the controller server, storage controller and the storage servers. Tasks are requests for uploading and downloading files of any type. The algorithm calculates the load of each node and achieves load balancing because controller server will forward tasks to servers with the least number of tasks while full servers will be excluded. The performance of this system was compared to the FCFS algorithm and the authors found that their proposed algorithm takes less time to complete a task and it distributes load evenly among nodes.

Multi objective task scheduling algorithms were investigated in [25, 26] to improve datacenter throughput and reduce cost without violating SLA. In both work the cloud is made of multiple service providers and each service provider manages multiple datacenters so a cloud broker is responsible for scheduling the received tasks to the allocated VMs in order to minimize the execution time. Each VM has an ID and MIPS value while each task has an ID, a QoS value and a size (Million Instructions). The broker requests the QoS of received tasks from the service providers to assign high priority for low QoS tasks. It also takes the list of VMs created in the datacenters to use MIPS value for assigning VM’s QoS. High priority tasks will be assigned to high QoS VMs and this result in better execution time and throughput in comparison to the FCFS and the priority algorithm.

A Bandwidth-aware task scheduling (BATS) algorithm was studied in [27] that not only consider task resource requirements for CPU and memory but also the network bandwidth requirements to minimize the total time to finish tasks. The Broker is responsible for accepting the tasks from the users and the list of the VMs with their computing power and bandwidth from the datacenters. It uses this information to build a nonlinear programming model and pass this model to be solved by Lingo™ solver. The result is the optimized task allocation scheme (proper number of tasks assigned to each VM). Then the broker distributes tasks to the VMs based on the optimized scheme that minimize the total time to complete all tasks.

GAs have been extensively used for finding the optimal solution to the task scheduling problem in cloud computing. A study [28] used a Modified Genetic Algorithm with the objective of makespan minimization. The initial population (possible solutions) was generated by using the Enhanced Max-Min algorithm. The evaluation of these solutions (chromosomes) was done by applying a fitness function in order to pinpoint the solution with the least makespan. Furthermore, single point crossover and mutation were done to the selected solutions to generate fitter solutions. Again, this work was made based on the assumption that tasks are independent. Authors in [29] proposed an improved version of GA and they compared its performance to the standard GA. The only difference is that they used Min-Min and Max-Min algorithms in the generation of the initial population to generate chromosomes that are fitter. They proved the improved genetic algorithm results in reduction in the execution time and proper utilization of resources. Another work that aims to minimizing the completion time and maximizing resource utilization using Tournament Selection Genetic Algorithm was presented in [30]. Tournament selection is used to choose the parents for the generation of new children. If there is a good solution but it is not selected in the crossover process it will not be removed from the list but it will be reserved to the next selection iteration and this what makes the proposed algorithm different from the previous ones.

2.3. Scheduling for Minimizing Power Consumption in Cloud Computing

For power consumption in cloud environment, researchers proposed different ways to reduce the amount of power consumed in datacenters and minimize carbon emission levels without compromising the quality of service. Scheduling is one of the techniques that used in cloud computing in order to minimize the power consumption. In [4] authors mentioned that the Power Usage Effectiveness (PUE) and the Datacenter Infrastructure Efficiency (DciE) are the two metrics used for measuring datacenter power efficiency and they proposed a framework to make green cloud by keeping track of energy usage of fulfilling users requests while meeting providers goals. In this framework, all users submit their requests to the Green Cloud Broker that selects based on users QoS the greenest provider (with less power consumption and carbon emission) to serve the request. The service providers who offer computational resources to execute HPC applications place their green offers (green services, prices and time) in a public directory accessed by the Green Broker which in turn will take all the data related to energy efficiency from the Carbon Emission Directory. The Green Broker selects the set of services with the least carbon emission after calculating the carbon emission of all the cloud providers offering the requested services.

To prove the Green framework efficiency, the authors studied five policies – Green and Profit oriented- used by the Green Broker for scheduling:

- Greedy Minimum Carbon Emission (GMCE): User applications are assigned to providers based on their carbon emission.
- Minimum Carbon Emission - Minimum Carbon Emission (MCE-MCE): applications are assigned to providers with minimum carbon emission due to datacenter location and application execution.
- Greedy Maximum Profit (GMP): user applications are assigned to providers that will execute them fast and with maximum profit.
- Maximum Profit - Maximum Profit (MP-MP): this considers the profit and applications deadline.
- Minimizing Carbon Emission and Maximizing Profit (MCE-MP): user applications are assigned to providers with minimum carbon emission and maximum profit.

And they found that the green policies reduce the carbon emission and energy consumption by 20% and have minimal effect on provider’s profit.

A task scheduling algorithm was proposed in [3] as an integer programming problem for minimizing data center power consumption with respect to task response time and the number of servers required for processing a task in a specific time frame. The work focused on minimizing the number of active servers with an assumption that a complete task is assigned to a single server and the network provides enough bandwidth to avoid delays. It is stated that the energy consumed by datacenter server is a sum of the total number of tasks assigned to a server, the processing time for each task and the power consumed by the server to process each task. The objective is to find an optimum assignment of tasks in order to minimize servers processing time which in turn will minimize the total energy. The proposed algorithm is called the most-efficient-server- first scheme and it contains a central task scheduler that maintains a sorted list with the most energy efficient servers on top. For a datacenter with single server type, the scheduler starts by assigning the incoming tasks to the most energy efficient servers then the servers update their energy consumption profile at the central scheduler. If these energy efficient servers are full, the newly arrived tasks will be scheduled to less energy efficient servers. This scheme did not consider the tasks deadline and the writers did experiments with only one datacenter that has servers of one type. They concluded that this scheme consumes more than 70 times less on server energy than a random-based task scheduling scheme. The random-based scheduling scheme schedules tasks to servers randomly without considering any additional constraints except available queues in each server.

The Dynamic Voltage and Frequency Scaling (DVFS) technique can be used to reduce the power consumption of IT equipment. In [31], researchers provide a scheduling algorithm using this technique in order to reduce power consumption, increase resource utilization and meet SLA requirements. In their work, servers and VMs are denoted by the minimum and maximum working frequency, weight and priority and also a specific SLA level that the consumer choose (the higher the level, more VMs are created). The system architecture consists of the *job submission* in the form of minimum and maximum amount of resources in frequency and the SLA, *the VM manager*: that sends the received job requirements and the state of the resources to the scheduling algorithm and creates VMs, deploy the job on them, and shutdown the servers and VMs that are idle, *the scheduling algorithm*: responsible for calculating the weights of the available resources and sort them in increasing order, selects the resource that satisfy the job requirements, its SLA, and consumes the least energy, *DVFS controller*: sets applicable supplies of voltages and frequencies to servers according to their resource requirements. Another energy efficient scheduling algorithm of deadline-constrained applications using the same technique was proposed in [32]. Their data center consists of homogenous physical hosts with different number of cores, VMs with dedicated cores, and a Resource Management System (RMS) installed in the PaaS layer. The RMS coordinates the execution of the urgent CPU-intensive BoT applications and manages the frequency and voltage of each core. The power consumed by a host is a function of the frequency level of each of its cores and the hosts that are not in use will be suspended until one of their VMs is required. The proposed algorithm scales the frequency and voltage of CPUs assigned to VMs to execute the tasks before the deadline. Submitted jobs consist of information about the tasks that compose the job, the estimated runtime of each task, and a deadline for the job to complete. The algorithm works as follow: for every arrived task, it iterates over the VMs till it finds the suitable one, insert the task in the queue of that VM and then sort the queue in ascending order of task deadlines. The next step is the calculation of the lowest frequency that can be used to meet the deadline of all tasks, if the highest frequency of a VM cannot meet the deadline then this VM is not suitable for the selected task. Authors of [33] proposed an algorithm that uses DVS in order to consolidate the load, hence maximize resource utilization, and minimize power consumption. In their environment, the tasks are fetched by the datacenter broker and grouped based on their size to high computing tasks, medium computing tasks and low computing tasks. The resources are grouped to resources that run in maximum frequency, medium frequency and low frequency and assigned to the incoming tasks accordingly. The algorithm starts by calculating the instruction length for each task and high computing tasks are scheduled to resources with maximum frequency, medium computing tasks to resources with medium frequency and so on. The scheduling within the resource is based on First Come First Serve and if one of the tasks in medium or low frequency resources fails it will be rescheduled to the high frequency resource. A multi-objective task scheduling algorithm that uses the DVFS called Green Task Scheduling (GTS) was presented in [34] to minimize the makspan, cost and energy consumption by minimizing the number of active servers. In the proposed environment, they have a *cloud controller* who takes service requests from consumers and sends the information about these requests to the GTS algorithm. The algorithm finds the proper schedule and sends it to the *green service allocator* that allocates the tasks to the VMs. The DVFS technique is used to turn on/off unused machine and set the proper frequencies to the running machines. Through experimentation, the Green Task Scheduling Algorithm reduced the makspan and the energy consumption in comparison with FCFS, Shortest Job First and Round Robin.

Load Balancing is another technique used for minimizing power consumption in Cloud computing. In [35], authors proposed a Preemptive Priority Based Job Scheduling Algorithm for load management, minimizing energy consumption and maximizing revenue. This algorithm schedules tasks to compute resources based on energy requirements and uses the DVFS to set resources frequency based on the load. The jobs are submitted to the VM manager in terms of the amount of frequency required; the VM manager calculates the weights of the available servers in terms of power cost and submits the jobs and the weights to the scheduling algorithm. The algorithm selects the server that consumes the least energy and assigns the VM in this server to perform the job. The proposed algorithm is preemptive in such a way that if a received job has higher priority than an executing job, the latter will be suspended and added to the ready queue. Instead of assigning priority to the received jobs, authors in [36] proposed a Shortest Job First scheduling algorithm to minimize the number of active servers and hence power consumption. This algorithm uses the Dynamic Power Management (DPM) scheme to assign the shortest jobs to the servers that execute them with minimum processing rate. In [37], researchers proposed an algorithm that focuses on the natural behavior of fireflies called Firefly algorithm to balance the load in Cloud datacenters. The algorithm starts by the generation of initial population of fireflies and evaluating the fitness of the generated fireflies by the objective function. The brightness function (i.e. objective function) is a load function with less bright firefly is counted as under loaded node. The aim is to take all nodes to a threshold level of load.

Authors in [38] proposed an algorithm that assigns more tasks to the virtual machines that will execute them with lower energy consumption under the premise that the makespan is within a threshold value. The algorithm sorts the tasks based on their size and sorts the VM based on the processing speed. Then, for each task in the list it selects the VM that can execute the assigned task with minimum speed. If the selected VM does not meet the deadline of the task, this VM is marked as unusable and the algorithm search for another VM. Through the experimentation, authors proved that the speed of VMs has a major effect in energy consumption.

Moreover, lots of research has been done in the area of green computing and the optimization of power consumption in the cloud environment using genetic algorithm. In [39], authors proposed a genetic algorithm with the goal of scheduling tasks based on the available resources and the energy consumption. In this work, the chromosome is made of genes equal to the number of tasks and the value of each gene is a positive integer between 1 and the number of virtual machines (each field represents the VM number assigned to the task). The fitness function sorts the VMs based on their energy consumption and the ones with minimum energy are used for the crossover and mutation. The results show that the proposed method is more scalable and has lower energy consumption in comparison to First-Fit Decreasing and Best-Fit Decreasing algorithms. Minimizing the makespan as well as the power consumption was the objective of the authors in [40] by using the Dynamic Voltage Scaling (DVS)-dynamic adjustment of processors voltage - to minimize energy consumption. They proposed two algorithms to define the fitness and select individuals in the genetic algorithm (Energy Consumption Time Unify Genetic Algorithm (ETU_GA) and Energy consumption Time Double Fitness Genetic Algorithm (ETDF_GA)). In the first algorithm, the fitness function is made of the sum of the makespan and the total power consumed. Each individual is given a probability based on its fitness over the total fitness and the selection is based on this probability. The second algorithm on the other hand defines two fitness functions, one for the makespan and the other one for the power consumed. Then for each individual it computes a probability for every fitness value and selection is based on Roulette wheel method.

Researchers in [41] proposed a task scheduling algorithm that uses GA in order to fully utilize all compute nodes while reducing the total power consumed by the Cloud. Tasks are submitted to the Cloud and the service scheduler is responsible for allocating tasks to resources. The chromosomes are made of genes that are equal to the number of tasks and their values represent the allocated resource ID to the task. Initial population is created randomly; the fitness function calculates the makespan of giving task execution pattern and the Roulette wheel method is used for selecting the individuals for the reproduction. A single point crossover and random mutation were used with the constraint that each task should be assigned to one resource only and it shouldn't exceed the maximum utilization capacity. A different approach was presented in [42] that use genetic algorithm in order to balance the load across cloud resources and hence avoid overheating which will result in reducing the amount of power consumed. In this approach, each individual is made of the node numbers and the load assigned to each node. The fitness depends on the energy consumption, the node load and the average load. Each individual will have a fitness value equals to the sum of the difference between the node load and the average load. A two-point crossover is used and the parents and the newly created children are of different sizes but with the same total load. Parents and children are made of different sizes to allow for shutting down the nodes with no load and therefore reducing power consumed. Also for each individual in the mutation phase, two randomly selected points are chosen to sum their load and distribute the sum equally between them. This approach did not concentrate on tasks related constraints.

Another energy aware task scheduling using genetic algorithm was presented in [43] for minimizing the makespan and the energy consumption in cloud datacenters. The energy consumption for an application execution on a VM is given by the power of the VM multiplied by the execution times of the application with the constraints that the application should be executed before the deadline and consideration of the minimum energy to be consumed when a task is assigned to a VM. The cloud service provider receives the applications requests that consist of number of jobs and contain information about the execution time of the application, the number of VMs needed and the deadline. Then the energy value for each application is calculated and if the value is greater than a threshold, the application will be forwarded to the cloud task manager. These requests will be forwarded to the energy-aware scheduler who will decide which tasks will be scheduled to which machine based on the genetic algorithm.

Most of the previous work in the subject, as presented in the literature focused on a specific type of resources (computational) and the power consumed by these resources without considering the effect of other resources in power consumption. This thesis considers the scheduling of tasks to the different types of resources based on resources' capabilities and the tasks' constraints such as execution time, start time and deadline. In this Thesis, the scheduling problem is formulated as an integer linear programming (ILP) model with the objective of minimizing power consumption.

Chapter 3. Methodology

This work aims on resolving a specific class of the scheduling problem within Cloud computing. This Chapter describes the scheduling problem within the Cloud environment and presents an optimization model. The proposed model is validated using Lingo™ Solver that finds the exact optimal solution [48].

3.1. Cloud Computing Environment

The Cloud environment consists of two types of entities namely, Cloud consumers and providers. The Cloud providers are the entities that provide resources and/or services to be consumed. Cloud consumers are the entities that consume resources and/or services provided by the Cloud provider. This section provides an overview of the nature of the Cloud consumer environment in the tasks' form and the providers' perspective in the form of resources.

The Cloud physical computational resources are categorized based on: processors architecture: such as Intel and AMD; processor speed; number of cores, and memory capacity. Moreover, storage resources can be characterized based on the storage type, storage capacity, and speed.

On the other hand, tasks can be characterized either by: their size in Millions of Instructions; priority; time element (arrival time, start time, waiting time, and deadline); dependency between tasks; and preemption. The Cloud environment can be composed of single or multiple datacenters that are managed by one or multiple service providers. The focus of this work will be on a Cloud that consists of one datacenter and owned by a single service provider.

3.1.1. Cloud Provider. The Cloud environment is made of N data centers $DC = \{DC_1, DC_2, \dots, DC_N\}$ where each datacenter contains number of resources R. These resources are:

- M Computational Resources $R^c = \{R_1^c, R_2^c, \dots, R_M^c\}$ where $R_i^c = \{e_i, s_i, m_i, ar_i\}$, and

  $i = 1, \dots, M$.

  e_i: Number of cores of computational resource i.

s_i: CPU speed of computational resource i.

m_i: Memory of computational resource i.

ar_i: Processor architecture of computational resource i.

