OPTIMIZATION OF ENERGY CONSUMPTION IN CLOUD COMPUTING DATACENTERS

by

Ahmed Osman Osman

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

June 2018

Approval Signatures

We, the undersigned, approve the Master's Thesis of Ahmed Osman Osman

Thesis Title: Optimization of Energy Consumption in Cloud Computing Datacenters

Signature	**Date of Signature** (dd/mm/yyyy)
_____	_____
Dr. Assim Sagahyroon Professor, Department of Computer Science and Engineering Thesis Advisor	
_____	_____
Dr. Fadi Aloul Professor, Department of Computer Science and Engineering Thesis Co-Advisor	
_____	_____
Dr. Raafat Aburukba Assistant Professor, Department of Computer Science and Engineering Thesis Co-Advisor	
_____	_____
Dr. Khaled El Fakih Associate Professor, Department of Computer Science and Engineering Thesis Committee Member	
_____	_____
Dr. Shayok Mukhopadhyay Assistant Professor, Department of Electrical Engineering Thesis Committee Member	
_____	_____
Dr. Fadi 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

I would like to thank my advisors Dr. Assim Sagahyroon, Dr. Raafat Aburukba, and Dr. Fadi Aloul for providing knowledge, guidance, support, and motivation throughout my research stages. I'm deeply beholden for their great assistance, worthy discussion and suggestions.

I would like to thank the professors of the Computer Science and Engineering department who taught me the master level courses with mighty teaching methods and skills. I am really appreciating their dignified advices and motivation.

I gratefully acknoweledge the American University of Sharjah for granting me Graduate Assistantship. I greatly appreciate their financial and moral support.

Dedication

To my loved ones…

Abstract

In recent years, cloud computing has emerged as a practical paradigm for providing IT resources, infrastructure and services. This has led to the establishment of large scale datacenters that have substantial energy demands for their operation. These centers are estimated to have the fastest growing carbon foot print among all information and communication technology sector. This work investigates the optimization of the energy consumption in cloud datacenters by using energy efficient allocation of tasks to resources. The work seeks to develop formal optimization models that minimize the energy consumption of computational resources and evaluates the use of existing optimization solvers in testing these models. Energy consumption of cloud computing datacenters is mainly disbursed by the CPU, memory, disk storage, and network, with the CPU consuming the major portion. Hence, as tasks arrive for processing, these tasks must be scheduled efficiently by the cloud resource allocation mechanism. Here, the scheduling problem is modeled using the Integer Linear Programming (ILP) techniques, where models are formulated with the objective of minimizing the total power consumed by the active and idle cores of the servers' CPUs while meeting a set of constraints. Next, we use these models to carry out a detailed performance comparison between a selected set of Generic ILP and 0-1 Boolean Satisfiability based solvers in solving the ILP formulations. Simulation work is carried out using datacenters configured following industry-standard servers specifications. Results indicate that the developed models have saved up to 37.9% in energy consumption when compared to common techniques such as Round Robin. Furthermore, results also showed that from our selected set of solvers, generic ILP solvers had superior performance when compared to SAT-based ILP solvers especially as the number of tasks and resources grow in size.

Keywords*: cloud computing, energy optimization, resource allocation, scheduling, Integer Linear Programming (ILP), Boolean Satisfiability (SAT)*

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CVS	Coordinated Voltage Scaling
DVFS	Dynamic Voltage and Frequency Scaling
ILP	Integer Linear Programing
IVS	Independent Voltage Scaling
LB	Lower Bound
MIPS	Million Instructions Per Second
NIST	National Institute of Standards and Technology
PB	Pseudo Boolean
POD	Predict Optimize Dispatch
QoS	Quality of Service
SAT	Boolean Satisfiability Problem
SLA	Service Level Agreement
UB	Upper Bound
VM	Virtual Machine
VOVO	Vary-on Vary-off

Chapter 1. Introduction

In recent years, cloud computing has emerged as a practical paradigm for hosting and delivering services over the internet. This in turn has led to the creation of huge datacenters which are power hungry entities with complex requirements and operational needs. A critical design parameter for datacenters is their power consumption characteristics and energy demands. This chapter provides a brief overview of cloud computing, and power consumption as related to the cloud computing environment. It also introduces resources allocation and scheduling as critical elements of significance within the context of a datacenter operation and the reduction of its electricity consumption.

1.1. Overview of Cloud Computing

Several definitions of cloud computing exist in the literature, each focusing on different characteristics of cloud computing. One of these definitions was introduced by Buyya et. al [1] that states: “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”. From the definition, multiple characteristics of the cloud can be identified:

- The abstraction of physical resources through virtualization which enables the creation and provisioning of multiple resources from a single physical resource to maximize the resources utilization.
- Dynamic provisioning of resources to consumers to meet their tasks’ requirement and execute services without the need for human interaction.
- The different physical resources are pooled and presented to the consumers through the use of virtualization.
- Cloud providers and consumers agree on the Quality of Service (QoS), redundancy, time constraints, and other attributes through the Service Level Agreement (SLA).

The National Institute of Standards and Technology (NIST) defined cloud computing as: “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” [2]. The essential characteristics of cloud computing according to NIST are:

- On-demand Self-service: consumers can provision and use cloud resources and services without the need for human interactions with cloud providers.
- Broad Network Access: cloud services are provided over the internet and can be accessed through various devices as long as they have network access.
- Resource Pooling: the cloud provider’s resources are pooled together to serve different consumers using multitenancy. Multitenancy is a service model, where multiple users share a single instance of the resource.
- Rapid Elasticity: the ability to dynamically allocate and deallocate resources based on the users demands.
- Measured Service: cloud services are measured and controlled by the cloud middleware to allow for charging the cloud consumers or optimizing the use of resources.

The different service models in cloud computing are as follows [2]:

- Software as a Service (SaaS): the service provided to the consumer is to access and use an application running on the cloud. Consumers do not control the underlying cloud infrastructure.
- Platform as a Service (PaaS): the service provided to the consumer is to develop and deploy applications on the cloud. The runtime environment is provided by the cloud platform. The user cannot manage or control the cloud infrastructure but has total control over the application and some configuration to the runtime environment.

- Infrastructure as a Service (IaaS): the service provided to the consumer is to provision different computing resources (processing, storage, or network resources). The consumer can deploy and run operating systems or applications using these resources.

Cloud infrastructure can be provisioned to various types of individuals and organizations creating different types of deployment models as follows [2]: Private clouds where exclusive physical resources are provided to a single organization, Community cloud where the cloud resources are provisioned exclusively to a community of similar-interest, Public cloud which can be used by everyone, and Hybrid cloud which consist of two or more of the previously mentioned cloud deployment models.

It can be noticed that both Buyya [1] and NIST [2] definitions agree on some of the cloud characteristics. Both definitions express that consumers can dynamically provision resources without the need of human intervention, and that resources are pooled together and provided to the consumers as per their request. The NIST definition is more comprehensive as it emphasizes more on the different types of service and deployment models.

1.2. Energy Consumption in Cloud Computing Datacenters

Cloud computing services are provided through datacenters. Datacenters contain large scale, networked compute infrastructure. The major physical resources in these centers can be classified into three categories: compute, storage, and network resources. Compute resources are pooled and perform the required processing as per the consumer requests. Storage resources are typically large storage units (Hard disks) that are used to store the provider's and users' data. Network resources consist of various network elements (Routers, Switches) to connect the physical resources together.

For the cloud providers to be able to meet the consumers' requests with high Quality of Service (QoS), they need to have large numbers of these resources with high capabilities. Hence, datacenters hosting these resources end up consuming significant amounts of energy. In 2010 for example, the electricity usage of

datacenters around the world represented $(1.1\% - 1.5\%)$ of the total worldwide electricity usage while in the United States, the datacenters usage represented up to 2.2% of the total energy consumed in the US [3]. This consumption problem has been exacerbated further by the use of blade servers that occupy less space but consume more power than traditional rack servers. All of these factors stress the need for designing energy efficient datacenters to minimize the energy costs and sustain the environment.

Some of the techniques that can contribute to energy efficiency in cloud computing include:

- Use of smart tasks allocation strategies including static as well as dynamic approaches to reduce energy consumption [4, 5].
- Use of virtualization techniques to enable the creation of multiple virtual resources on a single physical resource. This allows the reduction of the number of physical resources needed to perform a task and hence can contribute in reducing the energy consumption [6].
- Workload Consolidation to minimize the number of resources used and maximizes the utilization of the resources [6].
- Other techniques include Dynamic VMs Placement and VM Migration, Heat Management and Temperature-aware Allocation, and Load Balancing and Task Scheduling [7].

The work discussed here focuses on identifying efficient task allocation and scheduling techniques as a mean to minimize the energy consumption in cloud computing datacenters.

1.3. Scheduling in Cloud Computing

In essence Resources allocation and Scheduling are about making decisions to allocate resources to tasks over time with the goal of optimizing one or more objectives [8]. The resources and tasks differ depending on the environment in which the scheduling process is taking place. The tasks allocation and scheduling problems can be mapped to an optimization problem by building a model that represents all the

parameters affecting the problem and then solve that model to minimize or maximize some of these parameters. Solving the model will produce the optimal assignment of tasks to resources at a specific time which is the required solution of the scheduling problem. Methods used in solving optimization models can be categorized into two classes: Exact and Heuristic methods.

In the context of cloud computing, scheduling is the problem of allocating compute, storage, or network resources to tasks over a period of time, whether these resources are physical or virtual resources. Some of the related scheduling objectives can be minimizing the completion time, maximizing the profit, minimizing the energy consumption, maximize the resource utilization, or it can have multiple objectives.

In recent years, the scheduling aspects of a datacenter have been tackled with different optimization objectives in mind. For example, in [9], the tasks or processes are assigned in a round robin fashion between the available processors. The tasks are distributed equally between all the processors; problem is, some processors will end up being heavily loaded compared to others. The authors proposed overcoming this limitation by using a weighted round robin approach. The Min-Min static load balancing algorithm is used in [10]. Here, parameters related to the job or task are assumed to be known in advance. The algorithm starts by identifying the minimum completion time needed by a task and assigning tasks by the cloud manager on ascending completion time needs. A drawback is the fact that tasks requiring long execution times have to wait and this may lead to starvation.

In [11], a task scheduling algorithm was proposed with the objective of maximizing the profit by minimizing the makespan and maximizing resources utilization. The cloud computing environment was represented by a set of heterogeneous clusters containing different types and number of servers. The requests submitted by clients were assigned to one cluster only. The proposed solution consisted of two steps: finding an initial solution that shows the fastest two clusters and assign jobs to them using the Min-Min algorithm, then within each cluster tasks were assigned to the individual servers. The authors assumed the tasks were independent and that the cluster can execute more than one job, but each client is assigned to a single cluster. The proposed method was tested against the Min-Min algorithm in terms of the makespan values and it was shown that the proposed method

performed better. The work discussed in [12] proposed an optimization algorithm called the Group Leaders Optimization Algorithm (GLOA) to solve the scheduling problem in an optimum time. The objective of the work is to minimize the overall make span of the independent tasks. The method starts by randomly creating an initial population that represents different mappings of tasks to resources. This initial population is divided into several groups and assigned a fitness value for each member of each group. The member with the best fitness in each group is selected as the leader of the group. The leader of the group is mutated with a random member to create new members that are similar to the leader. Another mutation happens with members of different groups to create diversity and escape from local minima. These processes are repeated for a preset number of iterations and the group leader with the best fitness at the end is selected as a solution to the problem. The proposed method was compared to other scheduling algorithms including GA (genetic algorithm) and SA (simulated annealing) and combinations of them and it was found that it produces the minimum makespans. The authors of [13] proposed a Generalized Priority algorithm for scheduling tasks efficiently in cloud computing environment. The tasks were prioritized according to their size with the highest priority given to the task with the large size. The resources which are the VMs were also prioritized but according to their task executing speed expressed in Million Instructions Per Second (MIPS). The highest priority is given to the VM with the highest MIPS. A datacenter broker assigned the tasks with the highest size to the available VM with the highest MIPS. The proposed algorithm was found to be more efficient than FCFS (First Come First Serve) and RR (Round Robin) algorithms in terms of execution time. In [14], a multi-objectives task scheduling algorithm was introduced. The proposed algorithm map tasks to VMs in order to increase the datacenter throughput and minimize the cost of an application in SaaS cloud environment. The VMs had two attributes, ID and speed of the VM in terms of MIPS. The tasks had three attributes: ID, quality of the task as QoS value, and size of the task in Million Instructions (MI). A cloud broker took care of assigning the tasks to VMs. Each task was given a priority that is reversely proportional to its QoS value. The cloud broker maintained a list of all the VMs in the datacenter in descending order according to their MIPS. The tasks are then assigned to the VMs in this list. The highest priority task (lowest QoS value) is assigned to the first VM in the list (highest MIPS) and the second highest priority task is assigned to

the second VM on the list and so on. After reaching the last VM on the list, the next task is assigned to the first VM. The proposed algorithm was shown to perform better against FCFS and Priority scheduling algorithms in terms of both the execution time, and the system throughput. The aforementioned work highlights the importance of scheduling algorithms and polices in contributing to the efficient and cost-effective operations of datacenters.

1.4. Research Problem and Contribution

In recent years, energy efficiency of datacenters has attained key importance due to its economic, environmental and performance impacts. Datacenter spaces may consume up to 100 to 200 times as much electricity as a standard office space, furthermore the energy costs of powering a typical datacenter doubles every 5 years [15, 16]. This has brought the problem of energy optimization and power consumption reduction in datacenter to the forefront of current research activities that are related to design and implementation of these centers. In the context of cloud environment and datacenters, scheduling encompasses the following [17]:

1- Resources: physical/virtual devices with the ability to execute or process tasks;
2- Tasks: instructions to be executed by the resources.
3- Constraints: conditions must be considered when scheduling the tasks to the resources. They may be task-based, resource-based, or a combination of these. They could also be hard constraints, meaning constraints that must be full-filled or soft constraints that can be relaxed.
4- Objectives: the evaluation criteria that needs to be measured in order to assess the system performance.

Our research aims to formulate the scheduling problem (with the objective being the minimization of power consumption in the datacenter) as a 0-1 ILP model. The generated ILP model will be transformed into a Boolean Satisfiability Problem (SAT) that can be solved using SAT solvers with the results being mapped back to the original scheduling problem. We will validate the ILP models and examine the capabilities of current state of the art Generic ILP and SAT solvers when used to solve formulations of this scheduling problem. Scheduling is the allocation of tasks to

capable resources over time with the goal of optimizing one or more objectives that is subject to specific tasks and resources based constraints. In the proposed approach, the time is considered as a discrete element where tasks may arrive at different time windows. The tasks are scheduled in batches based on their arrivals. All scheduled tasks in a batch are to start at the same time and may have different execution time that depends on its size and the assigned compute capabilities.

SAT solvers have traditionally been used to solve decision problems. Recently, SAT solvers have also been extended to solve optimization problems, specifically 0-1 ILP problems. Hence, SAT solvers can now handle both decision and optimization problems. Recent studies have shown that SAT solvers can compete with the best available generic ILP solvers in solving 0–1 ILP problems. Nevertheless, the SAT problem is an NP-complete problem and many problems remain difficult to solve, regardless of their size. We have seen SAT algorithms go through tremendous improvements in the last few years [18], allowing larger problem instances to be solved in different applications domain. Different solvers employ various powerful algorithms that are sufficiently efficient to deal with large-scale SAT problems that typically arise in the various domains. Most of these algorithms claim competitive results in runtime efficiency and robustness. However, due to the 'structure' of instances in different domains, it is difficult to identify one best SAT solver for all instances, as some algorithms could show better runtimes with specific instance structures. Therefore, it is recommended that various competitive SAT solvers be evaluated on a given SAT instance. It is likely that the best identified SAT solver will also have similar performance on other problems from the same domain (or i.e. with similar structure).