- O *number* Storage Resources R^s= $\{R_1^s$, R_2^s,…, $R_O^s\}$where $R_j^s = \{cap_j, st_j, sp_j\}$, and

  $j = 1, \dots, O$.

  cap_j: Capacity of storage resource j.

  st_j: Type of storage resource j.

  sp_j: Speed of storage resource j.

- P Network Resources R^n= $\{R_1^n$, R_2^n,…, $R_P^n\}$where $R_k^n = \{b_k\}$, and

  $k = 1, \dots, P$.

  b_k: Bandwidth of network resource k.

3.1.2. Cloud Consumer. The tasks from consumers can be presented as a set of Q tasks $T_q = \{T_1, T_2, \dots, T_Q\}$ where each task can have different requirements presented as $T_q = \{R_{q'}^c, R_{q''}^s, R_{q'''}^n, T_{q,d}\}$. Each task can request a specific capability within a resource such as: $R_{q'}^c = \{R_{e,q'}^c, R_{s,q'}^c, R_{m,q'}^c, R_{ar,q'}^c\}$, $R_{q''}^s = \{R_{cap,q''}^s, R_{st,q''}^s, R_{sp,q''}^s\}$ and $R_{q'''}^n = \{R_{b,q'''}^n\}$. Also, tasks can have a start and end time requirements.

Where:

$q' = 1, \dots, Q'$, is a task requesting computational capability.

$q'' = 1, \dots, Q''$, is a task requesting storage capability.

$q''' = 1, \dots, Q'''$, is a task requesting network capability.

$R_{q'}^c$: Required computational resource by task q'.

$R_{e,q'}^c$: Required number of cores *e* by task q'.

$R_{s,q'}^c$: Required CPU speed *s* by task q'.

$R_{m,q'}^c$: Required memory capacity *m* by task q'.

$R_{ar,q'}^c$: Required processor architecture *ar* by task q'.

$R_{q''}^s$: Required storage resource by task q''.

$R_{cap,q''}^s$: Required storage capacity *cap* by task q''.

$R_{st,q''}^s$: Required storage type *st* by task q''.

$R^{s}_{sp,q''}$: Required storage speed *sp* by task q''.

$R^{n}_{q'''}$: Required network resource by task q'''.

$R^{n}_{b,q'''}$: Required network bandwidth *b* by task q'''.

$T_{q,st}$: Start time of task *q*.

$T_{q,et}$: Execution time of task *q*.

$T_{q,d}$: Deadline of task *q*.

3.1.3. Cloud Environment Setting. The task scheduling problem in Cloud computing is based on the following assumptions:

- Virtualization is used to create multiple instances from the physical resources.
- Each resource (computational or storage) can execute multiple tasks.
- A task is executed by only one resource.
- No preemption of tasks (i.e. a task cannot be interrupted).
- Power consumption of resources depends on the tasks assigned to them.

As shown in Figure 3.1, consumers submit requests to the Cloud. These requests are represented as tasks (i.e. a request can be composed of multiple tasks) and submitted to the Cloud orchestration layer that has the work flow of the tasks. The Resource Manager has the knowledge of the physical and virtual resources and it provides it to the scheduler. The energy-aware scheduler searches for the resources capable of executing the received tasks in terms of requested capabilities to the available capabilities per resource then it schedules each task to a resource in a way that minimize energy consumption. The scheduler must keep the tasks deadline into consideration in all cases.

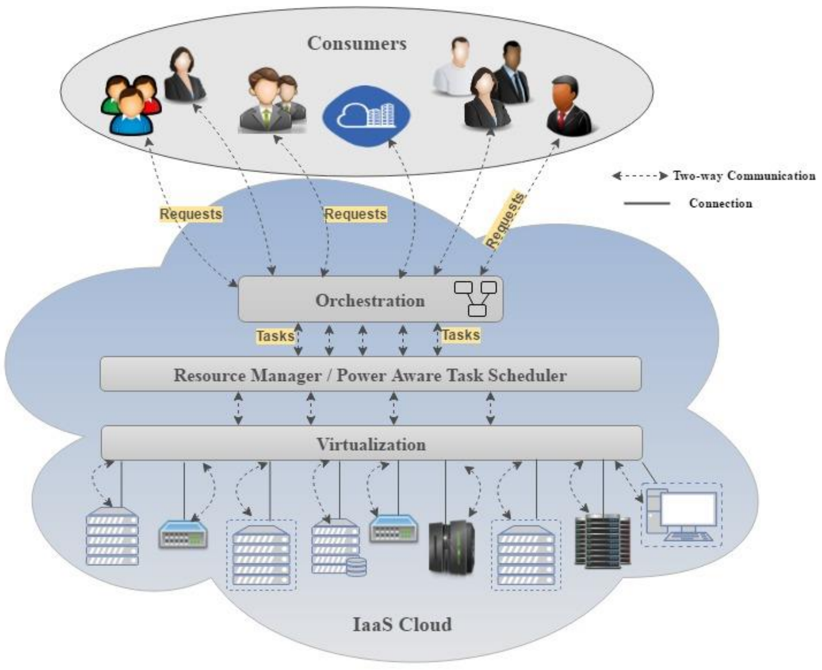


Figure 3.1: Energy Aware Scheduling

Scheduling can be mapped into an optimization problem where the best decision must be made to minimize or maximize an objective value. All optimization problems have an objective function that illustrates the value to be minimized or maximized and a set of constraints, which bound the possible solutions to the problem (Feasible region). In this work, we formulate the problem to an integer linear programming model to optimize the energy consumption.

3.2. Optimization Problem Formulation

The proposed model focuses on the minimization of energy consumption in a datacenter by scheduling the incoming tasks to the resources that will result in minimum energy while meeting the deadline for each task. Table 3.1 illustrates the notations used in the formulation of the proposed model.

Table 3.1: Proposed Model Notations

Notation	**Description**
$C_{R_i^c,R_{q'}^c}$	Power consumed by computational resource i when executing task q'.
$S_{R_j^s,R_{q''}^s}$	Power consumed by storage resource j when executing task q''.
$N_{R_k^n,R_{q'''}^n}$	Power consumed by network resource k when executing task q'''.
CS	Power consumed by Cooling System.
$x_{R_i^c,R_{q'}^c}$	Decision variable. Is 1 if task q' is executed on computational resource i and is 0 otherwise.
$x_{R_j^s,R_{q''}^s}$	Decision variable. Is 1 if task q'' is executed on storage resource j and is 0 otherwise.
$x_{R_k^n,R_{q'''}^n}$	Decision variable. Is 1 if task q''' is executed on network resource k and is 0 otherwise.
$R_{e_i}^c, R_{s_i}^c, R_{m_i}^c, R_{ar_i}^c$	Provided specifications by computational resource i (number of cores, CPU speed, memory capacity, processor architecture respectively).
$R_{e,q'}^c, R_{s,q'}^c, R_{m,q'}^c, R_{ar,q'}^c$	Requested computational resource specifications (number of cores, CPU speed, memory capacity, processor architecture respectively) by task q'.
$R_{cap_j}^s, R_{st_j}^s, R_{sp_j}^s$	Provided specifications by storage resource j (storage capacity, storage type, storage speed respectively).
$R_{cap,q''}^s, R_{st,q''}^s, R_{sp,q''}^s$	Requested storage resource specifications (storage capacity, storage type, storage speed respectively) by task q''.
$R_{b_k}^n$	Provided specifications by network resource k (bandwidth).
$R_{b,q'''}^n$	Requested network resource specifications (bandwidth) by task q'''.
$T_{q,st}, T_{q,et}, T_{q,d}$	Start time, Execution time and Deadline of task q.
$ST_{i,q,q^\wedge}$	Decision variable. Is 1 if two different tasks q and $q^\wedge$ are scheduled to resource i and task q will start before task $q^\wedge$ and is 0 otherwise.
$ST_{i,q^\wedge,q}$	Decision variable. Is 1 if two different tasks q and $q^\wedge$ are scheduled resource i and task $q^\wedge$ will start before task q and is 0 otherwise.

$$min \left(\sum_{i=1}^{M} \sum_{q'=1}^{Q'} \overset{[a]}{C_{R_i^c,R_{q'}^c} x_{R_i^c,R_{q'}^c}} + \sum_{j=1}^{O} \sum_{q''=1}^{Q''} \overset{[b]}{S_{R_j^s,R_{q''}^s} x_{R_j^s,R_{q''}^s}} + \sum_{k=1}^{P} \sum_{q'''=1}^{Q'''} \overset{[c]}{N_{R_k^n,R_{q'''}^n} x_{R_k^n,R_{q'''}^n}} + \overset{[d]}{CS} \right) \quad (1)$$

s.t.

$\sum_{q'=1}^{Q'} x_{R_i^c,R_{q'}^c} R_{e,q'}^c \leq R_{e_i}^c$ (1.1)

$\sum_{q'=1}^{Q'} x_{R_i^c,R_{q'}^c} R_{s,q'}^c \leq R_{s_i}^c$ (1.2)

$\sum_{q'=1}^{Q'} x_{R_i^c,R_{q'}^c} R_{m,q'}^c \leq R_{m_i}^c$ (1.3)

$\sum_{q'=1}^{Q'} x_{R_i^c,R_{q'}^c} R_{ar,q'}^c \leq R_{ar_i}^c$ (1.4)

$\sum_{q''=1}^{Q''} x_{R_j^s,R_{q''}^s} R_{cap,q''}^s \leq R_{cap_j}^s$ (1.5)

$\sum_{q''=1}^{Q''} x_{R_j^s,R_{q''}^s} R_{st,q''}^s \leq R_{st_j}^s$ (1.6)

$\sum_{q''=1}^{Q''} x_{R_j^s,R_{q''}^s} R_{sp,q''}^s \leq R_{sp_j}^s$ (1.7)

$\sum_{q'''=1}^{Q'''} x_{R_k^n,R_{q'''}^n} R_{b,q'''}^n \leq R_{b_k}^n$ (1.8)

$\sum_{i=1}^{M} x_{R_i^c,R_{q'}^c} = 1$ (1.9)

$\sum_{j=1}^{O} x_{R_j^s,R_{q''}^s} = 1$ (1.10)

$\sum_{k=1}^{P} x_{R_k^n,R_{q'''}^n} = 1$ (1.11)

$x_{R_i^c,R_{q'}^c} \left(T_{q',st} + T_{q',et}\right) \leq T_{q',d}$ (1.12)

$x_{R_j^s,R_{q''}^s} \left(T_{q'',st} + T_{q'',et}\right) \leq T_{q'',d}$ (1.13)

$ST_{i,q,q^\wedge} + ST_{i,q^\wedge,q} = 1 \ if \ (q \neq q^\wedge) \ and \ (x_{R_iR_q} = 1) \ and \ (x_{R_iR_{q^\wedge}} = 1)$ (1.14)

$T_{q,st} \geq ST_{i,q^\wedge,q} * (T_{q^\wedge,st} + T_{q^\wedge,et})$ (1.15)

$T_{q^\wedge,st} \geq ST_{i,q,q^\wedge} * (T_{q,st} + T_{q,et})$ (1.16)

$x_{R_i^c,R_{q'}^c} \in \{0,1\}, i = 1, \dots M, q' = 1, \dots Q'$ (1.17)

$x_{R_j^s,R_{q''}^s} \in \{0,1\}, j = 1, \dots O, q'' = 1, \dots Q''$ (1.18)

$x_{R_k^n,R_{q'''}^n} \in \{0,1\}, k = 1, \dots P, q''' = 1, \dots Q''', \forall \ T_q$ (1.19)

Model 1: ILP Optimization Model

Part [a] of the model focuses on the total power consumed by computational resources when executing tasks scheduled to them, while Part [b] calculates the total power consumption of storage resources. In addition to the storage resources, another two types of resources that consume power in the data center were considered namely, the network resources and cooling systems. To calculate the amount of power consumed by these resources part [c] and [d] were used respectively.

The total power consumed by computational resources varies with the utilization of the resource and can be estimated using equation 2 [44][45][46]:

$$C_{R_i^c,R_{q'}^c} = P_{R_{i,idle}^c} + \left(P_{R_{i,busy}^c} - P_{R_{i,idle}^c}\right) \times u_{q'} \qquad (2)$$

Where $C_{R_i^c,R_{q'}^c}$ is the estimated power consumption by the computational resource i, $P_{R_{i,idle}^c}$ is the power consumed when the resource is idle, $P_{R_{i,busy}^c}$ is the power consumed by the resource when executing the task q' scheduled to it, and $u_{q'}$ is a parameter that depends on the execution of task q' on resource i.

Moreover, the average power consumed by storage resources can be estimated using equation 3 [46]:

$$S_{R_j^s,R_{q''}^s} = \frac{P_{R_{j,idle}^s} \times 75\% + P_{R_{j,write}^s} \times 5\% + P_{R_{j,read}^s} \times 20\%}{100\%} \qquad (3)$$

Where $S_{R_j^s,R_{q''}^s}$ is the power consumed by storage resource j, $P_{R_{j,idle}^s}$, $P_{R_{j,write}^s}$, $P_{R_{j,read}^s}$ are the power consumed when the resource is idle, writing or reading respectively and the percentages denote the common percentage of the storage resource state duration.

Furthermore, in our proposed environment the power consumed by network resources ($N_{R_k^n,R_{q'''}^n}$) depends on the bandwidth provided by the resource while the power consumed by the cooling systems CS depends on its Coefficient of Performance (COP) [47]. The COP is used to measure the efficiency of the cooling system and it is defined as the ratio of the amount of power consumed by the resources to the amount of power consumed by the cooling system [47].

***The proposed model is subject to the following constraints*:**

Constraints 1.1 to 1.4 ensure that the computational specifications (number of cores, CPU speed, memory capacity and processor architecture) requested by tasks are satisfied in the obtained solution.

When the tasks demand for storage resources, the model guarantees through constraints 1.5 to 1.7 that their demand is not violated; therefor, a task cannot be assigned to a resource with less storage capacity, storage type, and speed, than the requested.

Constraint 1.8 ensures that the required network bandwidth is met.

Constraints 1.9 to 1.11 ensure that every task is executed by only one computational, storage, or network resource.

Constraints 1.12 and 1.13 ensure that the start time and the execution time of each task meets the deadline.

A resource cannot execute two different tasks at a specific time frame. Hence, constraints 1.14 to 1.16 ensure that the resource can start executing one of the tasks after ending the execution of the previous one.

3.3. Model Validation

The validation of the proposed model had been done using Lingo™. Lingo™ gives an exact solution to the provided model that includes the objective function , and constraints [48]. It also provides different kind of reports that can be used in decision making and sensitivity analysis.

To test the quality of the solution given by Lingo and the time taken by this solver to find a solution various experiments were conducted for different number of tasks and different number of resources. Tasks differ in the type of capabilities requested (number of cores, memory capacity, storage capacity, storage types, etc.) and resources differ in the type and amount of capabilities provided. The values shown in Table 3.2 were used for the number of tasks and resources.

Table 3.2: No. of Tasks and Resources

Experiment No.	No. of Tasks	No. of Resources
1	6	10
2	15	14
3	30	18
4	60	32
5	120	64

For each experiment, Lingo gives the solution in terms of a schedule that shows which task will be executed by which resource and the total energy consumed when executing this schedule. It also shows the time taken by the solver to find this exact solution. Table 3.3 illustrates the results obtained from solving the proposed model using Lingo solver for different number of tasks and resources.

Table 3.3: Lingo™ Objective Value and Elapsed Time

Experiment No.	No. of Tasks	No. of Resources	Objective Value	Lingo Elapsed Time (sec)
1	6	10	35	3.46
2	15	14	175	1460.38
3	30	18	590	86400
4	60	32	N/A	N/A
5	120	64	N/A	N/A

Figure 3.2 illustrates the objective values which are the optimum solutions provided by Lingo. The experiment number from Table 3.3 is presented in the x-axis, and the total power consumption over time (the objective value) in the y-axis. As shown in the figure, the power consumption increases with the increase in the number of tasks because either more resources will be provided to execute these tasks or the provisioned resources will run for longer times which will result in more power. However, the power values shown in the figure are the minimum values for executing each category of tasks in different resources.

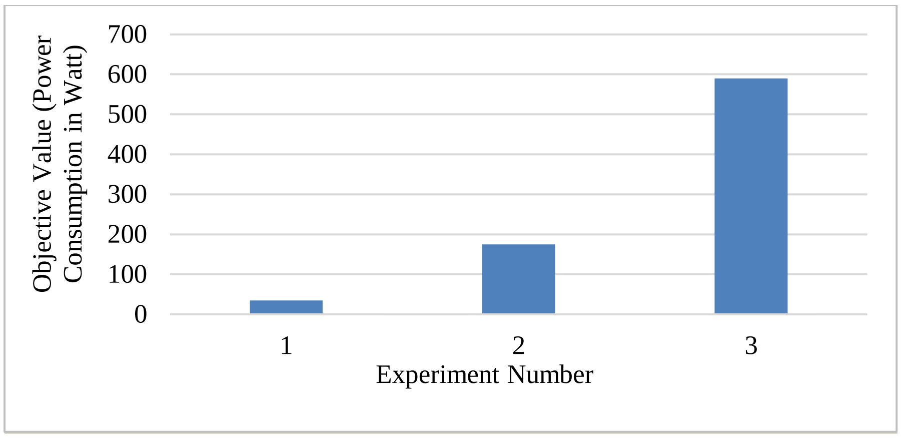


Figure 3.2: Total Power Consumption, Lingo™

One of the drawbacks of using Lingo that was clearly observed throughout the experiment is that it takes very long time to find a solution. Figure 3.3 shows the exponential increase in the elapsed time (y axis) with the increase in the number of scheduled tasks (x axis). It took 24 hours to find the exact solution for scheduling 30 tasks.

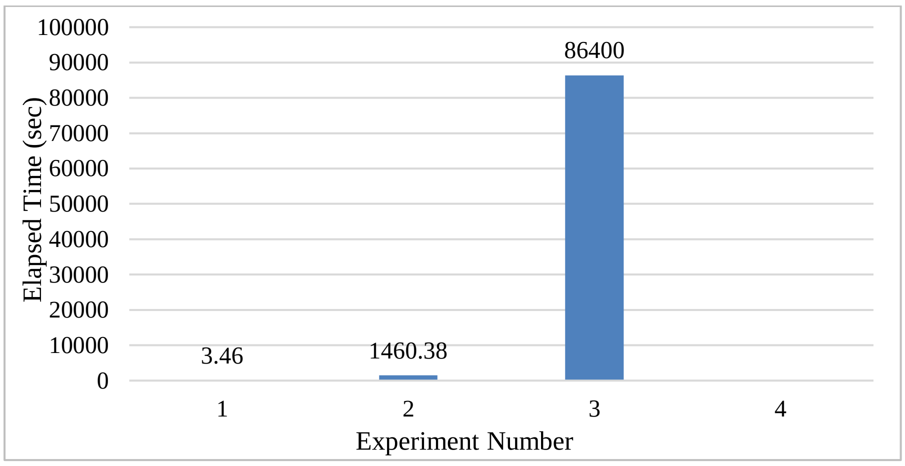


Figure 3.3: Elapsed Time, Lingo™

This work has not been extended for 60 tasks and 120 tasks because Lingo has been running for 10 days without providing a solution. Therefore, the genetic algorithm is implemented to overcome this drawback; to identify whether it will give a feasible solution in less time; and to compare the quality of the solution provided.

Chapter 4. Genetic Algorithms for Solving the ILP Model

In the Cloud environment, thousands of tasks will be received for scheduling and hence it is not feasible to make a scheduling decision in a very long time. Therefore, in this Chapter, we implemented Genetic Algorithm to find a near optimal solution to the proposed ILP model and the algorithm is validated by comparing the quality of the solution returned to the exact solution. Furthermore, since the Cloud environment is dynamic in nature i.e. tasks and resources can join and disjoin at any time, an Adaptive GA is proposed to minimize the energy consumption in a datacenter while keeping the tasks deadline into consideration.

4.1. Genetic Algorithm

Genetic algorithms are based on evolution by natural selection process to achieve efficient optimization [49]. This algorithm generates a population of chromosomes where each chromosome represents a possible solution to the problem. It starts by the Initialization phase which consists of randomly created solutions, also called individuals, for the initial population. Each individual is evaluated based on a fitness function and given a value that represents its fitness level. The creation of new generation (reproduction) from the current population is done by crossover and mutation. Every two chromosomes will be mated and a random location of the two mating chromosomes will be picked and swapped. Mutation is done by changing the value of the chromosome genes to another value within the allowed range of values to create a new offspring from the current population. These phases will be repeated until a termination condition is satisfied or the fitness value of the best individual did not change for a defined number of iterations.

Algorithm 4.1 illustrates the general idea of the Genetic Algorithm used in finding a solution for the ILP model. The algorithm inputs all the tasks to be scheduled and starts by the random generation of the initial population based on the resources that can execute each task. A chromosome is thus a vector of genes representing a possible solution to the optimization problem (possible schedule). The fitness of each chromosome (a schedule) is evaluated with respect to objective function (1) of the proposed model. Best-fit chromosomes, of a current population, are selected for crossover and mutation (reproduction). The mated parents are used for the creation of the new offspring that is evaluated for pinpointing the feasible solutions i.e. satisfy the given constraints, and penalize the unfeasible ones. This process is repeated until the termination criterion is satisfied and the fittest-so-far individual of all created population is preserved.

```
Initialize POP size;
Random generation of initial population of POP individuals;
Evaluate fitness of individuals;
repeat
    for (i=1 to POP step 2) do
        Randomly select 2 parents from the chromosomes;
        Apply crossover and mutation;
    endfor
    Evaluate fitness of new offspring population;
    Check feasibility of individuals;
    Do penalization;
    Preserve the fittest-so-far (elitism);
until (termination criterion is satisfied)
Solution = Fittest Chromosome (Best Schedule);
```

Algorithm 4.1: Genetic Algorithm

Before the initialization phase, the algorithm goes through all the tasks and builds a list that contains all the resources capable of executing each task based on the requested capabilities and the available resource capabilities.

4.1.1. Encoding of Chromosomes. The GA population is an array of individuals of length l = number of Tasks×2. Each individual is a vector G of l genes that corresponds to a possible solution (a schedule). A Gene[i], i is even, is assigned a task to be scheduled and G[i+1] is a resource from the list of resources that can execute this task. As an example, in the chromosome called Individual-1 in Figure 4.1, Tasks 1 and 3 are scheduled to Resources 6 and 5, Tasks 4 and 5 are scheduled to Resources 2 and 4, respectively, etc.

4.1.2. Initial Population. Tasks and the list of resources that can execute each task are used in the generation of the population. For each chromosome, Algorithm 4.2 starts by selecting a random task and a corresponding resource from the list of resources that can execute this task. The generation of population is done in a way that all tasks must be scheduled.

```
Input: Tasks Batch (tasks_batch), Population Size (pop_size), Chromosome Length
(chrom_length), Tasks-Resources List (t_r_list), Number of Tasks (num_tasks), Number
of Resources (num_res)
Output: Population (pop_chroms)

for (i=1 to pop_size) do
        for (j=1 to chrom_length step 2) do
        index = random() % num_tasks
        selected_task =  tasks_batch[index]
        pop_chroms[i,j] = setected_task
                for (k=1 to num_res) do
                selected_resource= t_r_list[selected_task, k]
                        if (selected_resource=0)
                        choose another resource from the t_r_list
                        else
                        pop_chroms [i,j+1]= selected_resource
                end for
        end for
end for
```

Algorithm 4.2: Initialization of Population

4.1.3. Fitness Function. For each schedule (chromosome) of the population, the fitness function (Algorithm 4.3) calculates the total power consumed which is the sum of the power consumed by each resource when executing the tasks scheduled to it in a specific time in that chromosome. That is the fitness of the given chromosome is calculated using function (1) of the proposed model (the objective function).

```
Input: Population (pop_chroms), Population size (pop_size), Chromosome Length
(chrom_length), Power Consumption Array (power)
Output: Population Fitness (pop_fitness)

for (i=1 to pop_size) do
pop_fitness[i]=0
end for
for (i=1 to pop_size) do
        for (j=1 to chrom_length) do
        task = pop_chroms [i,j]
        resource = pop_chroms [i,j+1]
        pop_fitness[i]= pop_fitness[i]+ power[task,resource]
        end for
end for
```

Algorithm 4.3: Fitness Function

4.1.4. Selection. In algorithm 4.4, the traditional Roulette Wheel method is used to select two parents from the current population. These two parents are mated (crossover) to reproduce a new offspring population.

```
Input: Population (pop_chroms), Population Probability (pop_prob), Sum of the
Probabilities (prob_sum), Population size (pop_size), Chromosome Length (chrom_length)
Output: Parents (parents)
Number of parents =2

for (i=1 to number of parents) do
num= random()% prob_sum
cumulative_prop = 0.0
        for (j=1 to pop_size) do
        cumulative_prop+= pop_prob[j]
                if (num<=cumulative_prop)
                selected_chrom=j
                for (k=1 to chrom_length) do
                        parents[i][k]= pop_chroms [j][k]
                end for
        break
        end for
end for
```

Algorithm 4.4: Selection Based on Roulette Wheel Method

4.1.5. Crossover. The crossover takes two individuals as input and 50% of the tasks from the two individuals will be selected to be crossed randomly. As shown in Algorithm 4.5, the selected set will be in the form of a task and a resource [T, R]. The algorithm checks if swapping the resources of the two individuals respects constraints 1.1 to 1.8 of the proposed model which makes sure that the tasks can be executed at the allocated resource. In addition, in order to produce better individuals, the algorithm checks if the crossover will result in minimizing the power consumption either for one or both individuals (i.e. the execution of individual-1 task in individual-2 resource will result in less power than executing it in the resource assigned to it in individual-1 and/or the same for individual-2 task). If so, then swapping is carried out and the process repeats over all the tasks that are selected for crossover. Figure 4.1 illustrates an example for the crossover of two individuals.

```
Input: Parents (parents), Power Consumption Array (power), Number of Tasks
(num_tasks)

Let X be cut points; X =50% of the tasks;
for (j=1 to X) do
        task_index = random ( )
        Individual-1_task = Individual-1[ task_index]
        Individual-1_resource = Individual-1[ task_index +1]
        Individual-2_task = Individual-2[ task_index]
        Individual-2_resource= Individual-2[ task_index +1]

                if (Individual-2_resource can execute Individual-1_task and
Individual-1_resource can execute Individual-2_task)
                        if (Individual-2_resource can execute Individual-1_task with
less power or Individual-1_resource can execute Individual-2_task with less power)
                Individual-1[ task_index +1] = Individual-2_resource
                Individual-2[ task_index +1] = Individual-1_resource
end for
```

Algorithm 4.5: Crossover

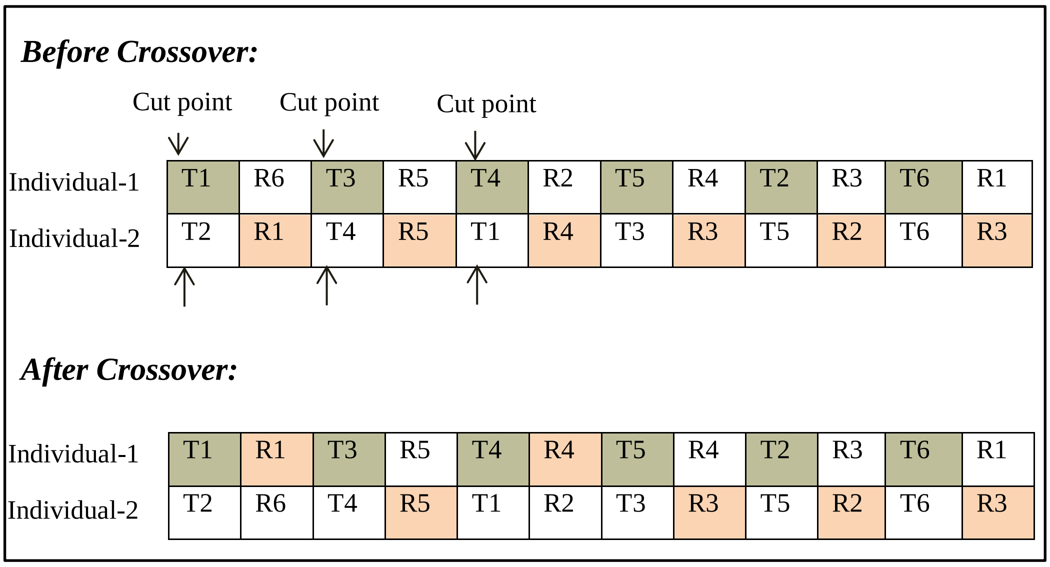

Before Crossover:
Cut point
Cut point
Cut point
Individual-1
T1 R6 T3 R5 T4 R2 T5 R4 T2 R3 T6 R1
Individual-2
T2 R1 T4 R5 T1 R4 T3 R3 T5 R2 T6 R3
After Crossover:
Individual-1
T1 R1 T3 R5 T4 R4 T5 R4 T2 R3 T6 R1
Individual-2
T2 R6 T4 R5 T1 R2 T3 R3 T5 R2 T6 R3

Figure 4.1: Crossover Example

4.1.6. Feasibility Check and Penalization.

4.1.6. Feasibility Check and Penalization. Before checking the feasibility of each chromosome i.e. determine if it satisfies constraints 1.12 to 1.16 of the ILP model, the algorithm determines the time each resource will start executing the tasks scheduled to it. This time, i.e. the start time, depends on the current schedule. That is, if two different tasks are scheduled to the same resource in the current schedule; the resource must execute one of these tasks after the execution of the other task (constraints 1.14 to 1.16) since a resource cannot execute two different tasks at the same time (no preemption).

The feasibility of a chromosome depends whether the tasks in the chromosome can be executed before their deadline or not. As shown in Algorithm 4.6, if for at least one task in the chromosome, the $\left(T_{q,st} + T_{q,et}\right) > T_{q,d}$, then the chromosome is infeasible and is penalized by assigning it a very large fitness value. In this way, infeasible solutions will not be selected for reproduction.

```
Input: Population (pop_chroms), Population size (pop_size), Chromosome Length
(chrom_length), Expected Execution Time Matrix (Exec_time), Population Fitness
(pop_fitness), Deadline Array

for ( i  =1 to pop_size) do
        for ( j=1 to chrom_length) do
        Task = pop_chroms [i,j]
        Resource= pop_chroms [i,j+1]

//Penalization
        if (start_time[i][Task]+ Exec_time[Task][ Resource]> Deadline[Task])
                pop_fitness[i] = pop_fitness[i] × 10;
        end for
end for
```

Algorithm 4.6: Feasibility Check and Penalization, GA

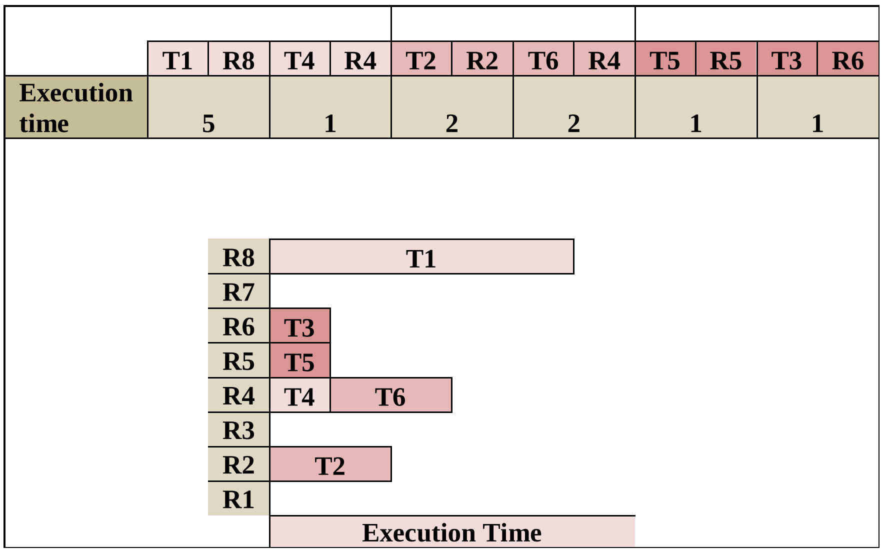
4.1.7. Elitism. For every population generated, Algorithm 4.7 compares the minimum fitness in that population with the best fitness calculated so far (the minimum power consumed) and saves the best-so-far individual (i.e. feasible and has the least power consumption).

```
Input: Population (pop_chroms), Population Fitness (pop_fitness), Chromosome Length
(chrom_length)
Output: Best Chromosome (best_chrom), Best Fitness (best_fitness)

for (i=1 to pop_size) do
        if (pop_fitness[i]< best_fitness)
                best_fitness = pop_fitness[i]
        for (j=1 to chrom_length) do
                best_chrom[j] = pop_chroms[i][j]
                end for
end for
```

Algorithm 4.7: Elitism

4.1.8. Termination Criteria. According to these experiments, the algorithm stops after 100 iterations or when the fitness of the best-so-far did not change for 4 consecutive iterations. The solution returned contains the minimum fitness value found, the schedule (chromosome) with that fitness and the execution time of every task as shown in Figure 4.2.