1.5. Thesis Organization

The rest of this document is organized as follows: Chapter 2 reviews the literature published in the area of power consumption modeling, task scheduling for optimizing energy consumption, and optimization tools. Chapter 3 presents the environment setting and the ILP model formulation. Chapter 4 introduces the steps followed to validate the models and the different test scenarios that are carried out. In Chapter 5 we present and discuss the results of the different tests. Finally, in Chapter

6 we conclude our research and describe some of the areas that can be explored in future work.

Chapter 2. Background and Literature Review

In this chapter, we discuss energy consumption in datacentres and introduce some of the power modelling levels and techniques proposed by researchers in the field. Furthermore, we review some of the allocation and scheduling techniques that are developed with the intent of optimizing energy. ILP modelling used in the context of energy minimization and use of SAT Solvers are discussed. Furthermore, some of proven optimization tools are briefly described.

2.1. Power Modeling Levels

Modeling of power in datacenters can be approached from different levels. A detailed review of the viable modeling techniques is beyond the scope of this work. However, readers are referred to [3] for a comprehensive review. Some of the level at which power may be modeled include:

- Digital circuit level
- Single server level
- Multiple servers level (cluster of servers)
- Datacenter level
- Multiple Datacenters level

In [19] it is proven that the dynamic power consumption of a CMOS-based microprocessor is proportional to the switching capacitance, the frequency, and the squared voltage. This relation can be represented by the following equation:

$$P = ACV^2f \tag{2.1}$$

where P is the power consumed by the processor, A is the switching factor, C is the physical capacitance, V is the supply voltage, and f is the clock frequency. This model although simple, but it forms the basic principle behind the Dynamic Voltage and Frequency Scaling (DVFS) power reduction techniques.

The Authors of [20] proposed a simple model for the power consumed at the server level. The model can be expressed as:

$$E(A) = E_{CPU}(A) + E_{MEMORY}(A) \quad (2.2)$$

where E is the total energy consumed by the server when executing algorithm A, E_{CPU} is the energy consumed by the CPU, and E_{MEMORY} is the energy consumed by the memory. Their model ignored the energy caused by the I/O operations because it was assumed to be the responsibility of the separate network attached storage.

A more comprehensive model was introduced by Lewis et. al in [21]. The authors developed a linear regression model to predict the power consumption on a single server blade. The model can be expressed as:

$$E_{system} = \alpha_0(E_{proc} + E_{mem}) + \alpha_1 E_{em} + \alpha_2 E_{board} + \alpha_3 E_{hdd} \quad (2.3)$$

where:

- E_{proc}: Energy consumed in the processor.
- E_{mem}: Energy consumed in the DDR & SDRAM chips.
- E_{em}: Energy consumed by the electromechanical component.
- E_{board}: Energy consumed by peripherals of the board.
- E_{hdd}: Energy consumed by the hard disk drive.

The constants α_0, α_1, α_2, and α_3 depends on the specific server architecture.

In [22], Fan et. al proposed a power model for servers based on the utilization level of the server. The model represented the relation between the utilization of the server and its power consumption as a linear relation, as in equation (2.4).

$$P_u = P_{idle} + (P_{busy} - P_{idle})u \quad (2.4)$$

Where P_u is the total server power, P_{busy} is the average power consumption of the server when fully utilized, P_{idle} is the average power when the server is idle or at 0% utilization, and u is the utilization factor.

The authors then proposed a nonlinear model for the server power based on their experimentation as follows:

$$P_u = P_{idle} + (P_{busy} - P_{idle})(2u - u^r) \quad (2.5)$$

where r is a calibration parameter to reduce the squared error and it is found by experimentation. The two models are tested to compare their accuracy and it was shown that the non-linear model has less than 1% error while the linear model has less than 5% error in predicting the server power.

Another approach to modeling the power was proposed by Li et. al in [23] which was based on modeling the power of Virtual Machines (VM). This model breaks down the power of the server to two parts as in equation (2.6): a constant baseline power that is determined empirically for each server, and the sum of the power consumed by all the VMs running on the server,

$$P_{server} = P_{baseline} + \sum_{i=1}^{n} P_{vm}(i) \quad (2.6)$$

where P_{server} is the total power of the server, $P_{baseline}$ is the constant power decided by experiment, P_{vm} is the power of the active virtual machine, and n is the total number of VMs that are active in the server.

2.2. Optimizing Energy Consumption in Cloud Computing

Optimizing the power consumption in a cloud computing environment can be achieved by considering it as a scheduling problem where the tasks submitted to the datacentre are scheduled to the different resources in the datacentre in such a manner to optimize the power consumed by computing resources. However, the scheduling problem can have multiple optimizations objectives such as increasing the utilization of resources, reducing the time taken to perform the tasks, maximizing the profit, or minimizing the energy consumption.

Liu et. al [24] proposed an optimization model for minimizing the energy consumption of cloud computing datacentres using task scheduling. The authors formulated an integer programming model of the task scheduling problem with the goal of minimizing the number of active servers and thus minimizing the overall

energy consumption of the datacentre without breaking the response time constraints. The formulation process considered two cases: an initial case where no task is received before and the new tasks are assigned immediately and a second case where the datacentre has backlogged tasks that result in queuing delays. In addition, the authors proposed a scheduling scheme named most-efficient-server first that assigns the most efficient server first. The model and the environment are then simulated with heterogeneous tasks. Their results showed that the proposed scheme minimizes the server energy consumption 70 times better than a random-based scheduling scheme on average.

In [25] a scheduling algorithm for online scheduling of servers of cloud computing providers was proposed. The algorithm is called Predict Optimize Dispatch (POD) based on the three main steps that the algorithm follows. It was assumed that the servers used have multiple processors with different speeds and different power consumption, and each server can be in busy, idle, or switched off mode. The objective of the scheduling algorithm is to minimize the energy consumption of a cloud computing environment. The algorithm follows three steps: first it predicts the workload assigned to the datacentre. The workload is anticipated from one of two resources: based on the cloud provider's information or using a new kind of multi-arm bandit workload prediction. Then, the algorithm modifies and optimizes the set of available servers to best fit the predicted workload. The final step is to schedule the jobs to the available processors using a modified version of Round Robin scheduling method. The authors then simulated their proposed solution and tested it against other online and offline scheduling algorithms and presented their results with a recommendation on which type of scheduling to use for different types of workload.

Another technique that is used with scheduling to minimize the energy consumption in cloud computing environment is Dynamic Voltage and Frequency Scaling (DVFS). In [26] a green energy-efficient scheduling algorithm for cloud computing datacentres using DVFS technique was proposed. The algorithm aims to increase the utilization factor of the different servers in the datacentre to reduce its energy consumption. The authors assumed heterogeneous servers with each server having different processor, RAM, and hard disk. Each server has a defined minimum

and maximum frequency and this frequency is reduced to minimize the energy while keeping in mind the requirements of the workload. VMs are created from the servers and the jobs are assigned to these VMs using priority job scheduling. The experimental results showed that the proposed algorithm reduce the energy consumption by 5 to 25% while maintaining the required performance and execution time.

Various combination of DVFS and node vary-on vary-off (VOVO) methods were evaluated in [27]. The objective of the research was to minimize the power consumption in clusters of servers during times where the workload is reduced. The authors explored five different policies with different degrees of complexity at the implementation level. The five policies are:

- Independent Voltage Scaling (IVS): uses processors that have voltage scaling property and can independently change their voltage and frequency according to their workload.
- Coordinated Voltage Scaling (CVS): exploits voltage scaled processors but coordinate their voltage scaling algorithms.
- Vary-on Vary-off (VOVO): turning the whole server on or off depending on the size of the workload.
- Combination of IVS and VOVO.
- Combination of CVS and VOVO.

These different policies were studied, and it was found that the simplest policy (IVS) saves between 20 to 29% of the energy consumed by the cluster. While VOVO policy energy savings are in the range of 22 to 42%. The policy that produced the most energy savings was combining VOVO with CVS which saved 18% more energy than normal VOVO but with a drawback of increased complexity when it came to the implementation part.

Other from scheduling, VM placement is also used to optimize the energy consumption in cloud computing datacentres. VM placement is the allocation of virtual machines to the available physical machines. This virtualization is one of the characteristics of cloud computing that makes it more green than normal computing.

Xin Ye et. al. [28] proposed a multi-objective VM placement model for cloud computing. The authors took into consideration the different requirements of both the cloud providers and users to build the model with the objectives of minimizing the energy consumption, maximizing load balance, maximizing resource utilization, and maximizing robustness. The proposed model is solved using an improved Energy-Efficient KnEA (Knee Point-Driven Evolutionary Algorithm) algorithm. The model is then tested to verify its consistency and to evaluate the performance of the improved algorithm. The results of their work showed that the proposed model outperforms the compared against algorithms in terms of energy saving, load balancing, and robustness.

Typically, with VM placement algorithms the objective is to minimize the number of the physical machines used to minimize the overall energy consumption of the datacentre. In [29] Hui Tian et. al. proposed another VM placement model with the objective of minimizing the total execution time of the physical machines. Their approach is based on the concept that the energy cost of the physical machines depends on both the number of the machines and the total execution time of the machines. The authors developed optimization models for both offline and online VM placement. The proposed online algorithm is then tested against Best Fit and First Fit algorithms while the proposed offline algorithm is compared against Best Fit Decreasing and First Fit Decreasing algorithms. In both cases the proposed algorithms performed better than the other algorithms. An additional test was carried out to compare the performance of the offline and online algorithms against each other. The online algorithm is shown to use more physical machines than the offline algorithm and with longer execution times.

2.3. ILP Optimization and SAT solvers

Linear Programming is a mathematical method for maximizing or minimizing a linear objective function under certain conditions or constraints consisting of different variables. Integer Linear Programming (ILP) is a special type of linear programming where some or all the variables in the objective function and constraints are limited to integer values. ILP problems can be categorized into two categories: 0-1 ILP problems where the variables in the problem are binary variables and have values of 0 or 1, and Generic ILP where the variables are not limited to binary variables [30].

In this work, the generated ILP model will be transformed into a Boolean Satisfiability Problem (SAT) equations. These equations will be solved using SAT solvers and the result will be mapped back to the original scheduling problem. Boolean Satisfiability Problem refers to finding a satisfying assignment for a problem or proving that none exist. A satisfying assignment for a certain Boolean formula is an assignment of the binary variables in the formula to 0 or 1 to make the overall formula evaluate to 1. The assignment must satisfy all the constraints of the problem simultaneously [31]. Both the SAT problem and the 0-1 ILP problem are represented using binary variables but in SAT, the Boolean formulas used to represent the problem and its constraints must all be in a product-of-sums form or a Conjunctive Normal Form (CNF) and the operators used in these forms are logical operators (AND, OR, and NOT) while in 0-1 ILP problem the constraints are simple inequalities of the form $Ax \leq b$ where $b \in \mathbb{Z}^n$, $A \in \mathbb{Z}^m \times \mathbb{Z}^n$, and $x \in \{0,1\}^n$. These constraints are referred to as Pseudo Boolean (PB) constraints and they are considered as a general form of the CNF constraints [32].

The SAT problem is considered to be NP complete problem. The solving time increases exponentially with the increase in the number of variables. The SAT solvers are algorithms that determine if a SAT problem is satisfiable or not and find a satisfying assignment in case the problem was satisfiable. The basic approach to solving a SAT problem is by going through the search space of the problem by assigning values to each variable and check whether this assignment satisfy all the constraints in the problem or not. Figure 2.1 illustrates the search space of a Boolean formula written in CNF format. The formula contains four different binary variables which results in 16 possible assignments, or in general if the formula has n variables then 2^n possible assignments exist. To satisfy the formula written above the tree as shown in Figure 2.1, the solver propagates through the search space using intelligent search techniques (such as DPLL procedure [33]) until it reaches a satisfying assignment. Depending on the solver's configuration, it can either stop after finding the first satisfying assignment or continue with finding all the satisfying assignments.

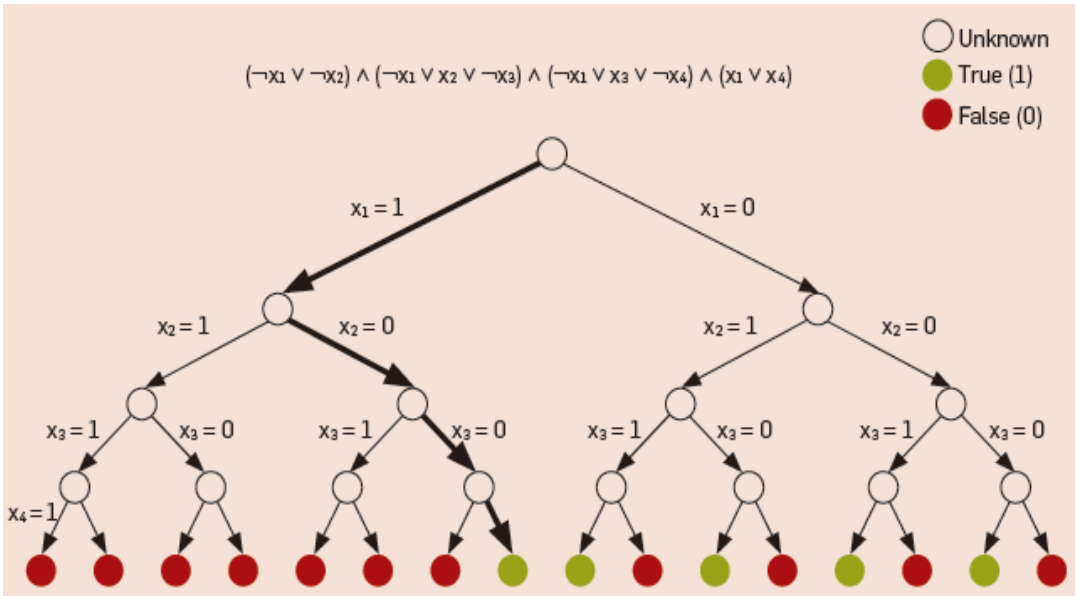

Figure 2.1: The search space of a sample formula [31]

In the past few years the use of SAT models to represent different problems in various domains increased rapidly due to the huge improvement in the SAT solving algorithms and the availability of better and more powerful computing capabilities. One of the great improvements to SAT-based solvers is the extension to handle PB constraints because they are easier to write and understand and can replace an exponential number of CNF constraints. Before this extension SAT models were limited to decision problems but with the help of PB constraints SAT can now model optimization problems and solve them using the different SAT-based PB solvers [34].

2.4. Optimization Tools

Commercialized and open source optimization tools with various embedded algorithms exist today. Those tools deal with generating a solution to a defined linear, non-linear, or integer models. Some of the popular tools include:

- CPLEX [35]: is a high-performance solver for linear programing, mixed integer programing, and quadratic programing. It is a well-known and wildly used commercial solver developed over 20 years ago and is currently owned by IBM. The solver uses the Simplex exact algorithm, and the name CPLEX is actually taken from C-Simplex which is Simplex algorithm implemented in C language. The solver is

improved and updated periodically to increase its performance and include different algorithms for different types of problems. some of the various methods used in CPLEX are: primal simplex algorithm, dual simplex algorithm, network simplex algorithm, and the barrier method. This solver will be used in testing the proposed model.

- LINGO [36]: is an efficient comprehensive tool used in building and solving different types of optimization models. It was developed by LINDO Systems as an optimization modeling software for linear, nonlinear, and integer programming. LINGO provides its users with a language to express the optimization models, an environment to build and edit their problem, and a set of solvers. These solvers include but not limited to: Primal and Dual Simplex solvers, Barrier solver, Integer solver, General Nonlinear solver, and Quadratic solver.
- Pseudo Boolean SAT Solvers: which are basically SAT solvers that can handle pseudo Boolean constraints allowing the SAT solvers to be used in solving optimization problems. In [34], for example, two power estimation problems were formulated as SAT problems and evaluated using generic ILP and SAT-based solvers. It was shown that the SAT based solver performed better than random based approaches in terms of finding the optimal values of the proposed problem.