	T1	R8	T4	R4	T2	R2	T6	R4	T5	R5	T3	R6
Execution time	5		1		2		2		1		1	

Figure 4.2: Fittest Chromosome

4.1.9. GA Experimentation. For testing the solution returned by the genetic algorithm and evaluating how far it is from the exact solution given by Lingo in terms of quality and time, the same experiments illustrated in Table 3.2 were conducted. The following values are used throughout the experiment:

Population Size: 20

Number of Iterations: 100

Stopping Condition: either Number of iterations or if the fitness value didn't change for 4 iterations.

Number of tasks and Number of Resources: the values shown in Table 3.2 were used.

Table 4.1 illustrates the results obtained from finding a solution to the proposed model using the Genetic algorithm for different number of tasks and resources.

Table 4.1: GA Fitness and Elapsed Time

Experiment No.	Num_Tasks	Num_Resources	GA Fitness	GA Elapsed Time (sec)
1	**6**	**10**	45	0.53
2	**15**	**14**	180	0.78
3	**30**	**18**	600	1.545
4	**60**	**32**	949	3.868
5	**120**	**64**	1991	9.828

Figure 4.3 illustrates the fitness values which are the best solutions provided by the GA. The x-axis presents the experiment number from Table 4.1 and the total power consumption per schedule (the fitness value) is in the y-axis. The increase in the power consumption with the increase in the number of tasks is due to either more resources are provisioned or the already running resources will run for longer times to execute the new tasks.

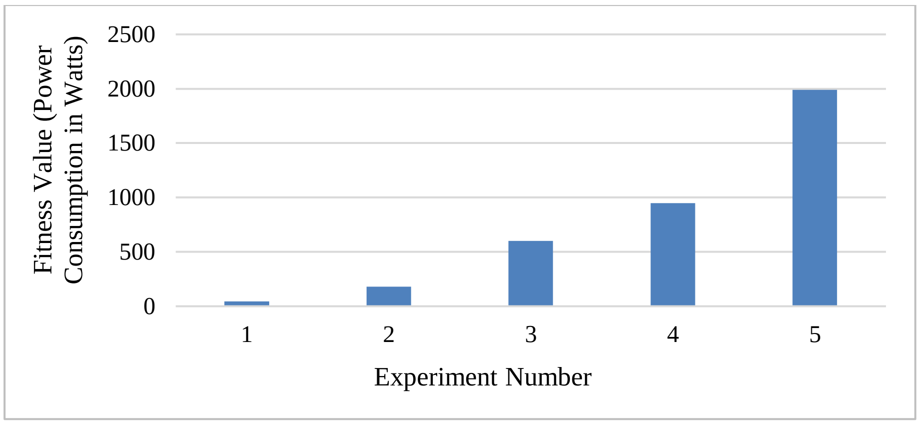


Figure 4.3: Total Power Consumption using GA

Figure 4.4 shows the time taken by the genetic algorithm to find the best solution for the different number of tasks and resources. The algorithm finds a solution very fast when the number of tasks is relatively small and the time increase with the increase in the number of tasks.

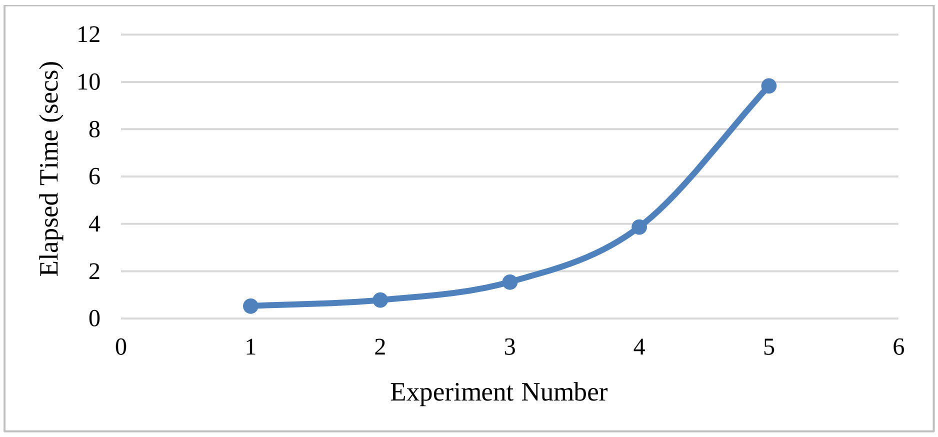


Figure 4.4: Elapsed Time, GA

4.2. Comparison between Lingo™ and the Genetic Algorithm

Figure 4.5 demonstrates a comparison between the values obtained from the validation of the proposed model using exact (Lingo) and heuristic (GA) methods. For the first three sets of experiments, the genetic algorithm gives results (power consumption) that are very close to the exact solution and this validates the algorithm. Furthermore, the algorithm provided a solution for the experiments set 4 and 5 since an exact solution could not be provided due to Lingo limitations.

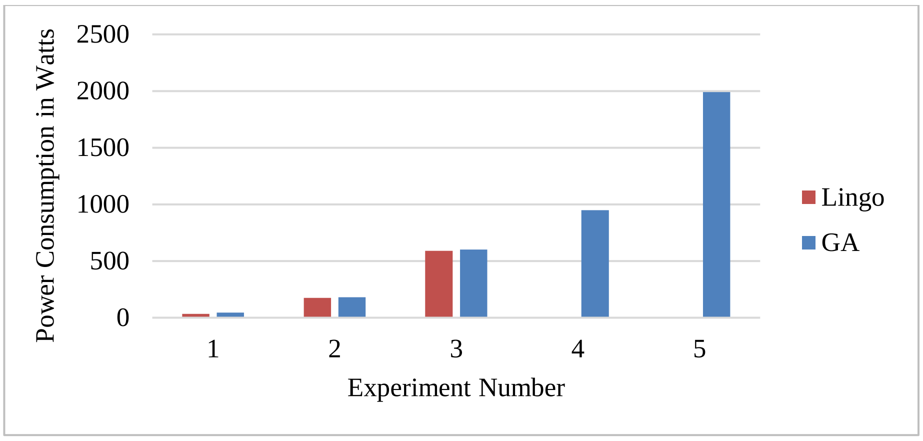


Figure 4.5: Total Power Consumption, Lingo™ vs. GA

On the other hand, Figure 4.6 shows Lingo took almost 24 hours to find a solution for scheduling 30 tasks (experiment 3) while the genetic algorithm took 1.5 seconds for the same set of tasks. Also, it took around 10 seconds to find a near optimal solution for 120 tasks (experiment 5) whereas Lingo was running for 10 days without providing an exact solution to the same set of tasks.

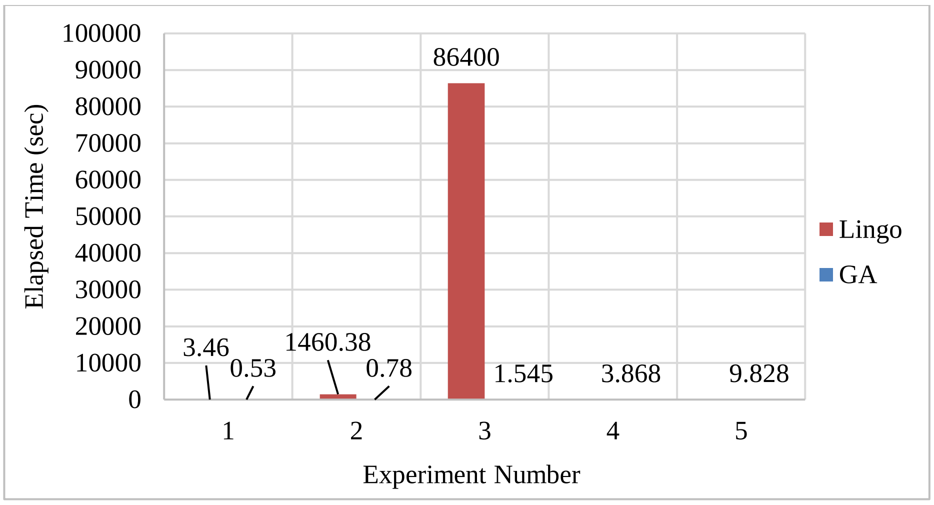


Figure 4.6: Elapsed Time, Lingo™ vs. GA

From the facts detailed above it is concluded that the genetic algorithm can be used for extending the experiments to larger number of tasks and resources and obtaining scheduling decisions. In the next sections, this algorithm is used for static scheduling i.e. all the tasks and resources are known in advance and the scheduling decision is made based on a prior knowledge of the tasks and resources. Moreover, a proposed algorithm (Adaptive GA) is developed for dynamic scheduling i.e. tasks and resources are not known in advance. The adaptive GA starts from the best schedules and puts unfinished and rescheduled tasks into consideration.

4.3. Adaptive Genetic Algorithm

Algorithm 4.8 illustrates the proposed Adaptive Genetic Algorithm used in minimizing power consumption over time in a Cloud datacenter. The Adaptive GA differs from the GA in steps (A-1 to A-8) because there is no prior knowledge about the tasks and resources and hence the arrival of tasks is random. The algorithm inputs an initial set of tasks, generates population for this set and evaluates it according to objective function (1) of the proposed model. Best-fit chromosomes, of a current population are selected for crossover and mutation yielding a new offspring population. The algorithm checks the feasible solutions in the new offspring to select the fittest among them as the best-so-far schedule. The process is repeated until the termination criterion is satisfied. The GA is adaptive in the sense, that after reaching a solution, it waits for some time, as determined by a delay function, till some new requests arrive, and then it starts the process of finding new solution considering the schedule assigned in the fittest-so-far chromosome of the previous solution and the new requests. In other words, whenever a new set of tasks is received, the algorithm finds the best schedule that minimizes the power consumption with respect to the previous schedule and the deadlines of the tasks.

In the Adaptive GA, the same encoding scheme from the GA is used but for every received batch of tasks. The proposed algorithm starts by making a list of the resources capable of executing each task based on capabilities.

The initialization is done for every batch of tasks received. The algorithm selects a random task from the received tasks and for each task it selects a random resource that can execute this task. In the GA, the received batch of tasks includes all the tasks to be scheduled while in the proposed Adaptive GA the batch contains rescheduled tasks from the previous schedules and newly arrived tasks.

```
Initialize **POP size**
A-1: Current Schedule is empty;
A-2: Let Tasks has an initial set of tasks;
A-3: repeat
Random generation of initial population of POP individuals;
Evaluate fitness of individuals;
          repeat
                    for (i=1 to POP/2) do
                              Randomly select 2 parents from the chromosomes;
                              Apply crossover and mutation;
                    endfor
                    Evaluate fitness of new offspring;
              A-4: Check feasibility of individuals;
                    Do penalization;
                    Preserve the fittest-so-far (elitism);
          until (termination criterion is satisfied)
Solution = Fittest Chromosome (Best Schedule);
A-5: Current Schedule = Schedule as in fittest-so-far chromosome;
A-6: Delay (based on selected distribution);
A-7: Update Tasks with new set of tasks;
A-8: until (True)
```

Algorithm 4.8: Adaptive GA

When it comes to fitness calculation, the Adaptive GA excludes the fitness of rescheduled tasks from the previous schedule and includes it when rescheduling these tasks to new resources. Moreover, the same selection and crossover scheme is used for the generation of new offspring with a size equal to the initial population size.

To check the feasibility of a chromosome, i.e. determine if it satisfies conditions 1.12 to 1.16, the Adaptive GA first determines the start time of executing each task in a specific resource. As shown in Algorithm 4.9, the start time depends on the previous schedule and the current schedule. That is, if the resource in the current schedule is assigned two tasks one in the previous schedule and one in the current schedule, this resource should start executing the task in the current schedule after the execution of the one in the previous schedule (no preemption of tasks). Also, if one or more tasks from the previous schedule should be rescheduled i.e. they do not start their execution except after the new set of tasks are received for scheduling, the resources of these tasks must be released and thus the start time of rescheduled tasks must be rescheduled. Furthermore, if two different tasks are scheduled to the same resource in the current schedule, the resource must execute one of these tasks after the execution of the other task (constraints 1.14 to 1.16) since a resource cannot execute two different tasks at the same time.

The feasibility of a chromosome depends whether the tasks of the chromosome can be executed before their deadlines or not. That is, a chromosome is feasible if for each Task in the chromosome, the $\left(T_{q,st} + T_{q,et}\right) \leq T_{q,d}$ (constraints 1.12 and 1.13). Otherwise, the chromosome is infeasible. An infeasible chromosome is penalized by assigning to it a very large fitness value. In this way, infeasible solutions will not be selected for reproduction, i.e., in the derivation of new offspring population.

As in GA, the Adaptive GA saves the best-so-far obtained individual of all generated populations. Elitism is done whenever a new batch of tasks arrives and the best schedule has to be found.

Likewise, for every batch of tasks, the algorithm stops when the termination criterion is satisfied. According to our experiments, this is done when the maximum number of generations 50 is reached or when the fitness of the fittest-so-far did not change for 7 consecutive iterations. The solution returned contains the minimum fitness value found, the schedule (chromosome) with that fitness and the start and execution time of every task.

The returned solution (the best schedule) will include tasks that should be rescheduled and this solution will be considered in deciding the start time of the new batch of tasks.

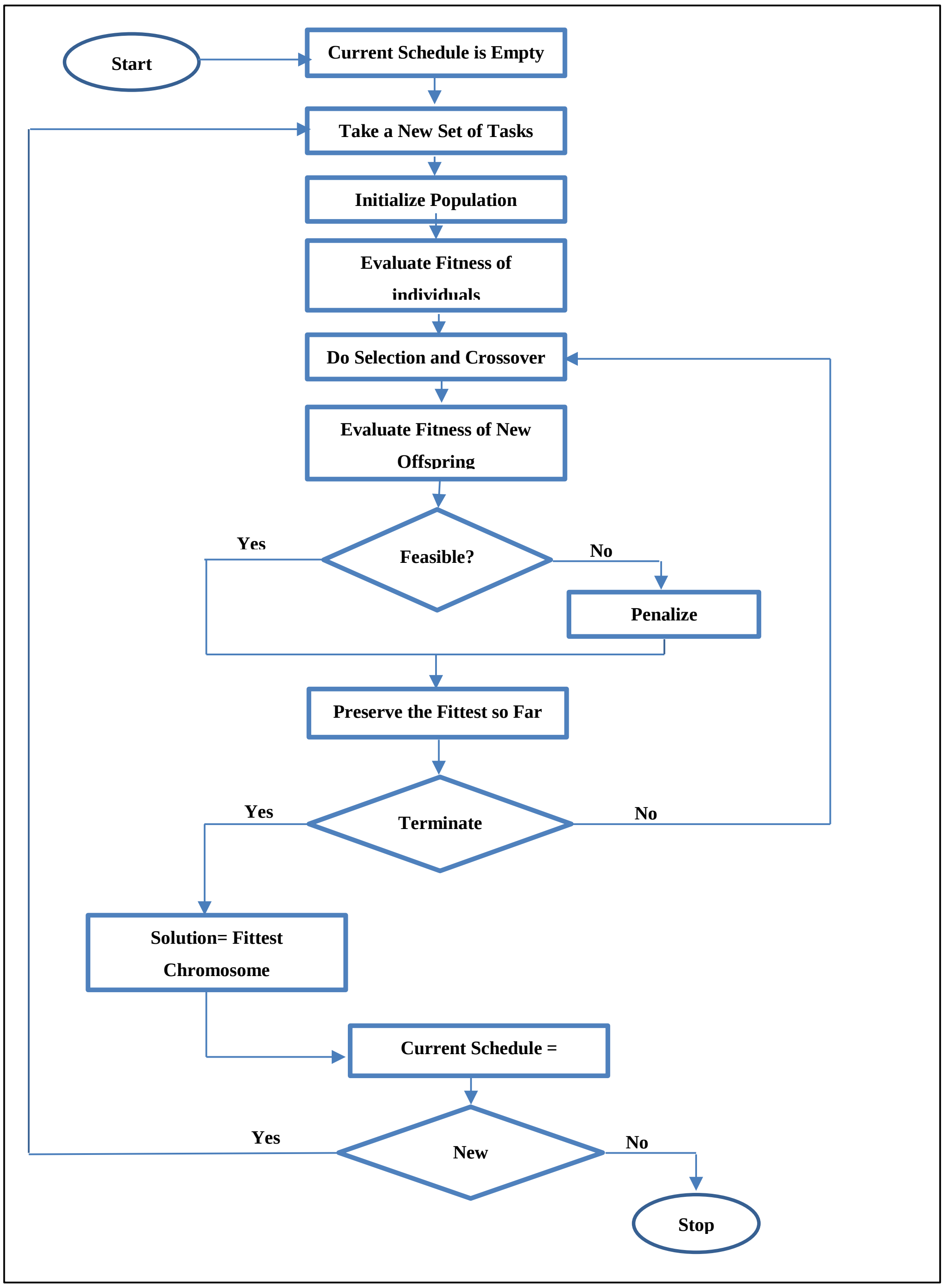
```
Input: Population (pop_chroms), Population size (pop_size), Chromosome Length
(chrom_length), Population Fitness (pop_fitness), Expected Execution Time (Exec_time),
Old Schedule (old_schedule), Deadline
Output: Start time (start_time), Penalization of infeasible chromosomes
for ( i  =1 to pop_size) do
    for ( j=1 to chrom_length) do
        start_time[i][j]=0;
    end for
end for
for ( i  =1 to pop_size) do
    for ( j=1 to chrom_length step 2) do
        Task = pop_chroms [i,j]
        Resource= pop_chroms [i,j+1]
        if (old_schedule is empty)
            //check only the current schedule
            for (k=1 to k<j step 2)
                if (pop_chroms[i][k]= Resource)
                start_time[i][Task]= start_time[i][k-1]+
Exec_time[pop_chroms[i][k-1]][ Resource]
            end for
        else
            //check old schedule and current schedule
            for (old =1 to chrom_length step 2 )
                if (old_schedule[old]= Resource)
                start_time[i][Task]= start_time[old-1]+
Exec_time[old_schedule[old-1]][ Resource]
            end for
            for (k=1 to k<j step 2)
                if (pop_chroms[i][k]= Resource)
                start_time[i][Task]= start_time[i][k-1]+
Exec_time[pop_chroms[i][k-1]][ Resource]
            end for
    //Penalization
    if (start_time[i][Task]+ Exec_time[Task][ Resource]> Deadline[Task])
        pop_fitness[i] = pop_fitness[i] × 10;
    end for
end for
```

Algorithm 4.9: Feasibility Check and Penalization, Adaptive GA

In the proposed environment, task arrival follows a random distribution, and after reaching a solution for every batch, the Adaptive GA waits for some time, according to the normal distribution, till new tasks arrive. Then it starts the process of finding the best schedule considering the old schedule and the deadline of the new set of tasks. The new set of tasks includes the newly arrived tasks and the tasks from the old schedule that did not start execution before the new tasks arrive. Rescheduling results in finding better solutions for rescheduled tasks (i.e. resources to execute rescheduled tasks with less power) and hence minimizing the energy consumption.

4.4. Adaptive Genetic Algorithm Workflow

Figure 4.7 illustrates the work flow of the Adaptive GA. This algorithm finds the best schedule for every newly arrived set of tasks.



Start
Current Schedule is Empty
Take a New Set of Tasks
Initialize Population
Evaluate Fitness of individuals
Do Selection and Crossover
Evaluate Fitness of New Offspring
Yes
Feasible?
No
Penalize
Preserve the Fittest so Far
Yes
Terminate
No
Solution= Fittest Chromosome
Current Schedule =
Yes
New
No
Stop

Figure 4.7: Adaptive GA Workflow

4.5. GA Parameters

Since genetic algorithms are non-deterministic, GA parameters such as population size not only influence the run-time of the algorithm to find a solution but also the achieved quality of the solution. Hence, experiments with population size ranging from 6 to 200 has been conducted in the GA and the Adaptive GA to determine the appropriate population size that results in better solution quality (i.e. fitness). Also, to do the experiments for different number of tasks and resources, we divided the problem size into three categories; *medium size problem* (i.e. 200 Task and 120 Resources), *Large size problem* (500 Task and 250 resources) and *X-Large size problem* (1000 Task and 500 Resource).

As shown in Figure 4.8 and 4.9, where the values of the population size are depicted in the x-axis and the fitness values (total power consumption over time) are presented in the y-axis, both the GA and the Adaptive GA -for the three problem sizes- give high fitness when the population size is small (6 and 10) but the fitness starts decreasing with population size of 40 and higher. The minimum (i.e., best) solution is obtained when population size is 200.

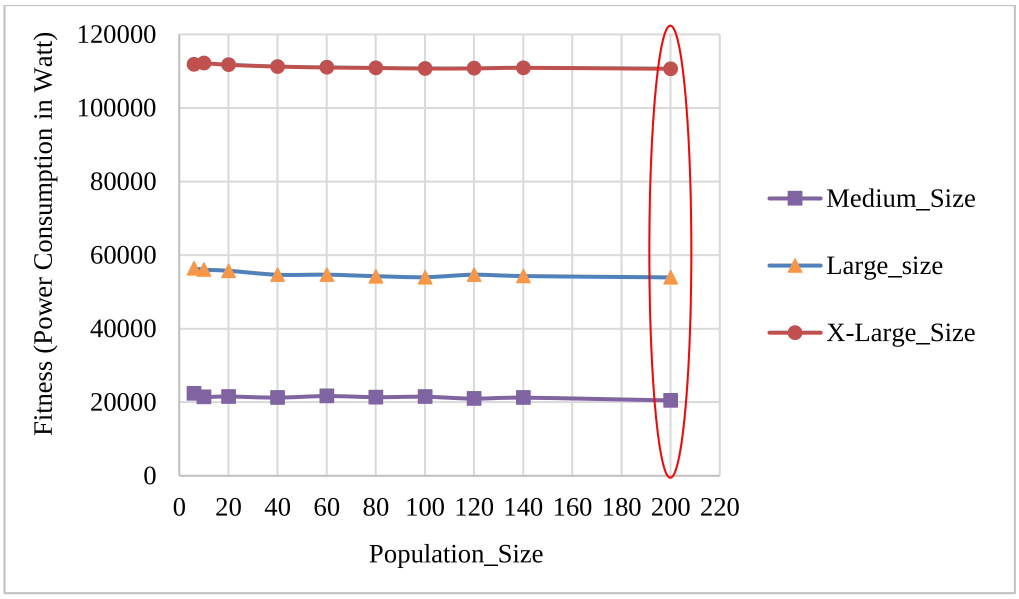


Figure 4.8: GA, Population Size vs. Fitness (Power Consumption)

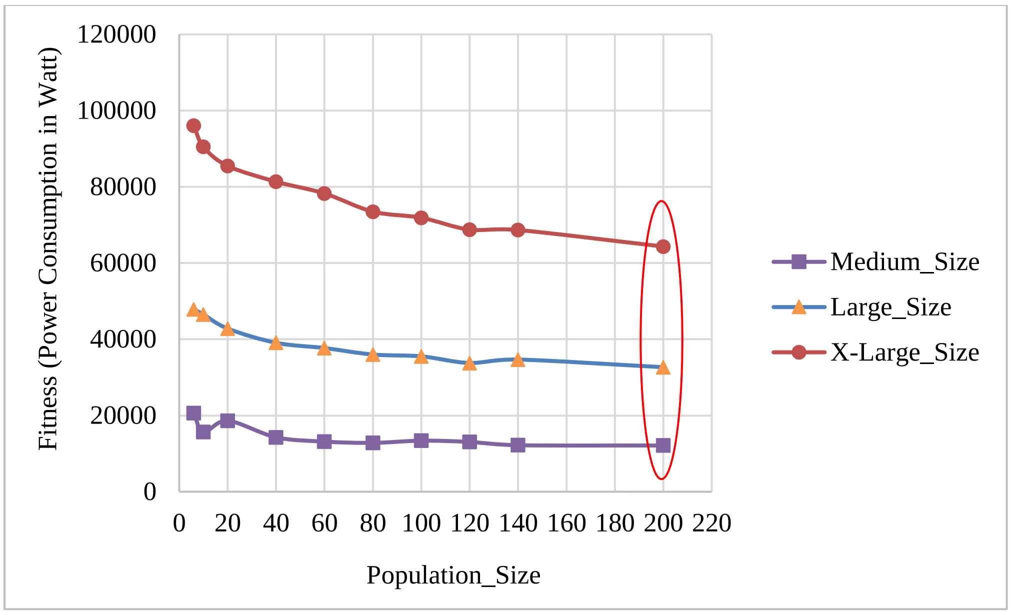


Figure 4.9: Adaptive GA, Population Size vs. Fitness (Power Consumption)

The increase in the population size gives the algorithm the opportunity to have better solutions (i.e. schedules with less power); as it allows the initialization of more chromosomes or individuals (i.e. possible feasible solutions). These individuals will contain, for the same batch of tasks, more resource options to execute them. This in turn increases the number of feasible solutions (i.e. schedules) and the number of solutions that minimize power consumption. Hence, since the population size of 200 gives the minimum fitness (i.e. power consumption), it will be used hereafter.

Another factor that affects the quality of the solution attained by the GA is the number of generations; that determine when the algorithm converges i.e. the number of generations to reproduce new offspring before the algorithm gives the best schedule. We conducted experiments for the three problem sizes and with a population size of 200 to pinpoint the number of generations that gives the best fitness for both GA and Adaptive GA. In Figures 4.10 and 4.11, the number of generations is presented in the x-axis and the fitness (i.e. power consumption over time) is in the y-axis. For the GA (Figure 4.10), medium size problem requires 140 generations to give the minimum fitness, large size problem requires 120 generations and the X-large size problem requires 50 generations, hence the number of generations in the GA depends on the size of the problem. For the Adaptive GA (Figure 4.11) on the other hand, all the problem sizes (medium, large and x-large) find the minimum fitness in 50 generations and hence the number of generations does not depend on the problem size.

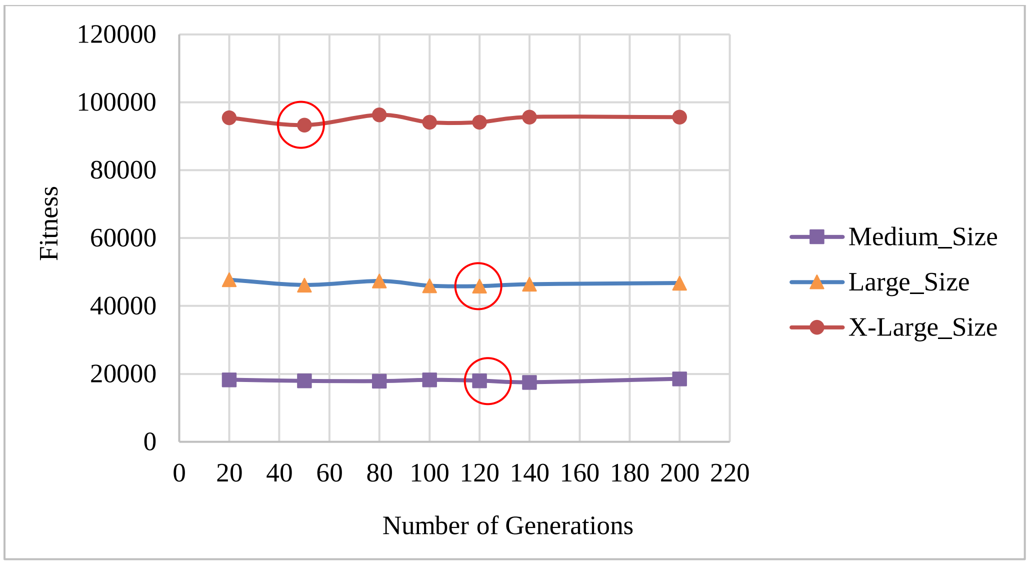


Figure 4.10: GA, Number of Generations vs. Fitness

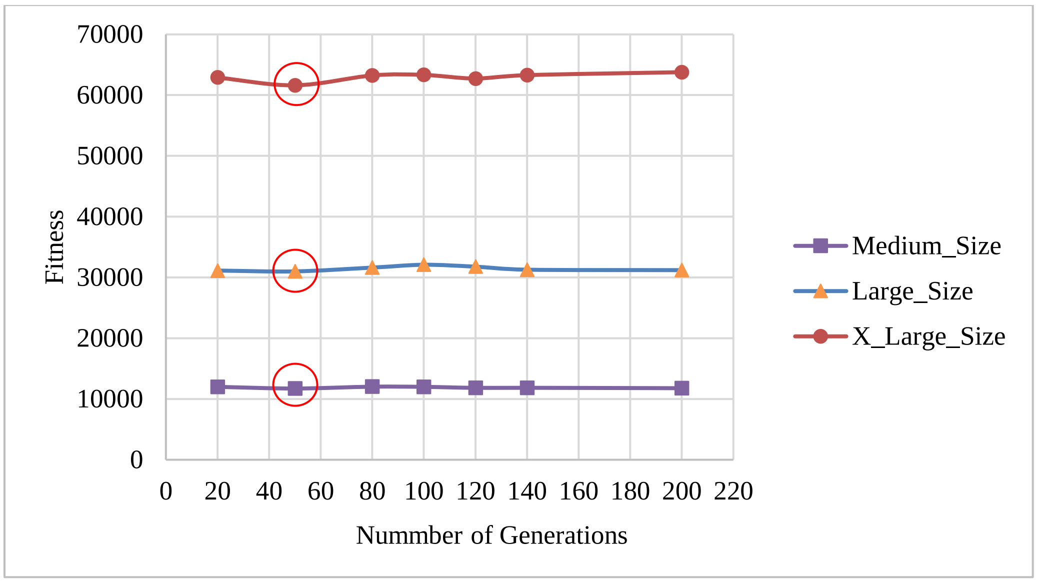


Figure 4.11: Adaptive GA, Number of Generations vs. Fitness

Concluded from the above experiments, in the subsequent sections population size of 200 and 50 generations will be used throughout the experimentation.

Chapter 5. Experimentation and Results

In this chapter, we present the simulation environment and results achieved for the implemented heuristic algorithm for optimizing energy consumption in the Cloud environment. We also compare the performance of the proposed Adaptive GA to the performance of the two implemented algorithms; Genetic Algorithm and the First Come First Serve Algorithm.

5.1. Simulation Environment

To evaluate and analyze the proposed algorithm, a simulated IaaS Cloud environment is created using C++ programming language. The simulated environment consists of one datacenter with different number of resources and tasks. The resources (Computational, Storage and Network) differ in the capabilities provided and the tasks differ in the capabilities requested and their deadline. The computational servers vary in the number of cores, processor speed, memory capacity and processor architecture, whereas storage servers differ in the storage capacity, storage type and storage speed. In the network resources, we consider the bandwidth with the assumption that the datacenter network topology was set in advance. The values shown in Table 5.1 are used as physical resources characteristics.

Table 5.1: Physical Resources Capabilities

Capability / Resource	No. of Cores	Processor Speed (GHz)	Memory Capacity (GB)	Processor Arch. (Bits)	Storage Capacity (TB)	Storage Speed (GHz)	Storage Type	Bandwidth (Gb/s)
Computational	5 to 40	2 to 7	64 to 256	32/64	------	------	------	------
Storage	------	------	------	------	0.5 to 1000	2 to 7	SSD/HDD	------
Network	------	------	------	------	------	------	------	10 to 100

Amongst the above-mentioned capabilities in Table 5.1, the computational resource capabilities have been used in previous research [50] while the storage and network capabilities are extracted from real systems specifications. In our simulated environment, the resources are generated randomly using these specifications. Furthermore, the tasks requirements and their respective deadline are randomly generated. In addition to this, every task has a start time which is determined by the algorithm and an execution time depending on the resource provisioned.