In this work PB-SAT solvers are used to solve the proposed optimization model. The testing process involves using different solvers and comparing their results. The solvers are selected according to the following criteria:

- The proven performance of the solver from the reported literature.
- The ability of the solver to handle large problem size.

Based on the above criteria, we identified three PB-SAT solvers, namely:

- NaPS [37]: Nagoya Pseudo Boolean Solver, is a solver for PB constraints that are linear and contain binary variables only. It was the best performing solver in the Pseudo Boolean competition 2016 in both categories of optimization with small integers and large integers and was among the best solvers in the

other categories. The information and results of the competition is available in [38].

- Minisat+ [39]: is a modified version of the Minisat solver to be able to deal with pseudo Boolean constraints, and despite being a simple solver it performed well in the pseudo Boolean competition through the years.
- Sat4j [40]: is a Boolean reasoning java library that is used to solve Boolean satisfaction and optimization problems. The sat4j solver is different from the other two solvers because it does not prioritize being the fastest in solving a problem. However, the solver is full featured, robust, and user friendly. The solver design follows the Java design guidelines and code conventions.

In addition, the performance of a generic ILP solver, named CPLEX [35] is also studied and compared with the other solvers.

Chapter 3. Methodology

This chapter analyses the nature and the characteristics of the cloud computing datacentre and models the scheduling problem. The scheduling problem in the cloud computing allocates requests/tasks to the capable resources at specific time while minimizing the energy consumption of the datacentre and achieving the requirements related to the tasks. This chapter presents the characteristics of the cloud computing environment, analyses the scheduling problem in the cloud datacenter, and formulates the scheduling problem as an Integer Linear Program (ILP) model.

3.1. Cloud Computing Environment

A cloud computing environment may consist of a single or multiple datacentres that may belong to a single or multiple providers. This work deals with a single cloud datacentre that belongs to a single provider. The cloud datacentre infrastructure contains three resource types: compute, storage, and network resources. The computing resources are connected together in a network. Servers are the main computing units in the datacentre. At this level, the attributes that can affect the power consumption are the type of these servers (tower, rack, or blade) and the architecture of the server which depends on the manufacturer (AMD, XEON, or INTEL, for example). Each server consists of: CPU, Memory, Network Interface Card (NIC), Hard disk (HD), with the rest of the peripherals assumed to be mounted on server's mainboard. A CPU can have different number of cores, each of which has a maximum frequency and voltage that they run at when fully utilized. Each CPU has a processor architecture and an operating system. For memory, typically, the attributes affecting its power consumption include: size, speed, type, and architecture. The characteristics of NIC that are related to power consumption include the Ethernet speed supported (in Mbps), data bus type, intelligent interrupt management, and the architecture of the specific NIC type [41]. In Storage resources, the different components include: storage units of different types (HDs, SSDs), Hardware RAID controller (to support different levels of RAID), and Power Supply Units. For storage units, the architecture of the unit, as well as, the maximum rates of read and write operations are the most relevant attributes for the power consumption of the unit. For HDs, we have attributes such as the speed in RPM and number of platters [41].

In this work, we focus on the compute resources since they consume about 26% of the total energy of the datacentre [27]. The main computing units are the servers. It is reported in [22] that for a typical server, the CPU consumes 38% of the total server power, Memory consumes 17%, Hard disk consumes 5%, and the remaining peripherals (including the on-board Fan) consume 40%. The memory of a server is assumed to consume power if the server is switched on and the CPU(s) inside that server are active and accessing it, hence, the power consumed by memory can be considered as a part of the power consumed by the CPU(s) of the server.

3.1.1. Datacentre characteristics. The datacenter is assumed to contain *I* servers, each server has *J* CPUs. The CPUs in each server can have multiple cores. The following assumptions are considered in modeling the problem:

- Servers may contain different number of CPUs.
- The CPUs of a server are either dual-core or quad-core.
- All cores in the same server have a certain amount of RAM in GB.
- All the cores inside the same server are assumed to run at a specific frequency in GHz.

3.1.2. Task characteristics. Typically, incoming tasks to the datacenter have the following characteristics:

- Type of the task: computing, storage, or networking task.
- Size of the task: how big the task is in number of instructions.
- Deadline of the task: the task execution should end before this time. Calculated using the task size and core frequency.
- Required Frequency: minimum core frequency in GHz needed to execute the task.
- Required RAM: minimum core RAM size needed to execute the task in GB.

3.1.3. Cloud computing scheduling problem. tasks are allocated to the available resources at a specific time in the datacentre with the objective of

minimizing the number of active servers, and consequently any idle servers are switched off to minimize the overall energy consumption. The following assumptions are made while developing the scheduling model:

- Each CPU can run two or four tasks at the same time depending on number of cores it contains; each core will run only one task at 100% utilization.
- A server will be switched ON if at least a single core in one of its CPUs is executing a task and will be switched OFF otherwise.
- A core can be ON (switched on and executing a task), idle (switched on but not running any task), or completely OFF (switched off).
- The tasks will be allocated in batches that arrive at distinct times (t_0, t_1, …, t_n).
- All the tasks in a specific batch will be scheduled immediately and will start running simultaneously (no task will be assigned to start later).
- A task can only be executed by a core that meets the frequency and RAM requirements of the task.
- For any task there is at least one core that meets the requirements of the task (no task will be rejected).
- The execution of a task on a certain core is assumed to consume a specific amount of energy that will be called the energy cost in this model.

The energy cost of executing a task in a core depends on the characteristics of both the task and the core as will be explained later in this chapter.

3.2. Optimization Model Formulation

This section introduces the Integer Linear Programming (ILP) model that minimizes the energy consumption of the servers in a cloud datacentre. Table 3.1 presents the variables used in developing the model of the problem and their meaning.

The intended objective is to find the specific assignment of tasks to cores that gives the minimum energy value. The total energy value is the power consumed by the cores when a task is allocated to it plus the power consumed by the idle cores.

Table 3.1: Variables and their definition

Variables	**Description**
S_i	Binary variable, equal to 1 if server *i* is on, and 0 otherwise.
$P_{i,j}$	Binary variable, equal to 1 if CPU *j* in server *i* is active and 0 otherwise.
$G_{i,j}$	Binary variable, equal to 1 if CPU *j* in server *i* is idle and 0 otherwise.
$C_{i,j,k}$	Binary variable, equal to 1 if core *k* in CPU *j* in server *i* is active, and 0 otherwise.
$Q_{i,j,k}$	Binary variable, equal to 1 if core *k* in CPU *j* in server *i* is idle, and 0 otherwise.
$T_{i,j,k,n}$	Binary variable, equal to 1 if task *n* is assigned to core *k* in CPU *j* in server *i*, and 0 otherwise.
$X_{i,j,k,n}$	The energy cost value of assigning task *n* to core *k* in CPU *j* in server *i*.
$Y_{i,j,k}$	The energy cost value of core *k* in CPU *j* in server *i* when its idle.
RR_n	The minimum amount of RAM requested by task *n*.
RF_n	The minimum amount of Frequency requested by task *n*.
$AR_{i,j,k}$	The available amount of RAM in core *k* in CPU *j* in server *i*.
$AF_{i,j,k}$	The available running Frequency of core *k* in CPU *j* in server *i*.

Figure 3.1 illustrates a simple datacentre configuration with two servers S_1 and S_2. S_1 has two dual-core CPUs and S_2 has one quad-core CPU. A server has two states either ON or OFF. Depending on the server state, each core is assumed to be either active, idle, or switched off, as represented by the green, blue, and red squares respectively. In the scenario depicted by Figure 3.1 all cores were initially off, then two tasks are assigned to $C_{1,1,1}$ and $C_{1,2,1}$ with a cost of $X_{1,1,1,1}$ and $X_{1,2,1,2}$ respectively.

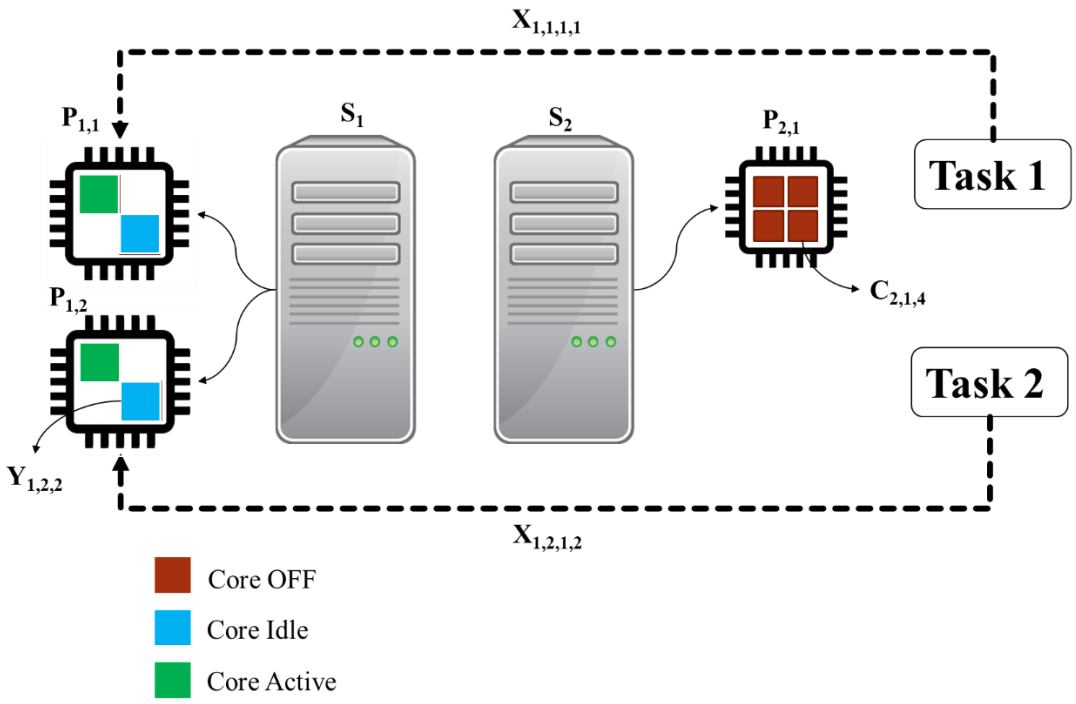

Figure 3.1: A simple datacenter configuration

The model formulation starts with an objective function to minimize the energy consumption when assigning tasks to cores in a datacenter. From Figure 3.1 we observe that the energy consumption comes from the cost of active cores ($X_{i,j,k,n}$) and the cost of idle cores ($Y_{i,j,k}$), while the off cores are assumed to have zero energy consumption. this is represented as follows:

$$Min\left(\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\sum_{n=1}^{N} X_{i,j,k,n}T_{i,j,k,n} + \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} Y_{i,j,k}Q_{i,j,k}\right) \quad (3.1)$$

The objective function described in equation (3.1) has two parts, the first part represents the energy consumed by the active cores. The active cores energy is calculated by summing the energy cost of all the cores that are running a task across the whole datacenter. This cost value typically depends on the size of the task and the characteristics of the core executing the task. The second component of the objective function sums the energy consumed by the remaining idle cores in the datacenter.

Initially, we assume the simultaneous arrival and assignment of a batch of tasks at time t0. With this assumption in mind we start developing the set of constraints that the objective function in equation (3.1) is subjected to. These constraints can be divided into two sets as explained below.

3.2.1 Datacenter configuration constraints. A server *i* can be active or switched off depending on the status of the cores and CPUs inside the server.

Starting from the cores, if a core *k* is active (i.e executing a task) then the CPU *j* that contains the core *k* is considered active as well. This is represented as follows:

$$P_{i,j} = 1 \qquad if \sum_{k=1}^{K} C_{i,j,k} \geq 1 \qquad \forall i \forall j \qquad (3.2)$$

Thus constraint (3.2) ensures that if one or more cores in a CPU are active then the CPU is active. However, if all the cores inside the CPU are not active ($\sum_{k=1}^{K} C_{i,j,k} = 0$) then the value of $P_{i,j}$ is not restricted by constraint (3.2) and it can be 0 or 1. This means that a CPU can be active ($P_{i,j} = 1$) while all the cores inside it are actually not active. To solve this conflict, constraint (3.3) is added to ensure that if the CPU is active ($P_{i,j} = 1$) then at least one of its cores is active and the constraint is expressed as follows:

$$\sum_{k=1}^{K} C_{i,j,k} \geq 1 \qquad if\ P_{i,j} = 1 \qquad \forall i, \forall j \qquad (3.3)$$

Looking back to Figure (3.1) we observe that $C_{1,1,1}$ is active, thus $P_{1,1}$ is active as well. The second core in $P_{1,1}$ is not running a task but it cannot be switched off because the CPU is still active, thus $C_{1,1,2}$ is idle. This leads to constraint (3.4) below:

$$\prod_{k=1}^{K} (C_{i,j,k} + Q_{i,j,k}) = 1 \qquad if\ P_{i,j} = 1 \qquad \forall i, \forall j \qquad (3.4)$$

Constraint (3.4) ensures that if a CPU is active, all the cores inside the CPU must be either active or idle but not switched off.

Constraint (3.5) is added to cover the fact that a core cannot be active and idle at the same time.

$$C_{i,j,k} + Q_{i,j,k} \leq 1 \qquad \forall i, \forall j, \forall k \qquad (3.5)$$

Another observation from Figure (3.1) is that all the cores in the second server S_2 are off. Thus constraint (3.6) is formulated to ensure that if a server is off all the cores inside the server are off.

$$\sum_{j=1}^{J} \sum_{k=1}^{K} C_{i,j,k} + Q_{i,j,k} = 0 \qquad if\ S_i = 0 \qquad \forall i \qquad (3.6)$$

One last scenario that can happen to the cores is when a CPU is idle. In this case all the cores inside the idle CPU must be idle. Constraint (3.7) ensures that if a CPU is idle then all its cores must be idle.

$$\sum_{k=1}^{K} Q_{i,j,k} = K \qquad if\ G_{i,j} = 1 \qquad \forall i, \forall j \qquad (3.7)$$

The same scenarios above apply to CPUs and Servers. Constraint (3.8) ensures that if one or more of the CPUs in server *i* are active then server *i* must be active.

$$S_i = 1 \qquad if\ \sum_{j=1}^{J} P_{i,j} \geq 1 \qquad \forall i \qquad (3.8)$$

Similar to constraint (3.2), there is no restriction on the value of S_i by constraint (3.8) if all the CPUs are not active $\left(\sum_{j=1}^{J} P_{i,j} = 0\right)$. This may lead to a case where the server is considered active ($S_i = 1$) while all the CPUs inside it are not active. To get rid of this problem, constraint (3.9) is added to ensure that if a server is active then at least one of its CPUs is active and the constraint is written as follows:

$$\sum_{j=1}^{J} P_{i,j} \geq 1 \qquad if\ S_i = 1 \qquad \forall i \qquad (3.9)$$

Constraint (3.10) makes sure that if a server is active then all the CPUs in an active server are either active or idle only (they cannot be off).

$$\prod_{j=1}^{J} (P_{i,j} + G_{i,j}) = 1 \qquad if\ S_i = 1 \qquad \forall i \qquad (3.10)$$

Constraint (3.11) guarantees that when a server *i* is off, all the CPUs inside the server are off.

$$\sum_{j=1}^{J} P_{i,j} + G_{i,j} = 0 \qquad if\ S_i = 0 \qquad \forall i \qquad (3.11)$$

Constraint (3.12) ensures that a CPU cannot be active and idle at the same time.

$$P_{i,j} + G_{i,j} \leq 1 \qquad \forall i, \forall j \qquad (3.12)$$

3.2.2. Tasks requirement and scheduling constraints. The following constraints are tasks related.

Constraint (3.13) takes care of the assumption that a task *n* must be assigned to one core only at any given time. Which means all tasks must be assigned simultaneously and a single task will be fully executed by one core only and cannot be split into multiple cores regardless of how big the task is.

$$\sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} T_{i,j,k,n} = 1 \qquad \forall n \qquad (3.13)$$

Constraint (3.14) associates the active cores with the tasks that are assigned to these cores. It also ensures that a not-active core (can be idle or switched off) is not assigned any task.

$$\sum_{n=1}^{N} T_{i,j,k,n} \triangleq \begin{cases} 1 & if\ C_{i,j,k} = 1 \\ 0 & if\ C_{i,j,k} = 0 \end{cases} \qquad \forall i, \forall j, \forall k \qquad (3.14)$$

Constraint (3.15) is added to ensure that the number of active cores is equal to the total number of tasks to be scheduled. If the number of tasks is *N* then constraint (3.15) is written as follows:

$$\sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} C_{i,j,k} = N \qquad (3.15)$$

Constraint (3.16) ensures that if task *n* is assigned to core *k* in CPU *j* in server *i*, the amount of RAM required by task *n* is less than or equal to the amount of RAM available in core *k*.

$$RR_n T_{i,j,k,n} \leq AR_{i,j,k} \qquad \forall i, \forall j, \forall k, \forall n \qquad (3.16)$$

Constraint (3.17) ensures that if task *n* is assigned to core *k* in CPU *j* in server *i*, the frequency rate required by a task *n* is less than or equal to the highest frequency available in core *k*.

$$RF_n T_{i,j,k,n} \leq AF_{i,j,k} \qquad \forall i, \forall j, \forall k, \forall n \qquad (3.17)$$

Table 3.2 presents the complete ILP optimization model with the objective function and all the constraints discussed in this section.