5.2. Simulation Results

To test the effectiveness of the proposed algorithm, we run the experiments for different number of tasks and resources as shown in Table 5.2. The following parameters are used in our simulation experiments:

Population Size: 200

Crossover: Multipoint

Stopping Condition: 50 generations or if the fitness of the best-so-far did not change for 7 consecutive iterations

Table 5.2: Number of Tasks and Resources

Experiment Number	Problem Size	No. of Tasks	No. of Resources
1	**Medium**	200	120
2	**Large**	500	250
3	**X-Large**	1000	500

The experiments were conducted to calculate the power consumption (in Watt) per scheduling decision when the proposed Adaptive GA is used to find the best schedule for different number of tasks and resources. We generated tasks and resources using various combinations of the capabilities shown in Table 5.1. For each problem size (Table 5.2), the results are obtained as the average of 3 different runs with each run having the same number of tasks and the number of resources but with different capabilities. The experiments were obtained using Intel(R) Core™ i7 6500U CPU @ 2.50GHz with 4 GB RAM machine under Windows 10 (64 bit).

Our proposed Adaptive GA schedules task sets as they arrive. Hence, the number of tasks shown in Table 5.2 is the total number of tasks scheduled. Figure 5.1 illustrates the fitness values which are the best solutions provided by the proposed algorithm. The problem sizes which are mapped to the total number of tasks and resources (from Table 5.2) are presented in x-axis and in the y-axis, the fitness value which is the total power consumed (in Watts) when scheduling all the tasks for each problem size is presented. The increase in the power consumption over time with the increase in the problem size is due to either more resources are provisioned or the already running resources will run for longer times to execute the newly arrived tasks.

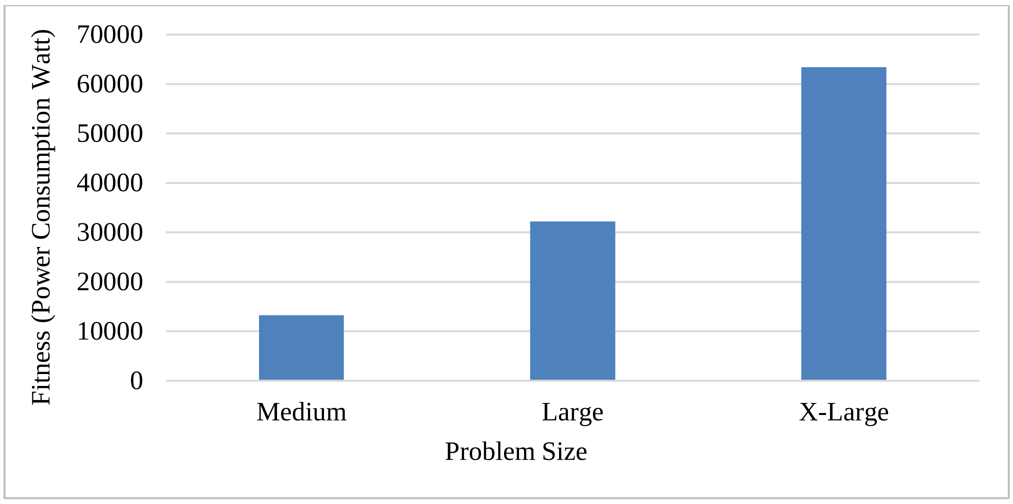


Figure 5.1: Total Power Consumption, Adaptive GA

On the other hand, the time taken by the algorithm to schedule all the tasks (Table 5.2) increases exponentially with the increase in the problem size. In Figure 5.2, the problem size is presented in the x-axis and the elapsed time in secs is presented in the y-axis. This growth is due to the longer time the algorithm takes to schedule all the received tasks (i.e. find a schedule for all batches of tasks received).

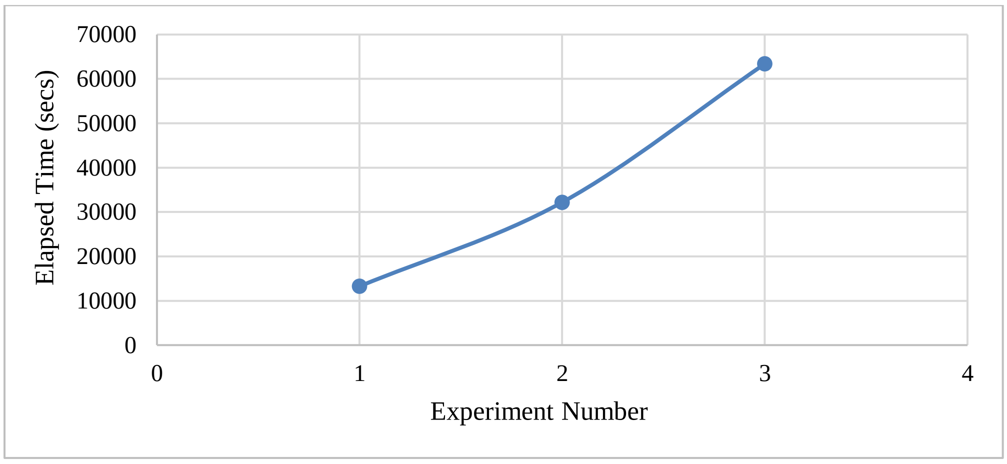


Figure 5.2: Total Elapsed Time, Adaptive GA

However, regardless of the problem size (i.e. total number of tasks), the algorithm takes almost the same time for finding the best schedule for every batch of tasks received. As shown in Figure 5.3, in the x-axis, we have the number of tasks (between 5 and 20) in every batch received and in the y-axis, we have the total time in seconds to find a schedule. The three problem sizes (Medium, Large and X-Large) differ in the total number of batches (i.e. more tasks mean more batches) but the time taken by the algorithm to find a schedule for all the three problem sizes varies with respect to the number of tasks in the batch from 1.7 to 8 seconds.

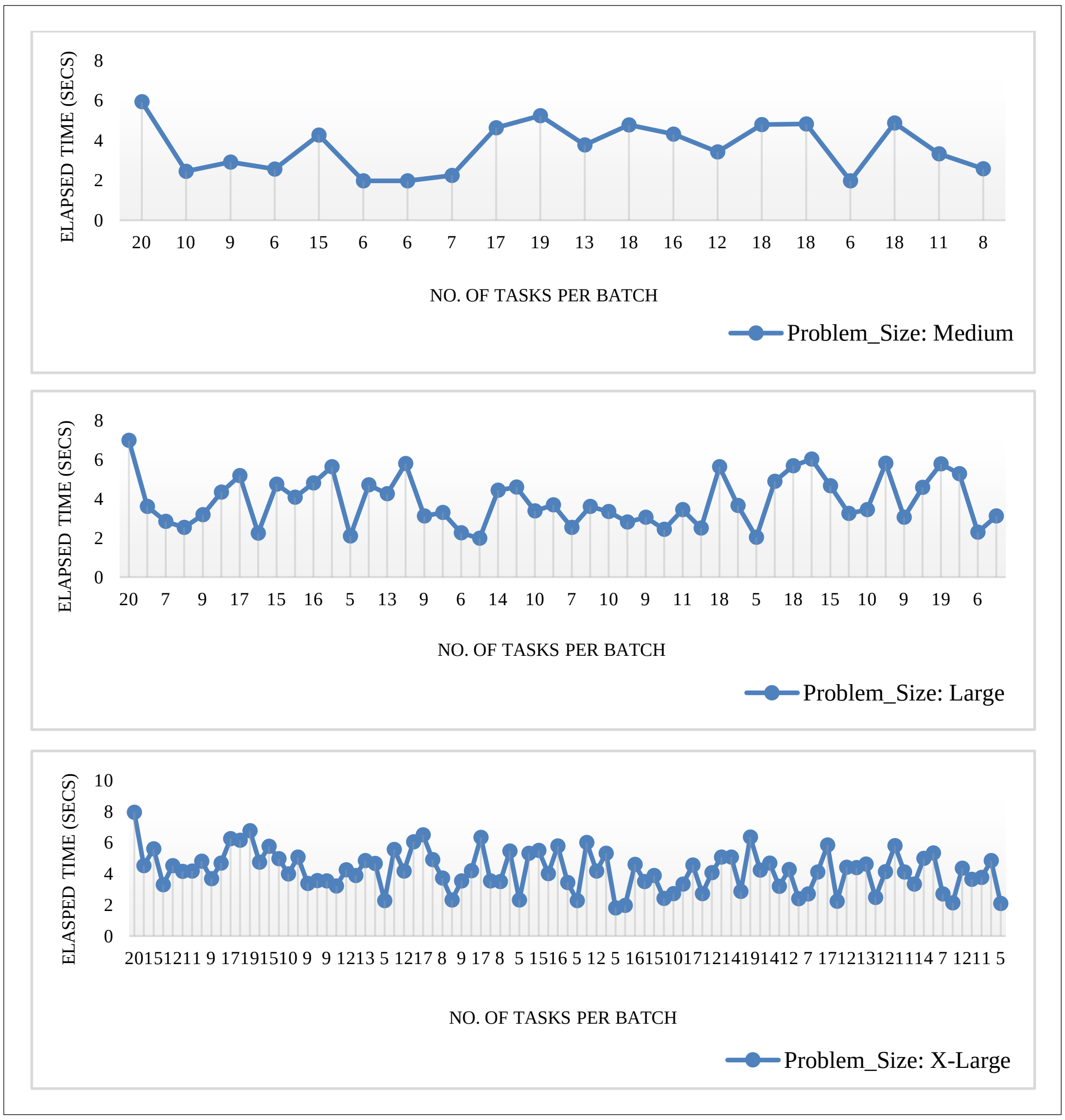


Figure 5.3: Problem Size vs. Elapsed Time per Tasks Batch

In the next section, the proposed Adaptive GA is compared to the GA implemented in section 4.1 and the conventional First Come First Serve Algorithm.

5.3. Performance Evaluation

The GA implemented in section 4.1 for finding a near optimal solution to the proposed ILP model, uses static scheduling (i.e. the tasks and resources are known in advance and a schedule is defined for all the tasks with less run-time overhead) whereas the Adaptive GA uses dynamic scheduling (i.e. tasks and resources can join and disjoin at any time) with batch mode strategy. Both algorithms are energy-aware scheduling algorithm with the objective of minimizing energy consumption in IaaS Cloud. In this section, the GA is used with the same Adaptive GA parameters from section 5.2, to find the best schedule for the problem sizes in Table 5.2 and is compared to the Adaptive GA in terms of the achieved quality of the solution (i.e. power consumption).

In Figure 5.4, we compare the Adaptive GA and the GA. In the x-axis, we have the different problem sizes and in the y-axis, we have the total power consumed over time when scheduling the tasks to resources in each category. The Adaptive GA had minimized the power consumption by 32% to 40% in comparison to the GA. This is due to the fact that in the GA, the initial population has the main impact in the new offspring populations, i.e. the resources selected for executing all the tasks in the initial population will be the same used in the reproduction and the generation of the new offspring population. Hence, the rest of the available resources, may result in less power, are not considered and the solution space depends mainly on the feasible solutions in the initial population. Whereas in the Adaptive GA, whenever a new set of tasks arrives, the algorithm checks the previous schedule for the tasks that did not start execution and it will reschedule these tasks with the new set of tasks. Rescheduling yields the release of the resources from the previous schedule. The Adaptive GA considers all the available resources (not only the ones from the initial schedule) for scheduling the new set of tasks, i.e. the set includes rescheduled and newly arrived tasks. This gives the Adaptive GA the possibility of selecting resources to the rescheduled tasks that can execute them with less power than the resources in the previous schedule.

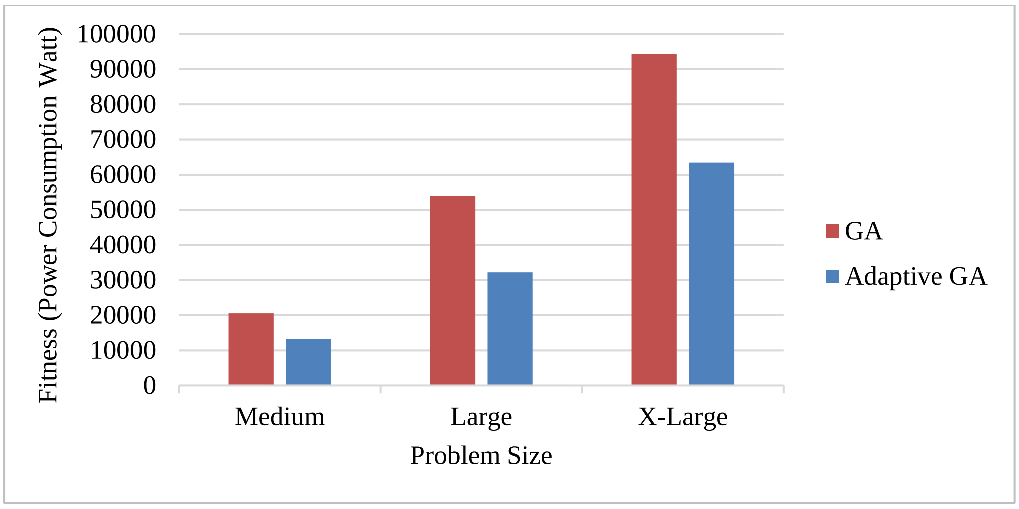


Figure 5.4: Adaptive GA vs. GA

We implemented the non-energy-aware scheduling algorithm, First Come First Serve (FCFS). This algorithm schedules the tasks as they arrive, to the first resource that can execute them. In our implementation of the FCFS scheduling algorithm for the dynamic Cloud environment, tasks arrival follows a random distribution. For every received batch of tasks, the algorithm starts by initializing a list of the resources capable of executing each task in the batch, based on the requested capabilities per task and the provided capabilities by resources (i.e. not all resources can execute every task).

The algorithm schedules the tasks in the received batch in a first come first served order. For every task, it will search for a resource from the list of resources that can execute this task and is free (i.e. a resource that is not executing any task), it schedules the task to this resource and determines the start time of this task in a way that meets its deadline. If all resources are provisioned (i.e. no free resource), the algorithm searches for a resource that can execute the selected task before its deadline. This is done by checking whether the selected resource is used in the current schedule or the previous schedule. In both cases, the start time of the new task should be after the execution of the old ones (i.e. no pre-emption of tasks). If the resource will execute the task before its deadline, the task will be scheduled to the selected resource, otherwise the algorithm will search for another resource.

In the FCFS, the tasks that did not start executing till the new set arrives will not be rescheduled. Also, the current schedule contains only the newly arrived tasks while the previous schedule has all the old tasks. Moreover, the power consumption is calculated for every schedule generated as the sum of the power consumed by all resources in that schedule when executing all the tasks scheduled to them.

To compare the quality of the solution achieved by the FCFS algorithm to that achieved by the Adaptive, we used the same problem sizes from Table 5.2. We executed the two algorithms (Adaptive GA and FCFS) for every problem size, and with the same set of data for resource specifications, tasks requirements and tasks deadline and reported the average power consumed when scheduling all tasks. As shown in Figure 5.5, the problem size (i.e. Total number of tasks and resources) is presented in the x-axis, and the power consumed when all the tasks are scheduled in the y-axis. It is clearly observed that scheduling the tasks to resources using a non-energy-aware scheduling algorithm (i.e. FCFS) consumes the maximum amount of power. However, from the graph it can be noted that the proposed Adaptive GA saves around 44% to 60% of power in comparison with the FCFS.