Table 3.2: The ILP Optimization Model

$$Min\left(\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\sum_{n=1}^{N} X_{i,j,k,n}T_{i,j,k,n} + \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} Y_{i,j,k}Q_{i,j,k}\right) \quad (3.1)$$

s.t.

$P_{i,j} = 1$	$if \sum_{k=1}^{K} C_{i,j,k} \geq 1$	$\forall i$	(3.2)
$\sum_{k=1}^{K} C_{i,j,k} \geq 1$	$if\ P_{i,j} = 1$	$\forall i, \forall j$	(3.3)
$\prod_{k=1}^{K}(C_{i,j,k} + Q_{i,j,k}) = 1$	$if\ P_{i,j} = 1$	$\forall i, \forall j$	(3.4)
$C_{i,j,k} + Q_{i,j,k} \leq 1$		$\forall i, \forall j, \forall k$	(3.5)
$\sum_{j=1}^{J}\sum_{k=1}^{K} C_{i,j,k} + Q_{i,j,k} = 0$	$if\ S_i = 0$	$\forall i$	(3.6)
$\sum_{k=1}^{K} Q_{i,j,k} = K$	$if\ G_{i,j} = 1$	$\forall i, \forall j$	(3.7)
$S_i = 1$	$if\ \sum_{j=1}^{J} P_{i,j} \geq 1$	$\forall i$	(3.8)
$\sum_{j=1}^{J} P_{i,j} \geq 1$	$if\ S_i = 1$	$\forall i$	(3.9)
$\prod_{j=1}^{J}(P_{i,j} + G_{i,j}) = 1$	$if\ S_i = 1$	$\forall i$	(3.10)
$\sum_{j=1}^{J} P_{i,j} + G_{i,j} = 0$	$if\ S_i = 0$	$\forall i$	(3.11)
$P_{i,j} + G_{i,j} \leq 1$		$\forall i, \forall j$	(3.12)
$\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} T_{i,j,k,n} = 1$		$\forall n$	(3.13)
$\sum_{n=1}^{N} T_{i,j,k,n} \triangleq \begin{cases} 1 & if\ C_{i,j,k} = 1 \\ 0 & if\ C_{i,j,k} = 0 \end{cases}$		$\forall i, \forall j, \forall k$	(3.14)
$\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} C_{i,j,k} = N$			(3.15)
$RR_n T_{i,j,k,n} \leq AR_{i,j,k}$		$\forall i, \forall j, \forall k, \forall n$	(3.16)

3.3. Dynamic Scheduling

In the second phase of the problem a new batch of tasks are to be scheduled at time t_1 after the first batch is already scheduled at time t_0. The new tasks will be

assigned to the remaining free cores following the scheduling of the first batch. The tasks that are already assigned at t_0 will not be considered or reassigned again. Based on the outcomes of scheduling the first batch, the following constraints will be added or modified as described below:

- Constraint (3.18) is added to ensure that all the cores that are still running a task from batch t_0 are marked as busy,

$$\sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} C_{i,j,k} = B \tag{3.18}$$

  where B is the number of tasks from the first batch that are still running at t_1.

- Constraint (3.19) is added to ensure that if a core $C_{i,j,k}$ is busy then no task from the second batch is assigned to it,

$$\sum_{n=1}^{M} T_{i,j,k,n} = 0 \qquad if\ C_{i,j,k} = 1 \tag{3.19}$$

  where M is the number of the new tasks in the second batch at t_1.

- Constraint (3.15) is modified to ensure that the total number of active cores is now equal to the total number of tasks to be scheduled (tasks from the second batch (M)) plus the number of tasks that are still running from the first batch (B), instead of just the total number of tasks from the first batch (N) as in equation (3.15).

$$\sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} C_{i,j,k} = B + M \tag{3.20}$$

3.4. Differentiating Between the Running Modes of the Servers

Next and to further study the parameters that affect optimization, some of the initial assumptions are modified to the following:

- All cores inside the servers are assumed to operate at a certain voltage level.
- Each core is assumed to have two running modes: High and Low.
- In High mode, the core runs with the maximum available frequency and voltage, while in Low mode, the core runs at the minimum available frequency and voltage.

- Cores inside the same server can run in different modes.

The motivation behind these assumptions is to enable a high capability server that is already executing a large size task to execute another smaller task on one of its idle cores. Otherwise we need to switch on a low capability server to run that small size task.

To implement the aforementioned assumptions, new variables are added, and the objective and some of the constraints are modified from the initial ones as given in Table 3.2. The new added variables and their definitions are provided in Table 3.3.

Table 3.3: New and Modified Variables and Their Definitions

Variables	**Description**
$L_{i,j,k,n}$	Binary variable, equal to 1 if task *n* is executed in core *k* in CPU *j* in server *i* in Low mode and 0 otherwise.
$H_{i,j,k,n}$	Binary variable, equal to 1 if task *n* is executed in core *k* in CPU *j* in server *i* in High mode and 0 otherwise.
$X_{i,j,k,n}$	The energy cost value of executing task *n* in core *k* in CPU *j* in server *i* while running in High mode.
$Y_{i,j,k,n}$	The energy cost value of executing task *n* in core *k* in CPU *j* in server *i* while running in Low mode.
$Z_{i,j,k}$	The energy cost value of core *k* in CPU *j* in server *i* when its idle.
$AH_{i,j,k}$	The available High frequency in core *k* in CPU *j* in server *i*.
$AL_{i,j,k}$	The available Low frequency in core *k* in CPU *j* in server *i*.

The modified objective function is given by equation (3.21) as follows:

$$Min\left(\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\sum_{n=1}^{N}X_{i,j,k,n}L_{i,j,k,n}+\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\sum_{n=1}^{N}Y_{i,j,k,n}H_{i,j,k,n}+\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}Z_{i,j,k}Q_{i,j,k}\right) \quad (3.21)$$

Subject to the same constraints as the original model with the following additions and modifications:

- Constraint (3.22) is added to ensure that a core cannot be running in High and Low mode at the same time.

$$L_{i,j,k,n}+H_{i,j,k,n}\leq 1 \qquad \forall i,\forall j,\forall k,\forall n \quad (3.22)$$

- Constraint (3.23) is added to ensure that if a task is executed on a specific core then it must be executed at either High or Low mode.

$$L_{i,j,k,n}+H_{i,j,k,n}=1 \quad if\ T_{i,j,k,n}=1 \qquad \forall i,\forall j,\forall k,\forall n \quad (3.23)$$

- Constraint (3.24) is added to ensure that if a task is not executed on any core then it cannot be executed at neither High nor Low mode.

$$L_{i,j,k,n}+H_{i,j,k,n}=0 \quad if\ T_{i,j,k,n}=0 \qquad \forall i,\forall j,\forall k,\forall n \quad (3.24)$$

- Constraints (3.22), (3.23), and (3.24) are summed in equation (3.25).

$$L_{i,j,k,n}+H_{i,j,k,n}-T_{i,j,k,n}=0 \qquad \forall i,\forall j,\forall k,\forall n \quad (3.25)$$

- Constraint (3.17) is replaced with two constraints, (3.26) and (3.27), to ensure that the required frequency by task *n* does not exceed the available high or low frequency of the core.

$$RF_nH_{i,j,k,n}\leq AH_{i,j,k} \qquad \forall i,\forall j,\forall k,\forall n \quad (3.26)$$

$$RF_nL_{i,j,k,n}\leq AL_{i,j,k} \qquad \forall i,\forall j,\forall k,\forall n \quad (3.27)$$

3.5. Estimating the Energy Cost

The energy cost of assigning a task *n* to core *k* in CPU *j* in server *i* represented by $X_{i,j,k,n}$ is modelled and tested using several approaches. The goal was to get a

realistic estimate to the energy consumption of a core. The equations used to model the cost are as follows:

Equation (3.28) is used in the first test where initially the goal is just to validate the model and the energy cost is not of a significant concern:

$$X_{i,j,k,n} = rand \tag{3.28}$$

where $rand$ is a random constant assigned to each task per core.

Equation (3.29) is the first attempt to include the task size in estimating the energy cost,

$$X_{i,j,k,n} = \frac{TaskSize_n}{500} * U_{i,j,k} \tag{3.29}$$

where TaskSize_n is the size of task n in number of instructions and $U_{i,j,k}$ represents the active energy cost of the CPU. The task size is divided by 500 just to scale the numbers down (500 is chosen because the smallest task size is assumed to be 500 instructions). The parameter $\text{U}_\text{i,j,k}$ is a randomly generated value that is assigned to each CPU in the datacenter. This value is equal in all the cores of the same CPU.

Equation (3.30) incorporate the core characteristics as well as the task size in the energy cost estimation,

$$X_{i,j,k,n} = \frac{TaskSize_n}{100} * v_{i,j,k}^2 \tag{3.30}$$

where v_k^2 is the square of the voltage of core k in CPU j in server i. The task size is scaled down by dividing it by 100. This equation comes from the proportionality relationship between power and frequency and voltage as in equation (3.31).

$$P \propto f \times v^2 \tag{3.31}$$

The task size is in number of instructions. All the cores are assumed to execute one instruction per cycle. This means that the task size in equation (3.30) represents the number of cycles the core need to complete the execution of a given task.

Chapter 4. Experimental Setup

In this chapter, the validation and testing process of the proposed models are described. The generated optimization models are validated and tested using SAT-based 0-1 PB solvers named NaPS, Minisat+, and Sat4j, and a generic ILP solver named CPLEX. The solvers compute the minimum energy consumption of the datacenter and the corresponding assignment of tasks to the cores that would achieve this power.

To test a model, several datacenter instances were created with different number of servers, different number and types of CPUs (dual-core or quad-core) in each server, and different number of tasks to be scheduled. The tests are carried out in an incremental manner as described in the following sections.

4.1. Test 1: Validating the Model Without Any Task Characteristics.

Initially, a simpler version of the model where task characteristics are not taken into consideration is tested. The tested instances consist of different datacentre configurations where a certain number of tasks is to be scheduled to the available cores. All cores are assumed to be capable of executing any task. The testing is carried out using datacentre instances that are randomly generated. Each instance has different number of servers. Each server is assumed to contain 1 to 4 CPUs. Each CPU can be either *dual* or *quad* core. Initially, the *active* and *idle* energy costs of the cores were assigned randomly but, are assumed to be the same for all cores inside the same CPU. Table 4.1 shows the instances used in this preliminary test and the number of tasks to be assigned in each case.

The initial model used and tested is a simpler version of the complete model with the objective function being:

$$Min\left(\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}X_{i,j,k}C_{i,j,k}+\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}Y_{i,j,k}Q_{i,j,k}\right) \quad (4.1)$$

where $X_{i,j,k}$ have a different definition from the original model of equation (3.1):

$X_{i,j,k}$: The energy cost value of core *k* in CPU *j* in server *i* when its active.

Furthermore, the model is subjected to constraints (3.2) through (3.15) of the complete model. Only constraints (3.16) and (3.17) that deal with the task characteristics are excluded. All the solvers are used in this test.

Table 4.1 : Testing Instances for Test 1

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks
1	2	3	8	2
2	2	4	12	4
3	3	5	16	6
4	6	17	44	10
5	8	24	72	20
6	12	36	108	25

4.2. Test 2: Comparing the Model from Test 1 to Other Scheduling Techniques.

In this test, the simplified model of Test 1 is compared against two other scheduling techniques, namely, Round Robin, and Maximum Possible Value.

4.2.1. Round Robin scheduling algorithm. This algorithm assigns the task to the first available core that can execute it. It does not consider minimizing the energy consumption while assigning tasks, and hence consumes minimal time in making these assignments.

4.2.2. Maximum possible value scheduling algorithm. This is not an actual scheduling algorithm but, it represents the worst possible scenario that can happen if no scheduling algorithm is used at all. In this case, the tasks are assigned to the set of cores which would execute them with the maximum energy consumption possible. The maximum value is found by changing the minimization function of the model to a maximization function and then solving it using a solver.

The testing instances used in Test 2 are the same instances of Table (4.1). This test is carried out using CPLEX solver alone, because we are only interested in finding and comparing the energy values and not the different solvers performance.

4.3. Test 3: Studying the Effect of Varying the Number of Servers.

The third test is conducted to analyse the effect of redistributing the same number of CPUs with the same total number of cores but using more servers. To do this, we start with a datacentre consisting of 4 servers each containing 6 CPUs. Half the CPUs are dual-core, while the other half are quad-core CPUs. The total number of CPUs is 24 with 72 total cores. The 24 CPUs are then redistributed into 6, 8, 12, and 24 servers, while the total number of CPUs and cores remained the same. In each case the proposed model is used to assign 20 Tasks to the datacentre. NaPS and CPLEX solvers are used for this test. Table 4.2 shows the instances used in this test.

Table 4.2: Testing Instances for Test 3

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks
1	4	24	72	20
2	6	24	72	20
3	8	24	72	20
4	12	24	72	20
5	24	24	72	20

4.4. Test 4: Adding Task Characteristics to the Model.

In the fourth test, the objective function in (3.1) is tested with tasks now having a predefined characteristic which is task size. All the constraints (3.2) up to (3.15) are included in this test. Again constraint (3.16) and (3.17) are excluded since frequency and RAM requirements are not implemented yet. The cost $X_{i,j,k,n}$ of each task per core is assumed as a positive integer that is proportional to the task size. This means the bigger the size, the higher the energy consumed by the cores to execute the tasks.

The following assumptions are applied:

- All cores inside the same CPU are identical, that is active cores and idle cores have the same energy needs.
- Task sizes are chosen randomly in the range of 500 – 10,000 instructions.
- The active energy cost for each CPU is chosen randomly in the range of 2 to 40.

- The idle energy cost for each CPU is assumed to be 50% of the active energy.

The cost $X_{i,j,k,n}$ is calculated using equation (3.29). The model is implemented and tested using the same testing instances in Table 4.1. The values of $X_{i,j,k,n}$ for each task per core were generated and assigned randomly for this preliminary test. The three PB-SAT solvers and CPLEX are used to test the model.

4.5. Test 5: Testing the Utilization of a Datacentre.

In this test, a fixed datacentre configuration is tested using the objective function in equation (3.1). The number of tasks is increased in each case until it is equal to the total number of cores in the datacentre (100% utilization). All the four solvers are used in this test. The minimum energy consumption was recorded after running the solvers for 100, 1000, and 10000 seconds. Table 4.3 shows the tested configuration with the total size of all the tasks in each case.

Table 4.3: Testing Instances for Test 5

Table 4.3: **Instance No.**	**No. of servers**	**Total no. of CPUs**	**Total no. of cores**	**No. of Tasks**	**Total Inst. of all Tasks**
1	6	12	36	6	21500
2	6	12	36	12	51000
3	6	12	36	18	86000
4	6	12	36	24	120500
5	6	12	36	30	158000
6	6	12	36	36	183000

4.6. Test 6: Testing the Model Using Real Benchmark.

To approximate real life datacenters energy consumption, the list of servers in Table 4.4 with typical energy values are used to test the proposed model. The first two columns in Table 4.4 represents the Vendor and the server enclosure name. The Processor column lists the specification of the CPU/s used in the server. The Processor specifications are: the name of the CPU, the clock frequency in MHz, the number of CPU chips in the server, and the total number of cores in the server. The last three columns of Table 4.4 lists the average power consumption of the server when its active at 100% utilization and when its idle, and the number of operations

that can be executed using one Watt. The datacenter configurations that are tested here are derived based on real-life servers and their energy consumption values. These configurations are as follows:

Config. 1. It consists of two servers: One Colfax International - AMD Opteron 2216HE with two dual-core CPUs, and one DEPO Computers - Intel Core i5-4570 with one quad-core CPU.

Config. 2. It consists of two servers: One Super Micro Computer, Inc. - Intel Xeon processor 5160 with two dual-core CPUs, and one Huawei Technologies Co., Ltd - Intel Xeon E5-2609 with two quad-core CPUs.

Config. 3. It consists of three servers: one Huawei Technologies Co., Ltd - Intel Xeon E5-2609 with two quad-core CPUs, one Super Micro Computer, Inc. - Intel Xeon processor 5160 with two dual-core CPUs, and one DEPO Computers - Intel Core i5-4570 with one quad-core CPU.

Config. 4. It consists of six servers: one Super Micro Computer, Inc. - Intel Xeon processor 5160 with two dual-core CPUs, one Colfax International - AMD Opteron 2216HE with two dual-core CPUs, one Acer Incorporated - Intel Xeon L5520 with four quad-core CPUs, one Huawei Technologies Co., Ltd - Intel Xeon E5-2609 with two quad-core CPUs, one ASUSTeK Computer Inc. - Intel Xeon L5430 with two quad-core CPUs, and one DEPO Computers - Intel Core i5-4570 with one quad-core CPU.

Config. 5. It consists of eight servers: one Quanta Computer Inc. – Intel E3-1260L v5 with one quad-core CPU, one DEPO Computers - Intel Core i5-4570 with one quad-core CPU, one Super Micro Computer, Inc. - Intel Xeon processor 5160 with two dual-core CPUs, one Colfax International - AMD Opteron 2216HE with two dual-core CPUs, two Huawei Technologies Co., Ltd - Intel Xeon E5-2609 each with two quad-core CPUs, one ASUSTeK Computer Inc. - Intel Xeon L5430 with two quad-core CPUs, and Hewlett-Packard Company - Intel Xeon L7345 with eight quad-core CPUs.

The active and idle energy costs are calculated from the Avg. Watts @ 100% and Avg. Watts @ idle values respectively. To estimate the value per core, the value for server power is divided by the number of cores inside the server. The datacenter

configurations are tested using the objective function in equation (4.1) using constraints (3.2) to (3.15). The model performance is also compared against Round Robin and Maximum Possible Value algorithms in terms of the energy consumption value found by each method.

Table 4.4: Energy Consumption Benchmark [42] used in Test 6

Vendor	System Enclosure	Processor				Avg. watts @ 100%	Avg. watts @ idle	Ops. per watt
		CPU Name	MHz	Chip	Core			
Quanta Computer Inc.	QuantaGrid S31A-1U	Intel E3-1260L v5	2900	1	4	47.9	14.4	7,908
DEPO Computers	DEPO Race X340H	Intel Core i5-4570	3200	1	4	131	83.2	1,469
Huawei Technologies Co., Ltd	RH2288H V2	* Intel Xeon E5-2609 RAM = 48 GB	2400	2	8	137	68.7	2,716
ASUSTeK Computer Inc.	ASUS RS160-E5	* Intel Xeon L5430 RAM = 8GB	2666	2	8	173	89.4	1,020
Acer Incorporated	Gateway GW1000-GW170 F1 GW1000	Intel Xeon L5520	2266	4	16	386	145	1,588
Hewlett-Packard Company	Proliant DL580 G5	Intel Xeon L7345	1860	4	16	387	271	546
Acer Incorporated	Gateway GW2000h-GW170h F1 GW2000h	Intel Xeon L5520	2266	8	32	748	260	1,674
Hewlett-Packard Company	ProLiant SL2x170z G6	Intel Xeon L5530	2400	8	32	664	191	2,316
Colfax International	CX2266-N2	* AMD Opteron 2216HE RAM = 16 GB	2400	2	4	276	164	203
Super Micro Computer, Inc.	Supermicro 6025B-TR+	* Intel Xeon processor 5160 RAM = 8 GB	3000	2	4	315	216	318

4.7. Test 7: Adding Frequency and RAM requirements.

In the previous tests, the model included only one characteristic of a Task which is Task's size. Also, the assumption was that any task can be executed on any server. This assumption led to large exponentially increasing search spaces. For a more realistic model and smaller search space, new Task requirements that control the assignment of tasks to servers are introduced to the model. Each Task is now assumed to have a required minimum frequency and minimum RAM size. This completes the model that is described in section 3.2 in the methodology chapter. To implement these requirements some preprocessing is needed to decide which cores can run which tasks. This is done before using any of the solvers. After providing the solvers with the list of cores that can execute each task, the solvers start working on the model to assign these tasks to the suitable cores with the minimum energy consumption. The

cost value $X_{i,j,k,n}$ used in this test is calculated from the task size directly using equation 4.2.

$$X_{i,j,k,n} = \frac{TaskSize_n}{100} \tag{4.2}$$

To test the new model the same datacenter configuration of Table 4.3 is used. The input instances are different of course since they include the new characteristics of tasks and servers. The RAM and Frequency of both tasks and servers were assigned as follows:

- Task RAM requirement: randomly generated in the range of 4 GB to 32 GB.
- Task Frequency requirement: randomly generated in the range of 2200 MHz to 2800 MHz (increments of 200 MHz only).
- Core RAM and Frequency: chosen based on the real data of four different servers (two quad-core and two dual-core). the servers chosen are marked with a (*) in Table 4.4.

4.8. Test 8: Testing the Full Modified Model.

After adding the task requirements and all the modifications required to implement them, the model is tested using the three PB-SAT solvers and CPLEX. Table 4.5 shows the instances used in this test.

Table 4.5: Testing Instances of Test 8

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks	Total Tasks Size (#inst)
1	2	4	12	6	25500
2	3	6	20	10	41000
3	4	8	24	12	47000
4	5	10	28	14	54500
5	7	14	44	22	87000
6	10	20	68	34	128500

4.9. Test 9: Adding High and Low Execution Modes.

To further optimize the energy consumption of a datacenter, cores are assumed to run in two modes: High, and Low. In High mode the core runs with the maximum frequency and voltage allowed, while in low mode, the core runs with the minimum

frequency and voltage possible. In this test, the model is modified as explained in section 3.4. The testing instances for this test are generated randomly from a pool of servers. The task requirements are created depending on to the task size. Table 4.6 shows the task size ranges and the corresponding tasks' requirements. Table 4.7 includes partial servers' specifications. Table 4.8 lists the instances created for this test. Each setup is tested twice using different number of tasks. The instances are first solved using all the solvers to find the minimum energy consumption and compare the solvers' performance. Then, the same instances are solved using Round Robin and Maximum Possible Value methods to compare their performance against the proposed model.

Table 4.6: Tasks' Size Ranges and Requirements

Task Size Range (#instructions)	Required Frequency (MHz)	Required RAM (GB)
500 – 2000	800	1
2500 – 4000	1000	2
4500 – 6000	2400	2
6500 – 8000	2600	3
8500 – 10000	2800	4

Table 4.7: Test 9 Servers' Information

Server No.	Server Type	High Freq. (MHz)	High Volt. (Volt)	Low Freq. (MHz)	Low Volt. (Volt)	RAM (GB)
Server1	Dual	2400	1.45	800	0.9	2
Server2	Dual	3000	1.35	1000	1.0	4
Server3	Quad	2400	1.35	1000	0.9	2
Server4	Quad	2600	1.45	800	1.0	4

Table 4.8: The testing instances for Test 9

Instance No.	**No. of servers**	**Total no. of CPUs**	**Total no. of cores**	**No. of Tasks**	**Total Tasks Size (#instructions)**
1	2	4	12	2	7,500
2	2	4	12	6	25,500
3	3	6	20	5	23,000
4	3	6	20	10	41,000
5	4	8	24	8	29,500
6	4	8	24	12	47,000
7	8	16	52	18	68,500
8	8	16	52	26	98,500

4.10. Test 10: Dynamic Task Scheduling

In all of the analysis above we assumed that all the tasks arrive as one batch and scheduled to execute simultaneously at time t_0. In this segment of our tests, we consider the case of another batch of tasks arriving at time t_1. The new batch of tasks is to be assigned to the remaining free cores of the datacenter while minimizing the energy consumption. The tasks of the first batch (t_0) that are still running at t_1 will not be rescheduled. However, their energy consumption will be taken into account when calculating the overall energy consumption of the second batch. Table 4.9 shows the testing instances used with a second batch of tasks at time t_1. Number of remaining tasks at t_1 refers to how many tasks from batch t_0 are still running when the second batch of tasks arrived at t_1. Number of new tasks at t_1 is how many new tasks arrived at t_1. The model is tested with the objective function in equation (4.1). Both the proposed model and Round Robin are used to schedule the tasks at t_0 and t_1 and the minimum energy consumption in each case is compared.

Table 4.9: Testing Instances used in Test 8

Instance No.	**No. of servers**	**Total no. of CPUs**	**Total no. of cores**	**No. of Tasks at t_0**	**No. of remaining Tasks at t_1**	**No. of new Tasks at t_1**
1	2	3	8	2	1	2
2	2	4	12	4	2	3
3	3	6	20	6	2	6
4	6	18	68	10	4	8
5	8	24	92	20	8	16

Chapter 5. Results and Analysis

In this chapter, we present the results of the tests described in Chapter 4, in the appropriate tables and graphs format. The performance of the different solvers is evaluated under the different testing scenarios. The relation between the solvers' performance and the ILP model formulation is analysed and discussed. All the tests are conducted on an Intel (R) Core (TM) *i5* machine @ 2.50 GHz – 2.70 GHz with 8 GB of RAM and Windows 10 Home 64-bit operating system.

5.1. Test 1: Results and Evaluation

This section presents and analyses the result of Test 1 specified in Section 4.1. The objective of this test is to validate the simpler version of the proposed model before implementing the full model. Table 5.1 shows the minimum energy consumption returned by the solvers and the average time taken by each solver.

Table 5.1: The results of all solvers using the objective function in equation (4.1)

Instance No.	**No. of Tasks**	**No. of Cores**	**Minimum Energy**	**CPU Time (seconds)**			
				NaPS	**Minisat+**	**Sat4j**	**CPLEX**
1	2	8	9	0.015	**0.004**	0.033	0.15
2	4	12	14	0.020	**0.008**	0.051	0.18
3	6	16	39	0.068	**0.053**	0.065	0.20
4	10	44	46	0.539	1.20	0.422	**0.37**
5	20	72	172	25.0	85.3	>1000 (172 in 942.5 s)	**1.21**
6	25	108	262	202.4	>1000 (262 in 779 s)	>1000 (289 in 937.5 s)	**2.41**

As presented in Table 5.1, CPLEX found the minimum energy consumption in all the tested instances. The time taken by CPLEX did not exceed 2.41 seconds for any instance. The PB-SAT solvers varied in their results. NaPS similar to CPLEX, also found the minimum energy in all the cases, however, the time taken by NaPS increased as the number of resources (servers, CPUs, and cores) and the number of tasks to be assigned to these resources are increased. NaPS execution time reached a maximum of 202.4 seconds to find the minimum in instance 6. Minisat+ found the

minimum in all the cases except the last one (instance 6) where it did not manage to finish within the allotted time (1000 seconds). Nevertheless, Minisat+ found the minimum energy consumption value (262) after 779 seconds, however, it continued with the search to find a better solution until the time ran out. The same issue occurred with Sat4j in instance 5, while in instance 6 the best value found by Sat4j within 1000 seconds was 289 with an error percentage of 10% of the minimum value. Figure 5.1 illustrate the difference in CPU time taken by all the solvers. We can see that Minisat+ is taking the lead in terms of execution time (CPU time) in the first 3 instances but, as the instance size increased CPLEX exhibited superior performance.

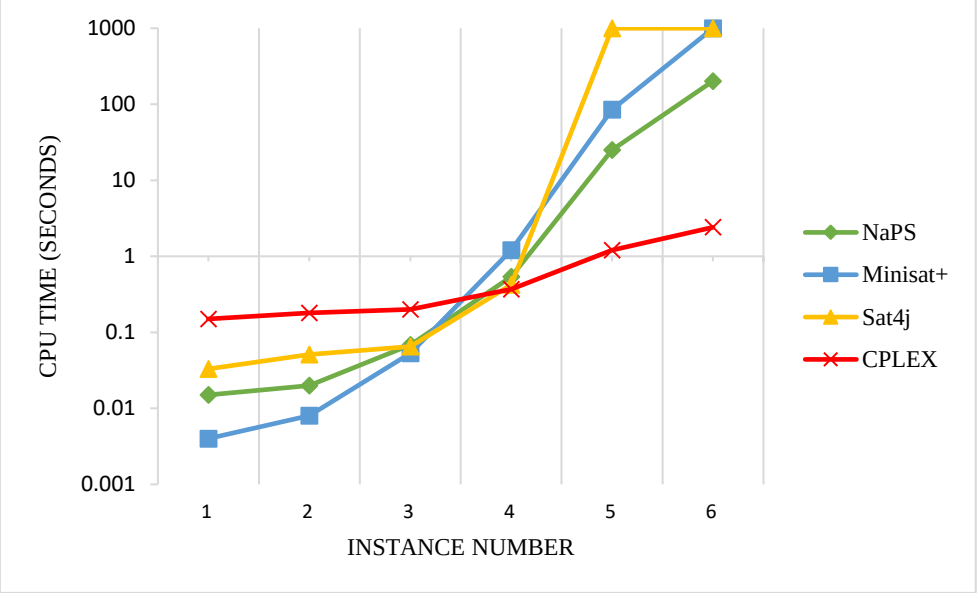

Figure 5.1: Test 1 Results - The average CPU time taken by the different solvers to find the minimum energy consumption.

Figure 5.2 depicts the relation between the number of cores in the datacentre, the minimum energy consumption, and the time taken by NaPS to solve the model. We can see that all the three factors increase in a similar manner. In other words, as the number of cores increase, the minimum energy consumption increases, and hence, NaPS takes longer time to find the minimum energy consumption.

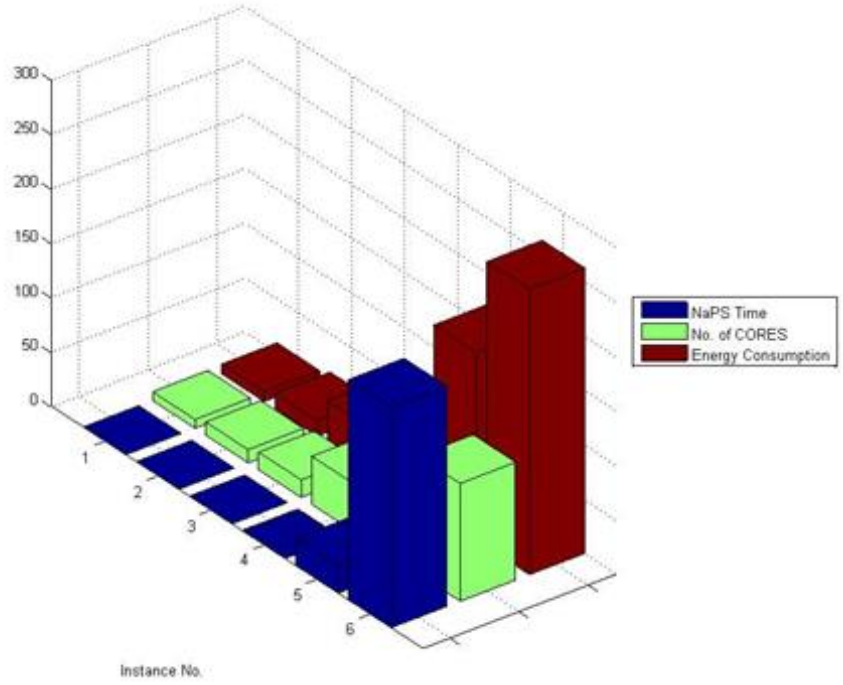

Figure 5.2: Illustration of Test 1 results.