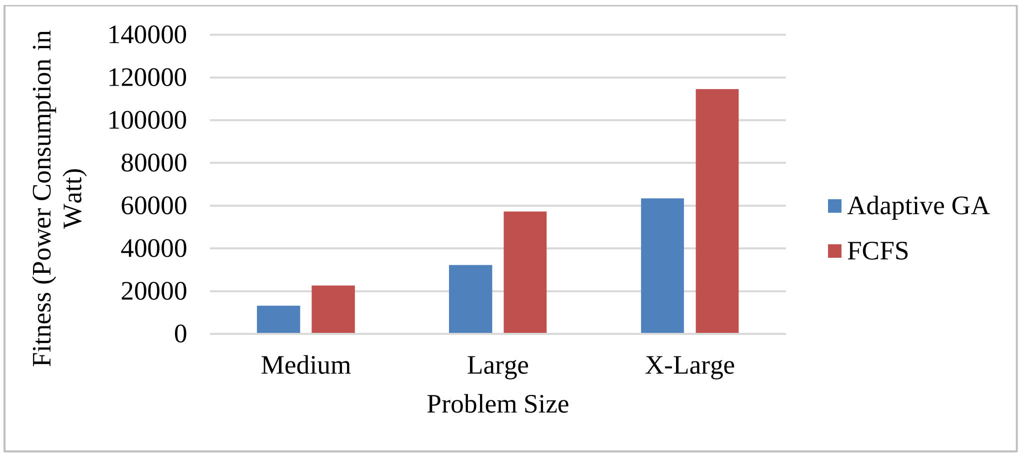


Figure 5.5: Power Consumption, Adaptive GA vs. FCFS

Figure 5.6 illustrates the total time taken by the dynamic algorithms to find a schedule for the problem sizes in Table 5.2. The x-axis presents the problem size and the y-axis presents the elapsed time in seconds. For the medium and large problem sizes, the FCFS outperforms the Adaptive GA. But, the Adaptive GA finds a solution faster than the FCFS for x-large problems. The reason behind the jump in the total time of the FCFS algorithm can be attributed to the fact that for every task to be scheduled, it first searches all the resources to find a free resource, if it did not find one, then it searches for a resource to execute the task before its deadline. Since the GA schedules all the tasks at once while the FCFS schedules tasks as they arrive in batches, the total time taken by the FCFS is only compared with the Adaptive GA as it follows the same concept.

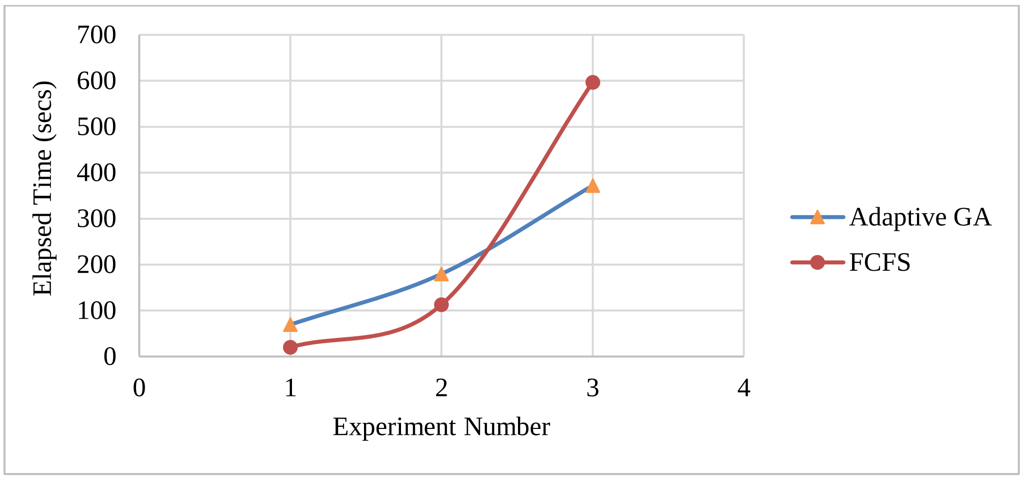


Figure 5.6: Total Elapsed Time, Adaptive GA vs. FCFS

5.4. Sensitivity Analysis

Sensitivity analysis is used to study how uncertainties and changes in input parameters affect the output of the proposed mathematical model. In our work, we deployed sensitivity analysis by systematically changing input parameters one at a time to assess their effect on the value of our objective function. The input parameters that have been changed are the number of tasks, the number of resources and the type of resources to trace their effect in the amount of power consumed.

Initially, we fixed the number of resources and increased the number of tasks gradually as shown in Table 5.3. Having the same number of resources to execute different number of tasks results in an increase in resources' usage which in turn increase power consumption. Figure 5.7 shows the increase in the power consumption per schedule (y-axis) with the increase in the number of tasks scheduled (x-axis). It also shows that the proposed Adaptive GA gives the minimum power consumption compared to the FCFS.

Table 5.3: Total Number of Tasks and Resources, Different Tasks

Num_Tasks	**Num_Resources**
1000	500
1200	500
1400	500
1600	500
2000	500

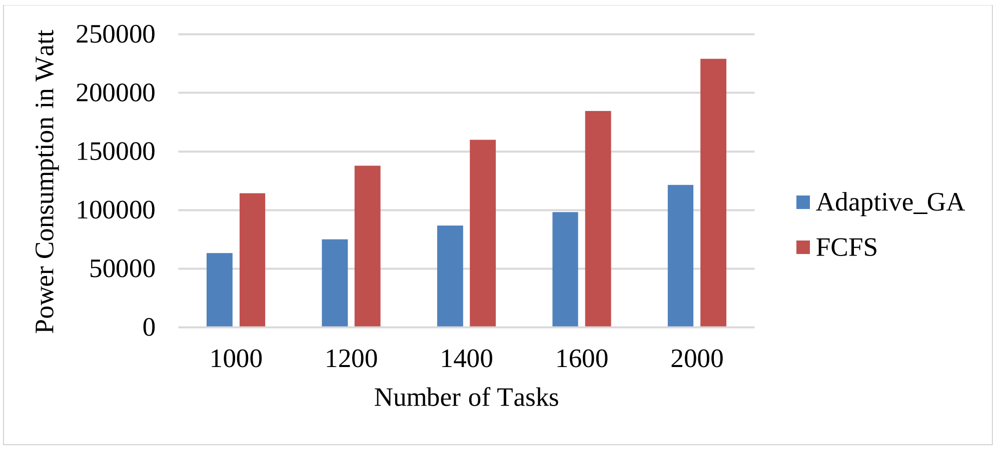


Figure 5.7: Power Consumption for Different Tasks

The total time taken by the Adaptive algorithms (i.e. GA and FCFS) to find the best schedule also increases with the increase in the number of tasks. As shown in Figure 5.8, FCFS takes more time than the Adaptive GA because it performs an extensive search first to find a free resource.

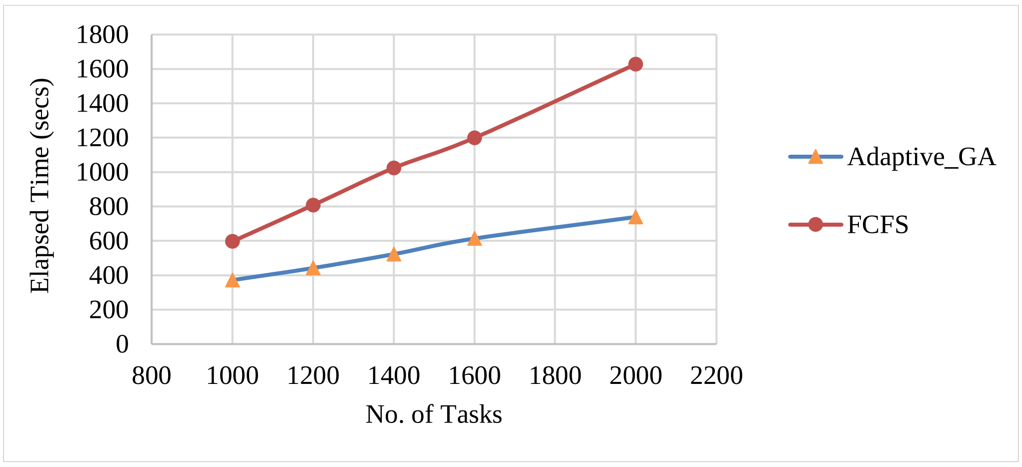


Figure 5.8: Total Scheduling Time for Different Tasks

On the contrary, increasing the number of resources for the same number of tasks has a small effect in the quality of the solution achieved. Table 5.4 shows the increase in the number of resources for the same number of tasks. As shown in Figure 5.9, the number of resources is presented in the x-axis and the power consumption over time in the y -axis. There is a slight decrease in the power consumption and the two algorithms behave in the same way for the different number of resources. The Figure shows that the proposed Adaptive GA gives the minimum power in all cases.

Table 5.4: Total Number of Tasks and Resources, Different Resources

Num_Tasks	**Num_Resources**
1000	500
1000	700
1000	900
1000	1100
1000	1500

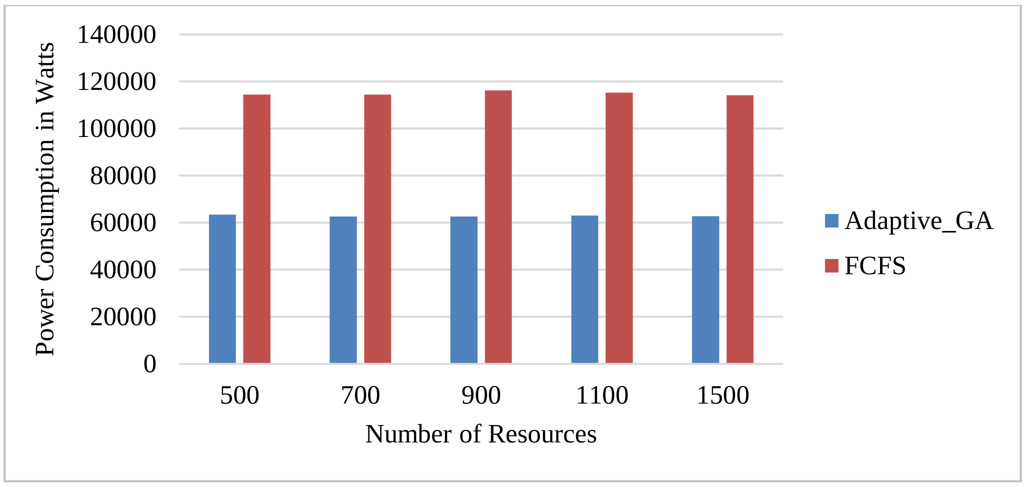


Figure 5.9: Power Consumption for Different Resources

However, the increase in the number of resources has a significant effect in the total time taken by the adaptive algorithms to schedule all the tasks. Figure 5.10 illustrates a notable drop in the total time (y-axis) of the FCFS when the number of resources (x-axis) exceeds the total number of tasks. This is because the algorithm will not take a long time to find a free resource. Yet, this is not the case for the Adaptive GA as its total time increase with the increase in the number of resources.

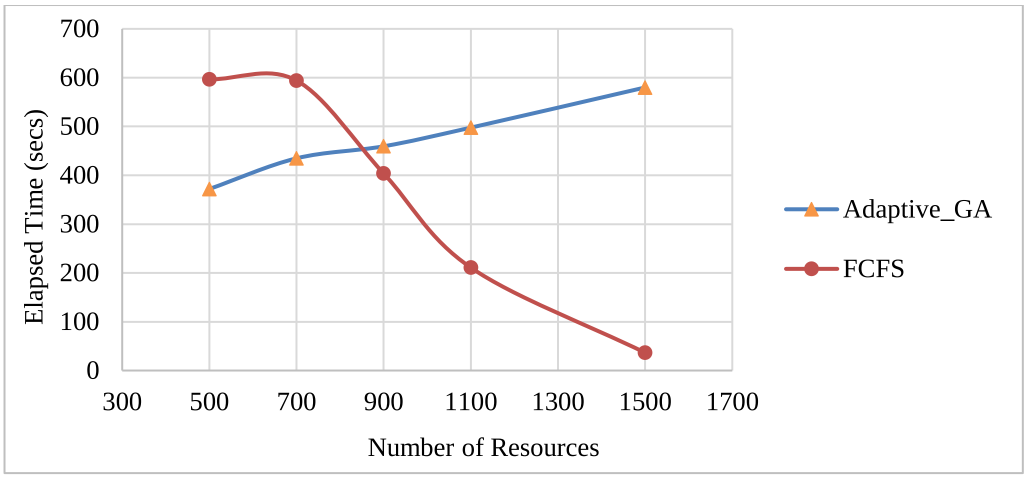


Figure 5.10: Total Scheduling Time for Different Resource

Most of the previous research [3][4][31][32][33][35][38][39][41][43] considered only computational resources when optimizing energy consumption in the Cloud environment. In our work, we conducted 2 types of experiments for the three algorithms. In the first experiment, we included all the resources (computational, storage, and network) while in the second experiment only computational servers are used. As shown in Figure 5.11, the Algorithms are presented in the x-axis and the power consumption when scheduling all the tasks in the y -axis. From this Figure, we can conclude that when considering only computational resources almost 55% of the actual power consumption is ignored.

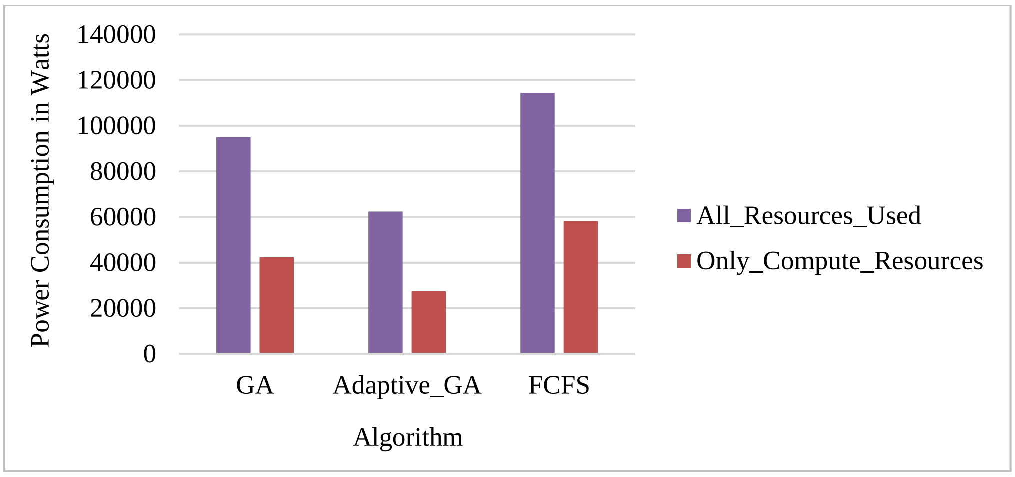


Figure 5.11: Power Consumption, All Resources vs. Only Compute Resources

Chapter 6. Conclusion and Future Work

In a Cloud environment with one datacenter and owned by a single service provider, the task scheduling problem has been addressed as an optimization problem. In this thesis, we formulated task scheduling as an Integer Linear Programming problem with the objective of minimizing energy consumption of the Cloud IaaS. The proposed model considered the computational resources as well as storage resources, network resources and cooling system as the entities that consume power within the Cloud environment. This model is subject to constraints that bound the resource provisioning process to each incoming task and schedule the tasks to resources in such a way that their deadline is met and the energy is optimized. To validate the proposed model, Lingo™ was used to give the exact solution in terms of a schedule that shows which task is executed by which resource and the minimum power consumed by this schedule. However, Lingo™ proved its inefficiency when the problem size increases (i.e. the number of tasks and resources) as it takes a very long time to provide a solution. Hence, a heuristic algorithm (Genetic Algorithm) has been used to provide a near optimal solution in less time.

The Cloud environment is dynamic in nature and the tasks and resources are not known in advance, therefore we proposed an Adaptive GA that uses dynamic scheduling with a batch mode strategy. For every received batch of tasks, the proposed algorithm finds a schedule that meets the tasks deadline and minimizes energy consumption. Tasks that did not start execution until the new set of tasks arrive are rescheduled (i.e. added to the new set of tasks) and their resources are released. When finding the new schedule, the proposed algorithm considers the previous schedule and the deadline of the new set of tasks. To test the effectiveness of the proposed Adaptive GA, a simulated IaaS Cloud environment was created using the C++ programming language. In the simulated environment, resource capabilities were extracted from previous research and real systems specifications and tasks requirements and deadline were generated randomly.