5.2. Test 2: Results and Evaluation

In this section, the results of the test described in Section 4.2 are presented and evaluated. Table 5.2 lists the energy consumption values found by CPLEX using the proposed model, and those returned by Round Robin, and Maximum Possible Value methods. Evidently, we see that the proposed model returned the minimum energy values in all of the tested instances.

Table 5.2: Comparison of the proposed model against other techniques.

Instance No.	**Energy consumption**			**CPU Time (seconds)**
	Maximum Possible Value	**Round Robin**	**Proposed Model**	**CPLEX**
1	26	12	**9**	0.15
2	50	20	**14**	0.18
3	100	57	**39**	0.20
4	196	55	**46**	0.37
5	534	216	**172**	1.21
6	1483	422	**262**	2.41

The results show that the proposed model can save up to 82.3% in energy when compared to the maximum possible energy consumption and up to 37.9% when compared against the energy consumption of Round Robin. The CPU time column lists the time taken by CPLEX to find the minimum using the proposed model. We observe that the time increases as the instances grow in size as mentioned earlier. On the other hand, a Round Robin solution is typically faster. Thus, in case of time sensitive applications and considering only the tested methods, one might consider using Round Robin to save time, meanwhile sacrificing energy consumption. However, if considering all options, some heuristic methods might give a near optimal solution in a reasonable amount of time. Figure 5.3 is a graphical illustration of the results presented in Table 5.2.

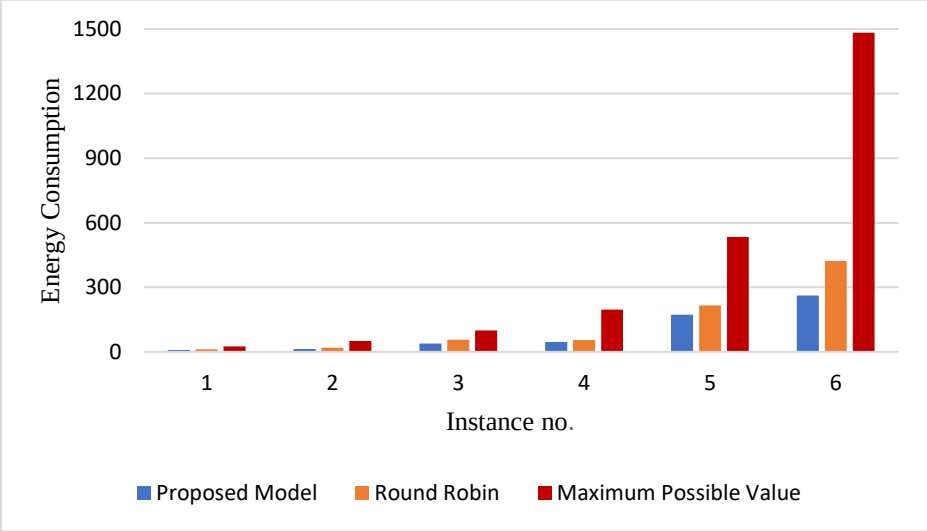

Figure 5.3: Test 2 Results - The energy consumption values found using the different tested methods.

5.3. Test 3: Results and Evaluation

Table 5.3 depicts the results of the third test described in Section 4.3. The objective of this test is to study the effect of changing the datacentre setup in a predefined manner on the energy consumption of the datacentre. From the results we observe that as we increase the number of servers while keeping the same total number of CPUs and cores, the energy consumption value needed to execute the same number of tasks decreases. This can be attributed to the fact that in the proposed

model, we switch on and off resources at the server level. For example, consider instance 1 and instance 5 from both Tables 4.2 and 5.3. In instance 1 we have 4 servers each containing 6 CPUs with a total of 72 cores. Instance 5 consist of 24 servers each with one single CPU and with the same number of 72 total cores across all servers. Running one task on a single core of a server from instance 1, will leave the other 3 CPUs in the same server completely idle. Keep in mind that idle CPUs consume less energy than active CPUs, but they still consume energy nonetheless. However, running one task on a single core of a server from instance 5 leaves no idle CPUs at all. Because each server contains only one CPU and the CPU is currently active and executing the task on one of its cores. All the other CPUs are inside separate servers that are all totally switched off and hence consume zero energy. From the results the energy consumption decreases until it reaches a minimum when the server to CPU ratio is 1:1. Figure 5.4 illustrate the results of Test 3.

Table 5.3: The energy consumption values when varying the number of servers.

Instance No.	**No. of servers**	**Total no. of CPUs**	**Total no. of cores**	**No. of Tasks**	**Min Energy**	**CPU Time (seconds)**	
						NaPS	**CPLEX**
1	4	24	72	20	280	6.16	**1.19**
2	6	24	72	20	224	9.83	**1.05**
3	8	24	72	20	192	8.80	**1.18**
4	12	24	72	20	200	24.47	**1.07**
5	24	24	72	20	92	16.00	**1.15**

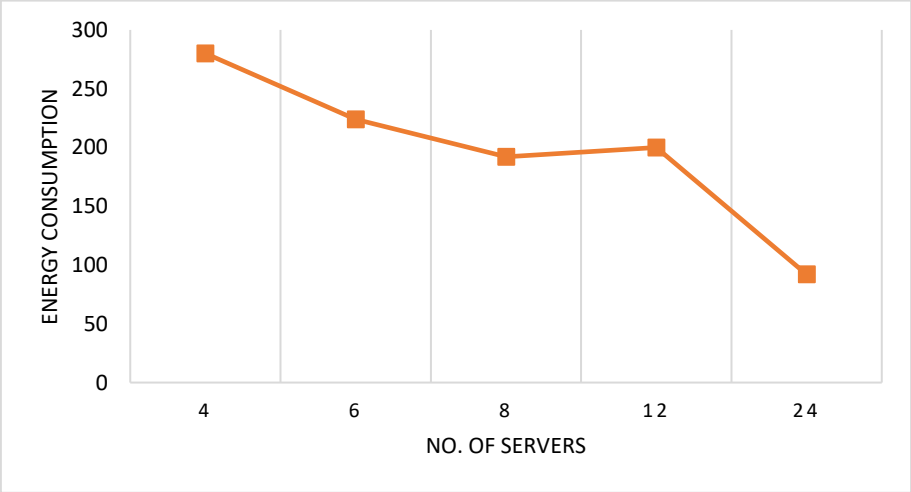

Figure 5.4: Test 3 Results - Energy consumption vs. number of servers.

5.4. Test 4: Results and Evaluation

This section presents and evaluates the results of the test specified in Section 4.4. The objective of this test is to examine the effect of introducing the task size into the model. Table 5.4 shows the results of solving the model after introducing task sizes. Past instance 4, the PB-SAT solvers failed to find the minimum energy consumption within the allotted time of 1000 seconds. CPLEX however, found the minimum of instance 5, but in instance 6 it took around 8000 seconds to get the optimum value.

Table 5.4: Energy consumption and Solvers' running times after adding tasks to the objective function.

Instance No.	**Minimum Energy**	**CPU Time (seconds)**			
		NaPS	**Minisat+**	**Sat4j**	**CPLEX**
1	11	0.015	**0.004**	0.053	0.16
2	18	0.067	**0.029**	0.072	0.14
3	23	0.281	0.451	**0.161**	0.18
4	55	1.197	1.711	9.9 Hrs	**0.40**
5	214	> 1000	> 1000	> 1000	**3.30**
6	286	> 1000	> 1000	> 1000	> 1000

The dramatic increase in solving time comes from the increase in the number of variables in the objective function as a result of adding the task size to the model and hence the huge search space. In instance 6 for example the total number of variables in the objective function is 2808 binary decision variables, with (2^{2808}) possible solutions. To work around this problem further modifications of the model are implemented and tested as discussed in a later section.

5.5. Test 5: Results and Evaluation

In this section the results of Test 5 specified in Section 4.5 are presented and evaluated. Table 5.5 shows the results of running the four solvers to find the minimum energy consumption at different utilization levels. If the solver could not find the results in less than 100 seconds, it is kept running until 10000 seconds to see how close the solver can get to the optimal value.

Table 5.5: Energy Consumption at Different Utilization Levels and Running Times.

Inst. No.	Util. level (Tasks/ No. of Cores)	Energy Consumption									
		NaPS			Minisat+			Sat4j			CPLEX
		100s	1000s	10000s	100s	1000s	10000s	100s	1000s	10000s	Value/time(s)
1	16.67%	194	-	-	194	-	-	194	-	-	194 / 0.32
2	33.33%	537	514	-	967	514	-	514	514	514	514 / 0.39
3	50.00%	1457	1457	1457	1558	1364	1203	1122	1118	1116	1102 / 0.53
4	66.67%	2522	2522	2522	2308	2130	2013	1844	1838	1834	1818 / 0.58
5	83.33%	3202	3202	3202	3936	3596	3411	3162	3146	3112	2958 / 0.68
6	100.0%	6102	6102	6102	5292	4900	4424	4028	4028	4028	4000 / 0.82

From the results we notice that increasing the utilization of the datacenter increases the minimum energy significantly as well as the time taken to find the minimum energy as illustrated in Figure 5.5.

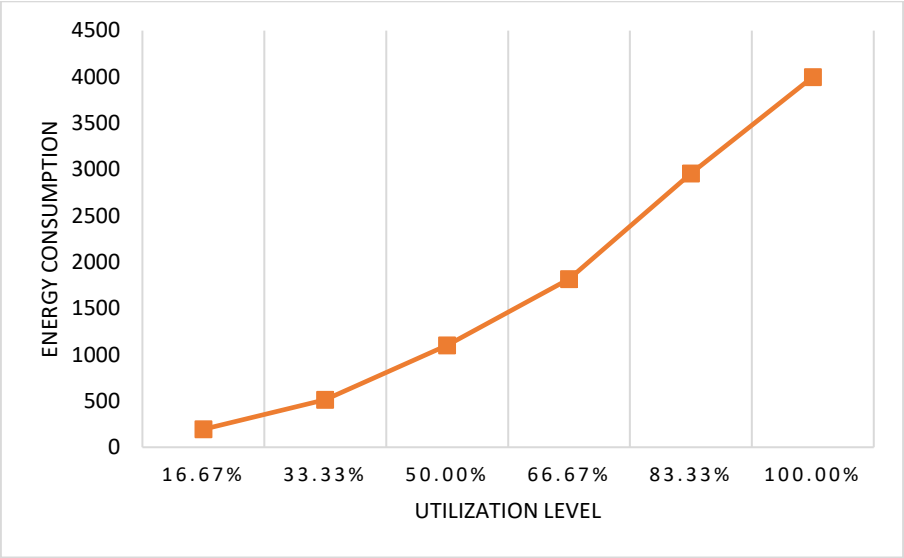

Figure 5.5: Test 5 Results - The energy consumption of a datacentre at different utilization levels.

The results of the SAT solvers show that if given time, Minisat+ and Sat4j return better results than NaPS. Comparing the results across all the different SAT solvers running times, we observe that Sat4j always returned the least amount of energy consumption. Looking at the results of Sat4j alone, the improvement of the results between 100 seconds and 10000 seconds is not significant. The best improvement in Sat4j results was 1.69% of the minimum energy which does not justify the extra running time. This indicates that it is possible to sacrifice the accuracy of the result for

the sake of less solving time. The drawback of using Sat4j in this test was that the solver did not terminate the search even after finding the minimum possible energy consumption (514) for instance 2, while the other two solvers found the same minimum and terminated in less than 1000 seconds. Furthermore, we notice that NaPS returned the same result after 100, 1000, and 10000 seconds. This result is actually found within the first few seconds. The solver then continues to look for better solutions but did not return any in this case, on the other hand, the other two PB-SAT solvers kept returning better solutions while searching for an optimum solution. CPLEX was able to find the minimum energy consumption in all the cases in less than 1 second. It is fair to conclude that the SAT solvers were getting closer to the minimum value and would have reached it if given enough time and memory.

5.6. Test 6: Results and Evaluation

This section presents and evaluates the results of Test 6 described in Section 4.6. From the results in Table 5.6 we observe that the proposed model succeeded in finding the minimum power consumption in all the cases. In configuration 3, Round Robin gave the same result as the proposed model because the solution was to use the first server in both cases. But, as the datacentre setup grow in size, the difference between the proposed model performance when compared with the other two approaches become more evident as illustrated in Figure 5.6. On average, the proposed model saved approximately 72% of the maximum power consumption with a maximum saving of 75.6% for configuration 3.

Table 5.6: The power consumption found with Maximum Possible Value, Round Robin, and the proposed model using real benchmarks.

Configuration	**No. of servers**	**Total no. of CPUs**	**Total no. of cores**	**No. of Tasks**	**Power Consumption**			**CPU Time (seconds)**
					Max Cons.	**Round Robin**	**Proposed Model**	
Config. 1	2	3	8	2	288	220	108	0.046
Config. 2	2	4	12	4	371	316	104	0.078
Config. 3	3	5	16	6	492	120	120	0.109
Config. 4	6	13	44	10	976	766	244	1.781
Config. 5	8	20	72	20	1268	876	320	16.66

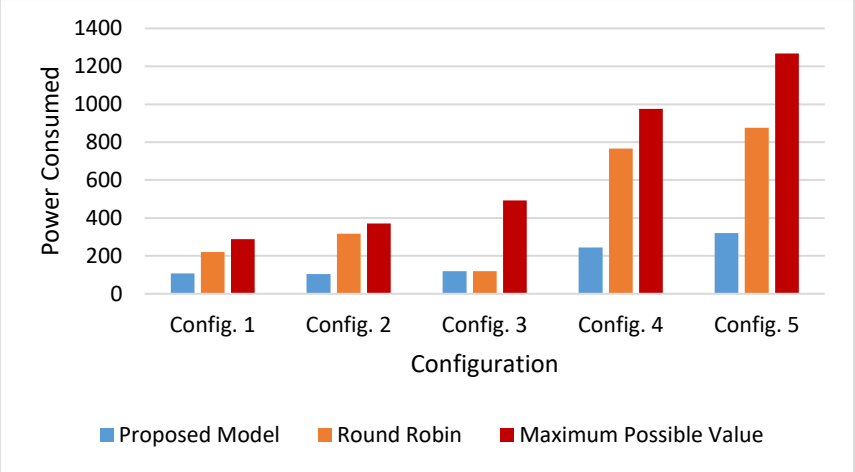

Figure 5.6: Test 6 Results - Power consumption values of the tested methods

5.7. Test 7: Results and Evaluation

In this section, the results of Test 7 specified in Section 4.7 are reported and analysed. The objective of this test is to study the effect of introducing the task frequency and RAM requirements to the model. Table 5.7 presents the results of testing the model after adding the frequency and RAM requirements. Initially, only NaPS is used to test the model with the results given in Table 5.7.

Table 5.7: Testing the utilization of a datacenter with Frequency and RAM requirements added.

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks	Total No. of Instructions of all Tasks	Min Energy Consumption NaPS	CPU Time (seconds)
1	6	12	36	6	20500	566	0.437
2	6	12	36	12	49500	990	0.984
3	6	12	36	18	71000	1468	5.781
4	6	12	36	24	100500	UNSAT	-
5	6	12	36	30	135500	UNSAT	-
6	6	12	36	36	170000	UNSAT	-

For the first three instances NaPS found an optimal assignment of tasks to servers that can be executed with minimum energy in less than 6 seconds. On the other hand, instances 4, 5, and 6 were all redeemed unsatisfiable by the solver. This implies that there is no possible solution that will assign the tasks to the servers regardless of how much energy is to be consumed.

Tracking the generated input instances (instance 4 as an example), the following conflicts are identified:

- The datacenter has 3 servers with 2400 MHz, 2 servers with 2600 MHz, and only one servers with 3000 MHz frequency.
- The server with 3000 MHz is a dual-core server with 2 CPUs; a total of 4 cores in the server.
- Until instance 3 the total number of tasks were 18 tasks. Among which there are 4 tasks that require a frequency of 2800 MHz or more and thus can be executed in only one server (i.e 3000 MHz server).
- Hence, starting from instance 4, any newly added task that require more than 2800 MHz cannot be assigned.

To avoid such conflicts in the generated instances the following changes are made:

- Previously the tasks requirements were chosen randomly independent of the task's size. This sometimes leads to smaller tasks requesting higher capabilities in memory and frequency requirement and therefore, wasting the datacenter resources. Thus, in the following test, tasks' requirements are matched to the tasks' sizes. The larger the task, the higher the requested capabilities in RAM size and Frequency.
- Another issue was in the set of real-life servers that are used to create the datacenter configuration. The server with the highest available frequency in that set is a dual-core server with 2 CPUs and a total of 4 cores. This server runs at 3000 MHz frequency with a total of 8 GB of RAM (2 GB per core). The cores of this server are not able to execute large size tasks even though they have high frequency due to their small RAM size. To work around this issue a new set of servers is generated as explained below:
    - Dual-core server with 2400 MHz and 2 GB of RAM per core.
    - Dual-core server with 3000 MHz and 4 GB of RAM per core.
    - Quad-core server with 2400 MHz and 2 GB of RAM per core.

- Quad-core server with 2600 MHz and 4 GB of RAM per core.

All servers are assumed to have two CPUs. The datacenter configuration is randomly created by selecting six of these servers. The number of tasks is next increased as before starting from 6 tasks up to 30 tasks. Table 5.8 present the reported results.

Table 5.8: Testing the utilization of a datacenter with an attempt to avoid conflicts in task requirements and available resources.

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks	Total No. of Instructions of all Tasks	Min Energy Consumption NaPS	CPU Time (seconds)
1	6	12	36	6	20500	229	0.812
2	6	12	36	12	49500	495	2.671
3	6	12	36	18	71000	734	25.91
4	6	12	36	24	100500	1005	43.8
5	6	12	36	30	135500	UNSAT	-

The results show that the modifications made has improved the satisfiability rate for the different instances. The solver returned lesser energy consumption values. It took more time to find the results as the number of tasks increased. This reflects that the solver had to work through an increased number of viable solutions (less conflicts and restrictions between requirements and resources) to find the minimum between them. At Instance 5, the number of tasks (30 tasks) is very close to the total number of cores (36 cores) in the datacenter, which resulted in another case of more requested capabilities than available resource. This can be attributed to the fact that in this work we assumed limited number of servers with high capabilities. Once assigned these servers are unusable. However, in real life even though the resources maybe limited in some cases, but they can always be utilized again after completing a task to execute another one. To compensate for this situation, we assume that the number of tasks can only reach a maximum of 50% of the total number of cores in the datacenter. Thus, making sure that there are always enough resources to execute the tasks and limit the question to only which of these resources will execute them with minimum energy consumption. Table 5.9 shows the result of testing different datacenter instances based on the aforementioned assumption. Results indicate that the assumptions we made ensured the assignment of tasks to cores without any conflict.

Table 5.9: Results of the model with task number limited to half the core number.

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks	Total Tasks Size (#instructions)	Min Energy Consumption NaPS	CPU Time In seconds
1	2	4	12	6	30000	326	0.203
2	3	6	20	10	44500	469	0.734
3	4	8	24	12	41500	415	1.109
4	5	10	28	14	55500	579	1.828
5	7	14	44	22	84500	869	102.8

As expected the CPU time taken by the solver increased with the increase in problem size but in all the cases the minimum energy consumption is found. It is also worth mentioning that the last instance (instance 5) was solved in around 100 seconds even though the solver timed out while trying to solve less sized instances (Instance 3 in both Tables 4.3 and 5.5 for example). Hence, adding tasks requirements while keeping in mind the correlation between the task size and its requirement and dealing with the limited resources assumption, reduced the search space of the problem all the while making the model more realistic.

5.8. Test 8: Results and Evaluation

This section details the results of Test 8 specified in Section 4.8. The objective of this test is to compare the performance of the solvers using the full model. Table 5.10 shows the results of the fully updated model using all the solvers. The Min Energy Consumption column lists the minimum energy found by the solvers. While Sat4j failed to find the optimum solution starting from instance 3, CPLEX, NaPS, and Minisat+ found the optimum solution in all the cases that were tested except for the last case (Instance 6). In instance 6, NaPS and Minisat+ ran for more than 10,000 seconds without finding the minimum energy. The CPU time taken by NaPS and Minisat+ increased dramatically from instance 4 to 5 to 6. The time taken by CPLEX was almost always better than the time taken by NaPS and Minisat+ to find the minimum energy as can be observer from Figure 5.7.

Table 5.10: Further tests using all the solvers.

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks	Total Inst. of all Tasks	Min Energy Consumption	CPU Time (seconds)			
							Sat4j	Minisat+	NaPS	CPLEX
1	2	4	12	6	25500	281	1.206	**0.136**	0.171	0.33
2	3	6	20	10	41000	434	-/0.353	0.508	0.984	**0.33**
3	4	8	24	12	47000	470	-/325.9	1.264	1.5	**0.42**
4	5	10	28	14	54500	569	-	3.632	3.593	**0.50**
5	7	14	44	22	87000	894	-	101.52	56.56	**0.99**
6	10	20	68	34	128500	1309	-	>4000 (1309 at 1440 s)	>4,000 (1467 at 2.1 s)	**2.59**

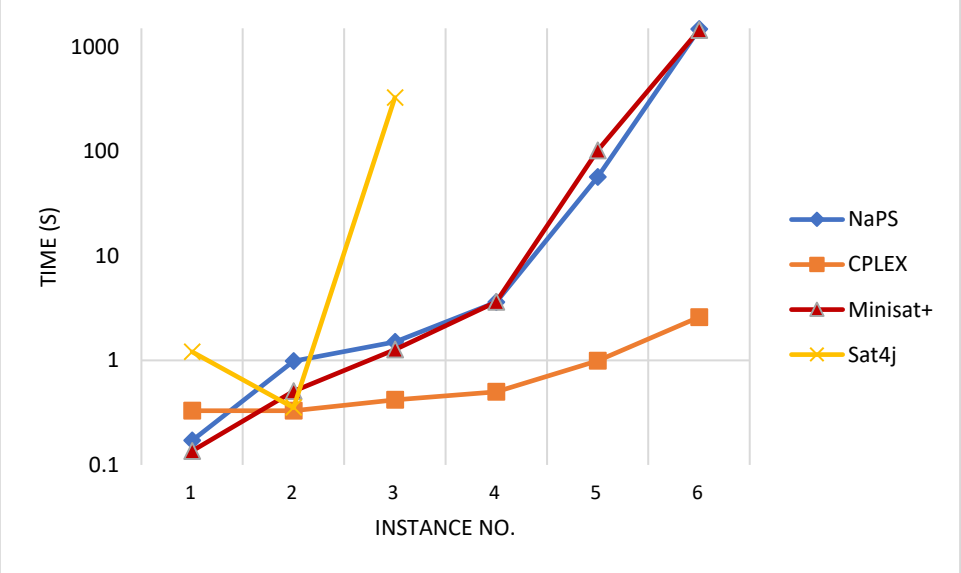

Figure 5.7: The CPU time taken by the solvers to find the minimum energy consumption.

The difference became clear in the last instance when NaPS was not able to find the optimal solution while CPLEX found it in 2.6 seconds only. It is worth mentioning that the result found by NaPS for instance 6 (1467) is only 12% more than the minimum energy (1309). This result was found in 2.1 seconds after running NaPS. The solution provided by NaPS is close to the optimal solution, but CPLEX found this

optimal solution in 2.6 seconds. This means at least for instances of this size running NaPS as a heuristic solver is not justified.

5.9. Test 9: Results and Evaluation

In this section the results of Test 9 specified in Section 4.9 are presented and discussed. The objective of this test is to examine the effect of adding the High and Low execution modes to the model. In the first part of the test, the PB-SAT solvers and CPLEX solved the testing instances to find the minimum energy consumption and the results are presented in Table 5.11. All solvers managed to find the minimum energy consumption in the first six instances.

Table 5.11: The minimum energy consumption and CPU time after adding High and Low execution modes.

Instance No.	**Total no. of inst. of all Tasks**	**Min Energy Consumption**	**CPU Time (seconds)**			
			NaPS	**Minisat+**	**Sat4j**	**CPLEX**
1	7,500	151	1.05	**0.05**	0.18	0.33
2	25,500	536	2.63	1.94	7.79	**0.35**
3	23,000	488	9.86	15.31	13.71	**0.33**
4	41,000	690	44.0	184.74	151.68	**0.53**
5	29,500	516	65.09	140.63	150.36	**0.51**
6	47,000	730	215.9	705.13	68.21	**0.68**
7	68,500	1101	> 1000 (1108)	> 1000 (1261)	> 1000 (1143)	**3.02**
8	98,500	1567	> 1000 (2080)	> 1000 (1833)	> 1000 (1756)	**4.96**

The CPU times for the PB-SAT solvers are relatively close to each other, with NaPS taking the lead in most of the instances. However, if we include CPLEX in the comparison we can see that except in instance 1 it was always faster than the other solvers with a big difference as can be seen in Figure 5.8. In the last two instances (7,8) the PB-SAT solvers are stopped after exceeding the 1000 second mark. The values between the brackets are the best solution found by the solvers up to that point in time. Only CPLEX have successfully found the minimum energy consumption in those instances and within less than 5 seconds. From Figure 5.9 we notice that the minimum energy consumption is directly related to the total number of instructions of the tasks to be scheduled. The time taken by NaPS is proportional to the total number of instructions as well.

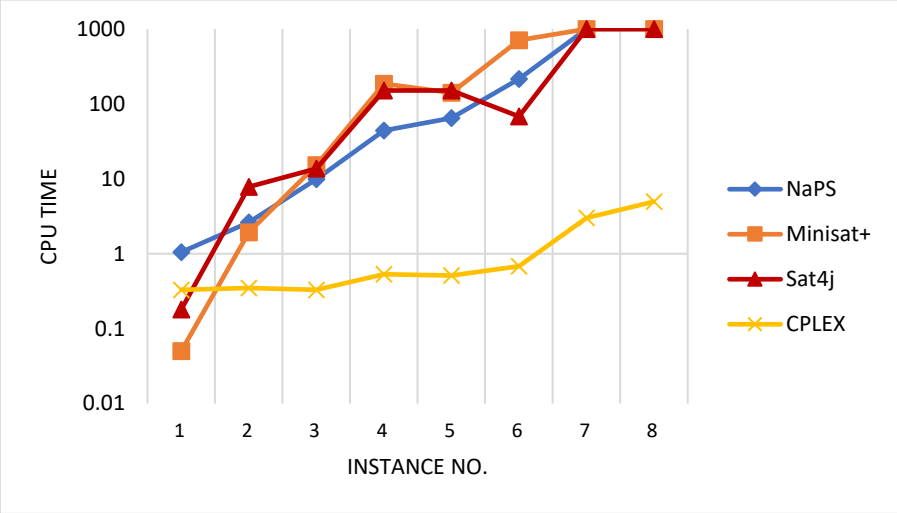

Figure 5.8: The performance of the PB-SAT solvers and CPLEX in Test 9

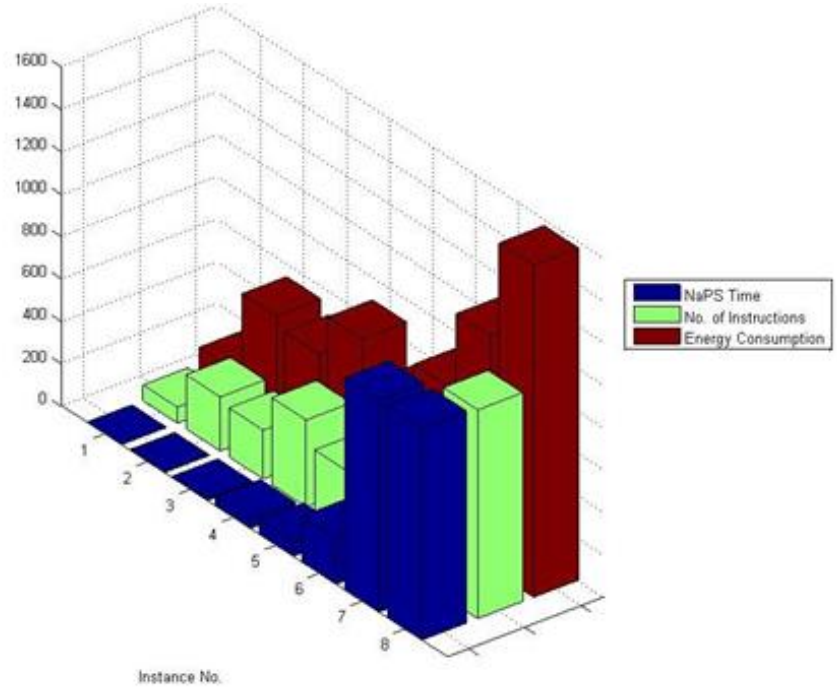

Figure 5.9: 3D Illustration of Test 9 results.

Table 5.12 shows the results of the second part of the test where the proposed model is compared with Round Robin and Maximum Possible Value. We observe that overall the proposed model has consistently returned the minimum energy

consumption when compared with the other two methods. The average amount of energy saved is 27.3% of the maximum energy consumption with a maximum of 35.6% energy saved in instance 7. We also notice that Round Robin method failed to schedule two of the instances (4 and 6). This can be attributed to the fact that Round Robin method schedules the tasks to the first available core that can execute these tasks even if the task is small and can be executed on some other lower capability core. This leads to larger tasks not finding any core that can be assigned to them because all the capable cores are executing smaller tasks and that exactly what happened in both instances 4 and 6. The only way Round Robin can schedule the tasks of instances 4 and 6 is by waiting until one of the busy cores finishes its task. Hence, in all cases the proposed model proved its superiority. Figure 5.10 is a graphical representation of the results of Table 5.12.

Table 5.12: Comparison of the proposed model with Round Robin and Maximum Possible Value methods.

Instance No.	Total Tasks Size	Energy Consumption		
		Max Cons.	Round Robin	Proposed Model
1	7,500	233	189	151
2	25,500	584	536	536
3	23,000	599	491	488
4	41,000	939	UNSAT	690
5	29,500	742	560	516
6	47,000	1081	UNSAT	730
7	68,500	1710	1383	1101
8	98,500	2277	1830	1567

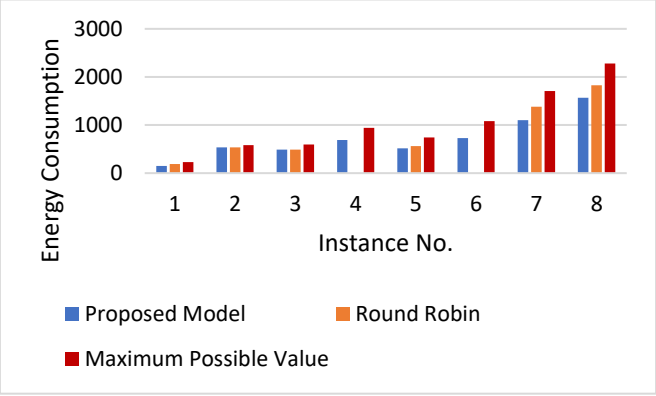

Figure 5.10: Energy Consumption of the proposed model, Round Robin, and Maximum Possible Value methods

5.9. Test 10: Results and Evaluation

This section presents and evaluate the results of scheduling a second batch of tasks as explained in Section 4.9. The objective of this test is to validate the modified model that schedules two batches of tasks at t_0 and t_1. Table 5.13 shows the results of the modified model. Contents of the fifth column (No. of Tasks at t_0) are the number of tasks in the first batch and the sixth column (Min Energy at t_0) is the energy consumption value of executing all the tasks in batch t_0. The seventh column (No. of remaining tasks at t_1) refers to how many tasks from batch t_0 are still running when the second batch of tasks arrived at t_1. Next, the eighth column (No. of new tasks at t_1) is how many new tasks arrived at t_1. The minimum energy consumption at t_1 is the total of the energy consumed by the tasks that are still running from t_0 plus the energy consumed by the new tasks at t_1 and is listed under the ninth column.