To conduct the experiments for different number of tasks and resources, we divided the problem into three categories; *Medium Size Problem* (200 Task and 120 Resource), *Large Size Problem* (500 Task and 250 Resource), and *X-Large Size Problem* (1000 Task and 500 Resource). The results obtained from the proposed algorithm were compared to the energy-aware GA (uses static scheduling) and the non-energy-aware FCFS algorithm (uses dynamic scheduling). The results were obtained as the average of three trials and they proved the efficiency of the proposed Adaptive GA as it gave the minimum power consumption for every problem size (32% to 40% from the GA and 44% to 60% from the FCFS). Furthermore, we conducted more experiments to assess the effect of changing the input parameters on the amount of energy consumed by each algorithm (Adaptive GA and FCFS). In the first experiment, we fixed the number of resources and increased the number of tasks gradually, and this resulted in an increase in the amount of power consumed by the two algorithms (Adaptive GA and FCFS) when scheduling all the tasks. In the second experiment, we fixed the number of tasks and increased the number of resources which caused a slight decrease in power consumption. In both experiments, the proposed Adaptive GA outperformed the FCFS. Moreover, we performed an experiment to justify the importance of considering all the resources (computational, storage, network resources and cooling systems) when optimizing energy consumption in the Cloud environment, and we found that 55% of the actual power consumption will be ignored if only computational resources are considered.

As a future work, the physical resources used in our proposed environment can be further explored in terms of hardware (i.e. circuits and transistors) and how their architecture affects the energy consumption. Furthermore, instead of considering only the bandwidth in the network resources, the network topology, the types of network resources and communication overhead can be considered. Moreover, tasks characteristics such as size, priority, and preemption can be integrated into the proposed model and more constraints can be added to enhance the scheduling process based on the new characteristics. Additionally, the proposed model can be extended to have a multi-objective function that addresses quality of service parameters such as the availability and reliability of services while optimizing energy consumption. All the above can be implemented on a Cloud environment that is made of one or more datacenters and owned by a single or multiple service providers.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation computer systems*, vol. 25, pp. 599-616, June 2009.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[3] N. Liu, Z. Dong, R. Rojas-Cessa, "Task Scheduling and Server Provisioning for Energy-efficient Cloud-computing data centers," in *IEEE 33rd International Conference on Distributed Computing Systems Workshops*, 2013, pp. 226-231.

[4] S. K. Garg and R. Buyya, "Green cloud computing and environmental sustainability," in *Harnessing Green IT: Principles and Practices*, 1st ed., UK: J.Wiley and Sons, 2012, pp. 315-340.

[5] A. Vosoogh and R. Nourmandi-Pour, "Scheduling Problems For Cloud Computing," *Cumhuriyet Science Journal*, vol. 36, pp. 2628-2652, May 2015.

[6] T. Mathew, K. C. Sekaran, J. Jose, "Study and analysis of various task scheduling algorithms in the cloud computing environment," in *International Conference on Advances in Computing, Communications and Informatics*, 2014, pp. 658-664.

[7] R. Aburukba, "Decentralized Resource Scheduling in Grid/Cloud Computing," Ph.D. dissertation, University of Western Ontario, Canada, 2013.

[8] S. B. Shaw and A. Singh, "A survey on scheduling and load balancing techniques in cloud computing environment," in *International Conference on Computer and Communication Technology*,2014, pp. 87-95.

[9] J. M. G. Barbosa and B. D. R. Moreira, "Dynamic job scheduling on heterogeneous clusters," in *Eighth International Symposium on Parallel and Distributed Computing*, 2009, pp. 3-10.

[10] D. Agarwal and S. Jain, "Efficient optimal algorithm of task scheduling in cloud computing environment," *International Journal of Computer Trends and Technology*, vol. 9, pp. 344-349, March 2014.

[11] T. Bawden, "Global warming: Data centres to consume three times as much energy in next decade, experts warn," Internet: http://www.independent.co.uk/environment/global-warming-data-centres-to-

consume-three-times-as-much-energy-in-next-decade-experts-warn a6830086.html, Jan. 23,2016.

[12] J. Puchinger and G. R. Raidl, "Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification," in *International Work-Conference on the Interplay Between Natural and Artificial Computation*, 2005, pp. 41-53.

[13] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, pp. 308-313, Jan. 1965.

[14] P. L. Rocha, M. G. Ravetti, G. R. Mateus, and P. M. Pardalos, "Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times," *Computers & Operations Research*, vol. 35, pp. 1250-1264, April 2008.

[15] P. Salot, "A survey of various scheduling algorithm in cloud computing environment," *International Journal of research and engineering Technology*, vol. 01, pp. 131-135, Feb. 2013.

[16] M. Dorigo, M. Birattari, and T. Stutzle, "Artificial ants as a computational intelligence technique," *IEEE Computational Intelligence Magazine*, vol. 1, pp. 28-39, 2006.

[17] A. R. Baig and M. Rashid, "Honey bee foraging algorithm for multimodal & dynamic optimization problems," in *Proc. of the 9th annual conference on Genetic and evolutionary computation*, 2007, pp. 169-169.

[18] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information processing letters*, vol. 85, pp. 317-325, March 2003.

[19] R. J. S. Raj and S. M. Prasad, "Survey on variants of heuristic algorithms for scheduling workflow of tasks," in *International Conference on Circuit, Power and Computing Technologies*,2016, pp. 1-4.

[20] K. Beghdad Bey, F. Benhammadi, F. Sebbak, and M. Mataoui, "New tasks scheduling strategy for resources allocation in Cloud Computing Environment," in *Internatioanl Conference on Modelling, Simulation, and Applied Optimization*,2015, pp. 1-5.

[21] Z. Pooranian, M. Shojafar, J. H. Abawajy, and M. Singhal, "GLOA: a new job scheduling algorithm for grid computing," *International Journal of Interactive Multimedia and Artificial Intelligence,* vol. 2, pp. 59-64, March 2013.

[22] N. J. Navimipour, "Task Scheduling in the Cloud Environments based on an Artificial Bee Colony Algorithm," in *Proc. of 2015 International Conference on Image Processing, Production and Computer Science,* 2015, pp. 38-44.

[23] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud Task Scheduling based on Ant colony optimization," in *8th International Conference on Computer Engineering & Systems,* 2013, pp. 64-69.

[24] S. Patil, R. A. Kulkarni, S. H. Patil, and N. Balaji, "Performance improvement in cloud computing through dynamic task scheduling algorithm," in *1st International Conference on Next Generation Computing Technologies*, 2015, pp. 96-100.

[25] A. V. Lakra and D. K. Yadav, "Multi-objective tasks scheduling algorithm for cloud computing throughput optimization," *Procedia Computer Science,* vol. 48, pp. 107-113, May 2015.

[26] E. P. V. Gohel, "Efficient QoS Based Tasks Scheduling using Multi-Objective Optimization for Cloud Computing," *International Journal of Innovative Research in Computer and Communication Engineering,* vol. 3, pp. 7169-7173, Aug. 2015.

[27] W. Lin, C. Liang, J. Z. Wang, and R. Buyya, "Bandwidth-aware divisible task scheduling for cloud computing," *Software: Practice and Experience,* vol. 44, pp. 163-174, Nov. 2014.

[28] S. Singh and M. Kalra, "Scheduling of Independent Tasks in Cloud Computing Using Modified Genetic Algorithm," in *Proc. of the International Conference on Computational Intelligence and Communication Networks*, 2014, pp. 565-569.

[29] P. Kumar and A. Verma, "Independent task scheduling in cloud computing by improved genetic algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 2, pp. 111-114, May 2012.

[30] Hamad, A. Safwat, and F. A. Omara, "Genetic-Based Task Scheduling Algorithm in Cloud Computing Environment," *International Journal of Advanced Computer Science & Applications,* vol. 7, pp. 550-556, 2016.

[31] Wu, Chia-Ming, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Computer Systems,* vol. 37, pp. 141-147, July 2014.

[32] R. N. Calheiros and R. Buyya, "Energy-efficient scheduling of urgent bag-of-tasks applications in clouds through DVFS," in *6th International Conference on Cloud Computing Technology and Science*, 2014, pp. 342-349.

[33] P. Kaur and M. Sachdeva, "Energy Efficient Task Scheduling in Cloud Computing," *International Journal of Computers and Distributed Systems,* vol. 4, pp. 132-137, Nov. 2016.

[34] A. S. Sofia and P. G. Kumar, "Energy Efficient Task Scheduling to Implement Green Cloud," *Asian Journal of Research in Social Sciences and Humanities,* vol. 7, pp. 443-458, Feb. 2017.

[35] G. Kaur and S. Midha, "A Preemptive Priority Based Job Scheduling Algorithm in Green Cloud Computing," in *6th International Conference on Cloud System and Big Data Engineering*, 2016, pp. 152-156.

[36] A. H. El Bakely and H. A.Hefny, "Using Shortest Job First Scheduling in GreenCloud Computing," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, pp. 348-354, Sept. 2015.

[37] A. Goyal and N. S. Chahal, "A Proposed Approach for Efficient Energy Utilization in Cloud Data Center," *International Journal of Computer Applications,* vol. 115, pp. 24-27, April 2015.

[38] G. Du, H. He, and Q. Meng, "Energy-efficient scheduling for tasks with deadline in virtualized environments, " *Mathematical Problems in Engineering,* vol. 2014, pp. 1-7, Sept. 2014.

[39] S. Hosseinzadeh and M. Hosseini, "Optimizing Energy Consumption in Clouds by using Genetic Algorithm," *Journal of Multidisciplinary Engineering Science and Technology,* vol. 2, pp. 1431-1434, June 2015.

[40] Y. Changtian and Y. Jiong, "Energy-aware genetic algorithms for task scheduling in cloud computing," in *Seventh ChinaGrid Annual Conference*, 2012, pp. 43-48.

[41] D. Kumar, B. Sahoo, B. Mondal, and T. Mandal, "A genetic algorithmic approach for energy efficient task consolidation in cloud computing," *International Journal of Computer Applications*, vol. 118, pp. 1-6, May 2015.

[42] N. Al Sallami and S. Al Aloussi, "A genetic algorithm in green cloud computing," *British Journal of Applied Science & Technology*, vol. 7, pp. 179-185, Jan. 2015.

[43] I. Kar and H. Das, "Energy Aware Task Scheduling Using Genetic Algorithm in Cloud Datacentres," *International Journal of Computer Science and Information Technology Research*, vol. 4, pp. 106-111, Mar. 2016.

[44] A. Beloglazov, "Energy-efficient Management of Virtual Machines in Data Centers for Cloud Computing," Ph.D. dissertation, University of Melbourne, Australia, 2013.

[45] D. Kumar, "Energy Efficient Resource Allocation for Cloud Computing," Ph.D. dissertation, National Institute of Technology, India, 2014.

[46] L. Minas. "The Problem of Power Consumption in Servers." Internet: https://software.intel.com/en-us/articles/the-problem-of-power-consumption-in-servers, June 14, 2014.

[47] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers," *Journal of Parallel and Distributed Computing*, vol. 71, pp. 732-749, June 2011.

[48] LINDO Systems Inc. "Powerful LINGO Solvers." Internet: http://www.lindo.com/index.php/products/lingo-and-optimization-modeling/89-products/lingo/88-powerful-lingo-solvers.

[49] N. Mansour and K. El-Fakih, "Simulated annealing and genetic algorithms for optimal regression testing," *Journal of Software Maintenance*, vol. 11, pp. 19-34, Jan. 1999.

[50] S. F. Piraghaj, "Energy-Efficient Management of Resources in Container-based Clouds," Ph.D. dissertation, University of Melbourne, Australia, 2016.

Appendix A: Experimentation Data

This section illustrates a sample of the data used in the experimentation for the tasks requirements, tasks deadline, resources capabilities, the execution time array for each task in the resources capable of executing it, and the power consumption array for each resource when executing the tasks scheduled to it.

A.1. Tasks Requirements

Table A.1 illustrates the requirements requested by 5 tasks in the form of computational, storage or network requirements. The term N/A in the table means that the task didn't request the specific capability.

Table A.1: Tasks, Requested Capabilities

Task	No. of Cores	Processor Speed (GHz)	Memory Capacity (GB)	Processor Arch. (Bits)	Storage Capacity (TB)	Storage Speed (GHz)	Storage Type	Bandwidth (Gb/s)
1	11	2	56	32	N/A	N/A	N/A	5
2	N/A	N/A	N/A	N/A	300	1	SSD	40
3	17	4	128	64	200	4	HDD	N/A
4	N/A	N/A	N/A	N/A	20	1	SSD	10
5	3	4	40	N/A	N/A	N/A	N/A	N/A

A.2. Tasks Deadline

Every task is executed within a specific time frame that is represented as the deadline. Table A.2 shows the deadline of the 5 tasks in seconds.

Table A.2: Tasks, Deadline

Task	Deadline (secs)
1	3
2	60
3	120
4	10
5	30

A.3. Resources Capabilities

The resources within the data center i.e. computational, storage, and network and every resource has different capabilities. This section demonstrates the capabilities provided by 10 resources.

A.3.1. Computational Resources

Table A.3 illustrates the capabilities provided by 5 computational resources.

Table A.3: Computational Resources, Provided Capabilities

Computational Resource	No. of Cores	Processor Speed (GHz)	Memory Capacity (GB)	Processor Arch. (Bits)
1	20	7	128	32
2	5	5	64	64
3	30	4	256	64
4	12	3	200	32
5	15	2	100	32

A.3.2. Storage Resources

From the storage resources, we considered the storage capacity, storage speed, and storage type and this is shown in Table A.4 for 3 storage resources.

Table A.4: Storage Resources, Provided Capabilities

Storage Resource	Storage Capacity (TB)	Storage Speed (GHz)	Storage Type
6	500	4	SSD
7	300	1	SSD
8	200	4	HDD

A.3.2. Network Resources

Finally, in the network resources the bandwidth is considered with the assumption that the data center network topology was set in advance. Hence, the bandwidth of 2 network resources is illustrates in Table A.5.

Table A.5: Network Resources, Provided Capabilities

Network Resource	Bandwidth (Gb/s)
9	100
10	50

A.4. Tasks Execution Time

The execution of each task depends on the requested capabilities and the capabilities provided by resources. Table A.6 shows the execution time of every task in the resources capable of executing it.

Table A.6: Execution Time of Tasks in Capable Resources

Task	Resource 1	Resource 2	Resource 3	Resource 4	Resource 5	Resource 6	Resource 7	Resource 8
1	1	N/A	N/A	2	0.5	N/A	N/A	N/A
2	N/A	N/A	N/A	N/A	N/A	10	5	N/A
3	N/A	N/A	100	N/A	N/A	N/A	N/A	50
4	N/A	N/A	N/A	N/A	N/A	5	3	N/A
5	10	25	15	N/A	N/A	N/A	N/A	N/A

A.5. Resources Power Consumption

Furthermore, the power consumed by each resource depends on the tasks assigned to them. Table A.7.a and A.7.b illustrates the power consumption per resource when executing the task.

Table A.7.a: Power Consumption of Resources

Task	Resource 1	Resource 2	Resource 3	Resource 4	Resource 5
1	20	N/A	N/A	50	10
2	N/A	N/A	N/A	N/A	N/A
3	N/A	N/A	53	N/A	N/A
4	N/A	N/A	N/A	N/A	N/A
5	23	67	90	N/A	N/A

Table A.7.b: Power Consumption of Resources

Task	Resource 6	Resource 7	Resource 8	Resource 9	Resource 10
1	N/A	N/A	N/A	30	15
2	100	5	N/A	122	200
3	N/A	N/A	20	80	170
4	5	3	N/A	6	20
5	N/A	N/A	N/A	50	190

Vita

Huda Ibrahim was born in 1989, in Khartoum, Sudan. She received her primary and secondary education in Khartoum, Sudan. She received her B.Sc. degree in Information Technology from Khartoum University in 2010. From 2011 to 2014, she worked as an IT Specialist at Ericsson Sudan and Sabal for Telecommunication.

In September 2015, she joined the Computer Engineering master's program in the American University of Sharjah as a graduate teaching assistant. During her master's study, she co-authored 1 paper which was presented in international conference. Her research interests are in scheduling, optimization and cloud computing.