Table 5.13: The results of scheduling a second batch of tasks at t1.

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks at t_0	Min Energy at t_0	CPU time in seconds at t_0	No. of remaining Tasks at t_1	No. of new Tasks at t_1	Min Energy at t_1	CPU time in seconds at t_1
1	2	3	8	2	24	0.015	1	2	32	0.015
2	2	4	12	4	40	0.015	2	3	97	0.031
3	3	6	20	6	92	0.078	2	6	108	0.078
4	6	18	68	10	80	0.562	4	8	96	0.515
5	8	24	92	20	184	10.17	8	16	224	3.921

From the results we see that the modification made to the original model have successfully allowed scheduling a second batch of tasks at t_1 while minimizing the overall energy consumption of the datacenter. We also observe that scheduling the second batch of tasks took less time in instance 4 and 5 because some cores were already busy running the remaining tasks from batch t_0, and hence were removed from the pool of available cores. Further testing is carried out to compare the proposed modified model to Round Robin algorithm in scheduling a second batch of tasks. The results are tabulated in Table 5.14. In the small instances (instances 1 and 2), Round Robin and the proposed model results are close and even equal in some cases, but as the number of servers, CPUs, and cores increase, the proposed model results became better compared to Round Robin. Considering the largest instance tested (instance 5) we observe that Round Robin results are 235.7% worse than the proposed model. The

difference between the two algorithms' results at t_0 and t_1 is illustrated in Figures 5.11 and 5.12.

Table 5.14: Comparing Round Robin and the proposed model in scheduling a second batch of tasks.

Instance No.	No. of servers	Total no. of CPUs	Total no. of cores	No. of Tasks at t_0	Min Energy at t_0 Proposed model	Min Energy at t_0 RR	No of remaining Tasks at t_1	No of new Tasks at t_1	Min Energy at t_1 Proposed model	Min Energy at t_1 RR
1	2	3	8	2	24	24	1	2	32	32
2	2	4	12	4	40	40	2	3	97	119
3	3	6	20	6	92	134	2	6	108	164
4	6	18	68	10	80	226	4	8	96	228
5	8	24	92	20	184	288	8	16	224	528

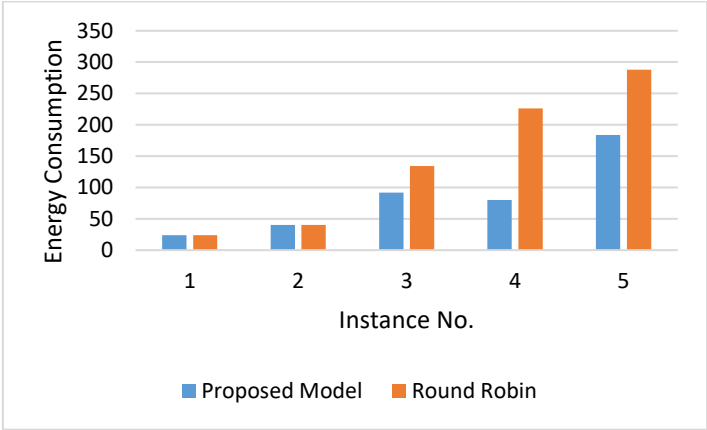

Figure 5.11: Energy Consumption after scheduling the task of t0

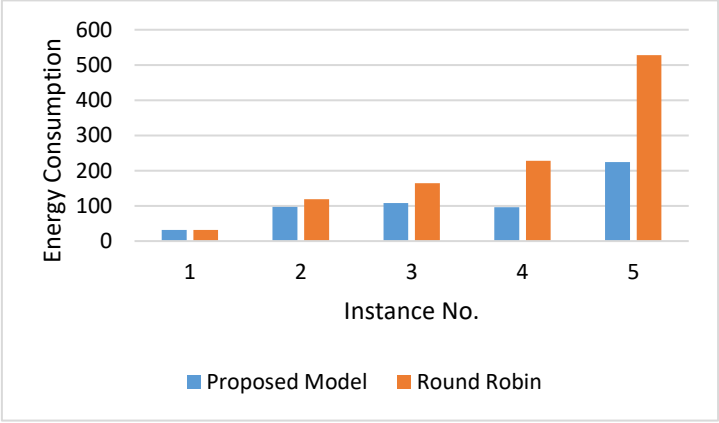

Figure 5.12: Energy Consumption after scheduling the task of t1.

Chapter 6. Conclusion and Future Work

Despite the fact that cloud computing has emerged as a very viable approach with huge opportunities for the IT industry, there are still critical technical issues pertaining to its successful use that need to be addressed. At the forefront of these issues is the power consumption of its datacentres

In this work, the energy consumption of a datacentre is minimized using energy efficient tasks scheduling and allocation of resources. We proposed a framework to tackle the energy efficiency in the context of a datacentre operation through energy conscious scheduling and reduction in the number of active servers. The resources considered in this research are the computing resources only. We developed ILP optimization models for the task allocation and scheduling problem in a datacentre with the objective of assigning the tasks to the resources that would execute them with the least amount of energy consumption. The basic idea is to use an optimum number of servers to execute the tasks with the unused servers being turned off subject to the satisfaction of number of constraints. A base model is validated and tested without including any task characteristics. It succeeded in returning the minimum energy consumption in all the tested scenarios. The model is also compared with Round Robin and Maximum Possible Value methods, using real benchmark values, in both cases the proposed model outperformed the others.

The base mode is incrementally enhanced and tested leading to a more comprehensive model that includes task characteristics such as: task's size, task's required frequency, and task's required RAM included in its formation. The model was still able to find the minimum energy consumption in most of the tested cases. Finally, the proposed model was modified to handle the arrival and scheduling of a second batch of tasks after the initial batch is already assigned to various servers in the datacentre. The performance of this model was compared to Round Robin algorithm, and it was evident that the modified model was able to allocate the two batches of tasks with less energy consumption than Round Robin.

Various datacenters configurations are studied using different scenarios. In each scenario a set of resources (Servers, CPUs, and Cores) is generated with a certain number of tasks to be allocated using a version of the proposed model. The results of

these scenarios showed that the proposed base model in some cases can save up to 82.3% in energy when compared to the maximum possible energy consumption and up to 37.9% when compared against the energy consumption of Round Robin. Furthermore, we found that increasing the number of servers under certain constraints yields less energy consumption when executing the same number of tasks. In addition, the energy consumption of a datacenter was found as expected to be proportional to the utilization level of the datacenter, that is, as the number of active core increases the energy consumption as well increases. Upon testing the complete model and comparing it with Round Robin and Maximum Possible Value methods, results demonstrated that the proposed model can save up to 35.6% of the energy consumption in some cases when compared to Maximum Possible Value method. Furthermore, while Round Robin algorithm was not always successful in finding a solution for allocating all the tasks, still the proposed model outperformed it in the few cases where it did find a solution.

Another contribution in this work involved using the proposed ILP formulations to test the performance of three PB-SAT solvers namely: NaPS, Minisat+ and Sat4j, and a generic ILP solver named CPLEX in solving the energy optimization problem for datacentres. The performance of the three PB-SAT solvers was relatively close to each other, and they were able to find the optimum energy for most of the tested instances. However, as the size of the search space increased the PB-SAT solvers started to time-out. Therefore, for PB-SAT solvers the time needed to explore the search space makes them impractical for finding solutions for large scale datacentres. On the other hand, the generic ILP-solver CPLEX outperformed all the PB-SAT solvers and was able to handle relatively large datacentres configurations and find the optimal solution (the minimum energy consumption) in a shorter amount of time.

Potential future directions of this work include:

- Extending the ILP models developed in this work to include scheduling of network and storage resources.
- Extending the proposed model to have a multi-objective function such as reliability and availability of services while optimizing energy consumption.

- Explore the use of ILP in modelling the energy problem but in virtualized datacentres contexts.

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and 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, no. 6, pp. 599-616, June 2009.

[2] P. Mell and T. Grance. "The NIST definition of cloud computing". *Communications of the ACM*, vol. 53, no. 6, p. 50, December 2010.

[3] M. Dayarathna, Y. Wen, and R. Fan. "Data center energy consumption modeling: A survey". *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732-794, 2016.

[4] L. Wang, and E. Gelenbe. "Experiments with smart workload allocation to cloud servers". In *Proc. of IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*, 2015, pp. 31-35.

[5] C. Kim and H. Kameda. "An algorithm for optimal static load balancing in distributed computer systems". *IEEE Transactions on Computers*, vol. 41, no. 3, pp. 381-384, March 1992.

[6] B. Priya, E.S. Pilli, and R.C. Joshi. "A survey on energy and power consumption models for Greener Cloud". In *Proc. of IEEE 3rd International Advance Computing Conference (IACC)*, 2013, pp. 76-82.

[7] S.K. Garg and R. Buyya. "Green cloud computing and environmental sustainability". *Harnessing Green IT: Principles and Practices*, pp. 315-340, August 2012.

[8] M.L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[9] S. Aslam and M.A. Shah. "Load balancing algorithms in cloud computing: A survey of modern techniques". In *Proc. of National Software Engineering Conference (NSEC)*, 2015, pp. 30-35.

[10] R. Kaur and P. Luthra. "Load balancing in cloud computing". In *Proc. of International Conference on Recent Trends in Information, Telecommunication and Computing (ITC)*, 2012, pp. 374-381.

[11] K.B. Bey, F. Benhammadi, F. Sebbak, and M. Mataoui. "New tasks scheduling strategy for resources allocation in cloud computing environment". In *Proc. of the 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, 2015, pp. 1-5.

[12] 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, no. 1, pp. 59-64, 2013.

[13] D. Agarwal and S. Jain. "Efficient optimal algorithm of task scheduling in cloud computing environment". *International Journal of Computer Trends and Technology*, vol. 9, no. 7, pp. 344-349, 2014.

[14] 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, January 2014.

[15] R. Buyya, C. Vecchiola, and S.T. Selvi. *Mastering cloud computing: foundations and applications programming*. Newnes, 2013.

[16] M. Poess and R.O. Nambiar. "Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results". In *Proc. of the VLDB Endowment*, vol. 1, no. 2, pp.1229-1240, August 2008.

[17] R. Aburukba, H. Ghenniwa, and W. Shen. "Agent-based approach for dynamic scheduling in content-based networks". In *Proc. of the IEEE International Conference on e-Business Engineering (ICEBE)*, 2006, pp. 425-432.

[18] A. Biere, M. Heule, and H. van Maaren. *Handbook of satisfiability.* IOS press, 2009.

[19] R. Ge, X. Feng, and K.W. Cameron. "Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters". In *Proc. of the ACM/IEEE SC 2005 Supercomputing Conference*, 2005. pp. 34-34.

[20] S. Roy, A. Rudra, and A. Verma. "An energy complexity model for algorithms." In *Proc. of the 4th conference on Innovations in Theoretical Computer Science* 2013, pp. 283-304.

[21] A.W. Lewis, S. Ghosh, and N.F. Tzeng. "Run-time Energy Consumption Estimation Based on Workload in Server Systems". *HotPower*, vol. 8, pp. 17-21, December 2008.

[22] X. Fan, W.D. Weber, and L.A. Barroso. "Power provisioning for a warehouse-sized computer". In *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 13-23, June 2007.

[23] Y. Li, Y. Wang, B. Yin, and L. Guan. "An online power metering model for cloud environment". In *Proc. of the 11th IEEE International Symposium on Network Computing and Applications (NCA)*, 2012, pp. 175-180.

[24] N. Liu, Z. Dong, and R. Rojas-Cessa. "Task scheduling and server provisioning for energy-efficient cloud-computing data centers". In *Proc. of the IEEE 33rd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2013, pp. 226-231.

[25] A. Dambreville, J. Tomasik, J. Cohen, and F. Dufoulon. "Load Prediction for Energy-Aware Scheduling for Cloud Computing Platforms". In *Proc. of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2604-2607.

[26] C.M. Wu, 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.

[27] E.M. Elnozahy, M. Kistler, and R. Rajamony. "Energy-efficient server clusters". In *International Workshop on Power-Aware Computer Systems,* 2002, pp. 179-197.

[28] H. Tian, J. Wu, and H. Shen. "Efficient Algorithms for VM Placement in Cloud Data Centers". In *Proc. of the 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT),* 2017, pp. 75-80.

[29] X. Ye, Y. Yin, and L. Lan. "Energy-Efficient Many-Objective Virtual Machine Placement Optimization in a Cloud Computing Environment". *IEEE Access,* vol. 5, pp.16006-16020, 2017.

[30] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[31] S. Malik and L. Zhang. "Boolean satisfiability from theoretical hardness to practical success". *Communications of the ACM,* vol. 52, no. 8, pp. 76-82, August 2009.

[32] F. Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah. "Generic ILP versus specialized 0-1 ILP: An update". In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design,* 2002, pp. 450-457.

[33] C.P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. "Satisfiability solvers". *Foundations of Artificial Intelligence,* vol. 3, pp. 89-134, 2008.

[34] A. Sagahyroon, and F. Aloul. "Using SAT-based techniques in power estimation". *Microelectronics Journal*, vol. 38, no. 6-7, pp. 706-715, June 2007.

[35] IBM. "IBM ILOG CPLEX Optimizer". Internet:http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/ [Jun. 8, 2018].

[36] 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, 2017 [Jun. 8, 2018].

[37] M. Sakai and H. Nabeshima. "Construction of an ROBDD for a PB-Constraint in Band Form and Related Techniques for PB-Solvers". *IEICE Transactions on Information and Systems,* vol. 98, no. 6, pp. 1121-1127, June 2015.

[38] Pseudo-Boolean Competition. Internet:http://www.cril.univ-artois.fr/PB16/, 2016 [Jun. 8, 2018].

[39] N. Eén and N. Sörensson. "An extensible SAT-solver". In *International Conference on Theory and Applications of Satisfiability Testing,* 2003, pp. 502-518.

[40] D. Le Berre and A. Parrain. "The sat4j library, release 2.2, system description". *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, pp. 59-64, 2010.

[41] R. Basmadjian, N. Ali, F. Niedermeier, H. De Meer, and G. Giuliani. "A methodology to predict the power consumption of servers in data centres".

In *Proc. of the 2nd international conference on energy-efficient computing and networking*, 2011, pp. 1-10.

[42] Standard Performance Evaluation Corporation Internet: http://www.spec.org/power_ssj2008/results/index.html, Apr. 25, 2018 [Jun. 8, 2018].

Vita

Ahmed Osman was born in 1992, in Sana’a, Yemen. He received his primary and secondary education in Khartoum, Sudan. He received his B.Sc. degree in Electrical and Electronic Engineering from the University of Khartoum in 2014. From 2015 to 2016, he worked as a Computer Engineer in Badr Research Center.

In February 2016, he joined the Computer Engineering master's program in the American University of Sharjah as a graduate teaching assistant. During his master's study, he co-authored 2 papers which were presented in national conferences. His research interests are in (Cloud computing, SAT solvers, and Data mining).