Performance-Predictable Resource Management of Container-based Genetic Algorithm Workloads in Cloud Infrastructure

Thamer Alrefai

Doctor of Philosophy

University of York

Computer Science

October, 2021

Abstract

Cloud computing, adopted by major providers like Amazon and Google, offers on-demand, pay-as-you-go services and resources through shared pools. Users submit workloads comprising multiple jobs, each containing tasks, including a specific genetic algorithm (GA) workload detailed in this thesis. This GA workload contains independent tasks from real-time multiprocessor allocation and Sudoku puzzle case studies, each with fixed deadlines and fitness requirements. Effective resource management is critical to enhance the Quality of Service (QoS) for cloud users. It involves resource allocation and adhering to QoS standards, guided by workload specifics. Container orchestration emerges as an essential deployment and management approach.

This thesis focuses on managing multiple instances of genetic algorithms (GAs) in a cloud environment to achieve user-defined fitness levels within specified deadlines. It presents various approaches to allocate GAs to cloud nodes and control their execution iteratively. Initially, it introduces approaches such as fitness tracking (FT), fitness prediction (FP), fitness-prediction-based linear regression (FPLR), and fitness prediction based on weighted least square (FP-WLS) for managing the workload. To enhance resource efficiency, the thesis also addresses node interference, allowing multiple tasks to share resources while minimizing their impact on each other. It proposes a weighted-based node interference approach, considering fitness levels and response times during iterations to optimize task allocation.

The performance of these approaches was experimentally evaluated by testing two GA applications and comparing them against state-of-the-art container-based orchestration approaches. Thus, different approaches were compared considering the number of successful tasks which can be defined by the number of tasks executed on time and achieved the fitness required. Comparison was also made between different approaches by taking iteration analysis into consideration. In situations where performance prediction was used, prediction errors like Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) were used to evaluate and compare the performance of the prediction approaches.

Contents

Abstract **iii**

List of Figures **vii**

List of Tables **ix**

Acknowledgments **x**

Dedication **xi**

Declaration **xii**

Publications **xiii**

1 Introduction **1**

1.1 Motivation 5

1.2 Use-cases for managing GA workload 7

1.3 Research Problems 8

1.3.1 The problem of managing the GA workload in container-based technologies 8

1.3.2 The problem of node interference in container-based technologies 9

1.4 Research hypothesis 9

1.5 Thesis contributions 10

1.6 Thesis outline 10

1.7 Summary 12

2 Background **14**

2.1 Cloud computing 14

2.1.1 Cloud computing characteristics 16

2.1.2 Cloud computing service models 17

2.1.3 Cloud computing deployment models 17

2.2 Resource management in cloud computing 18

2.2.1 Resource allocation and scheduling 19
2.2.2 Resource monitoring 20
2.2.3 Elasticity 21
2.3 Container-based technologies 22
2.4 Optimisation methods 25
2.5 Summary 27

3 Literature Survey 28
3.1 System model 28
3.2 Workload model 29
3.2.1 Application characteristics 30
3.2.2 Workload generator 32
3.3 Workload deployment 33
3.4 Workload profiling 36
3.5 Cloud monitoring 37
3.6 Performance prediction 39
3.6.1 Why do we need performance prediction? 40
3.6.2 Performance prediction approaches 41
3.6.3 Prediction evaluation metrics 44
3.7 Resource allocation and scheduling 45
3.8 Optimization problem models and solvers 50
3.9 Summary 51

4 Experimental platform, metrics and methods 52
4.1 Encapsulation of genetic algorithm 53
4.2 Application model 53
4.3 Proposed orchestrator 54
4.4 Metrics 55
4.5 Experimental method 56
4.6 Summary 58

5 Management of container-based genetic algorithm workloads over cloud infrastructure 59
5.1 Comparing different allocation techniques 60
5.1.1 Fitness tracking (FT): 60
5.1.2 Fitness prediction (FP): 61
5.1.3 Fitness-prediction-based linear regression (FPLR): 62
5.1.4 Fitness-prediction-based weighted least square curve fitting (FPWLS) 63

5.2 Evaluation 64
5.2.1 Experimental setup 64
5.2.2 Experimental results 65
5.3 Summary 79

6 Handling node interference in managing genetic algorithm workloads 80
6.1 Weight-based node interference approach 81
6.2 Admission control for firm real-time task 84
6.3 Evaluation 85
6.3.1 Experimental setup 85
6.3.2 Experimental results 86
6.4 Summary 98

7 Conclusions and Future Work 100
7.1 Future Work 102
7.1.1 Generalizations to different workloads 103

References 105

List of Figures

1.1 Technology stack 3
1.2 Docker Swarm architecture. 7

2.1 Characteristics of cloud computing and its models. 15
2.2 Example monitoring architecture. 21
2.3 Vertical scaling VS horizontal scaling. 22
2.4 Virtual machines VS Container-based technologies. 23
2.5 Example of a Dockerfile 24
2.6 GA process. 26

3.1 Allocate the workload to a VM 29
3.2 DAG shapes. 31
3.3 Timing Analysis. Extracted from [1]. 31
3.4 weather application as an example 32
3.5 Docker Swarm architecture. 34
3.6 Use Docker to deploy multiple instances of application 35
3.7 ATOM framework. Extracted from [2]. 39
3.8 Overview of the proposed methodology. Extracted from [3]. . . . 42
3.9 Priority-based resource allocation. Extracted from [4]. 46
3.10 Guaranteed admission control. Extracted from [5]. 47
3.11 Guaranteed admission control. Extracted from [5]. 47

4.1 Inputs and outputs of the GA. 53
4.2 Proposed orchestrator. 55

5.1 State machine of fitness tracking approach (FT). 61
5.2 Results of 10 experiments of different approaches and different GA workloads (real-time multiprocessor allocation) concerning the number of successful tasks. 66
5.3 Results of 10 experiments of different approaches and different GA workloads (Sudoku puzzle) concerning the number of successful tasks. 67

5.4 The number of tasks done and dropped in each iteration of the real-time multiprocessor allocation in FT and FP approaches. . 70
5.5 The number of tasks done and dropped in each iteration of the real-time multiprocessor allocation in FPLR and FPWLS. . . . 71
5.6 The number of tasks done and dropped in each iteration of the Sudoku puzzle in FT and FP approaches. 72
5.7 The number of tasks done and dropped in each iteration of the Sudoku puzzle. 73
5.8 Prediction errors of the real-time multiprocessor allocation. . . 75
5.9 Prediction errors of the Sudoku puzzle. 76
5.10 Delta fitness of the real-time multiprocessor allocation. 77
5.11 Delta fitness of the Sudoku puzzle. 78

6.1 Result of 10 experiments of different approaches and different GA workloads (real-time multiprocessor allocation) in terms of the number of successful tasks. 86
6.2 Result of 10 experiments of different approaches and different GA workloads (Sudoku puzzle) in terms of the number of successful tasks. 87
6.3 Number of tasks done and dropped in each iteration of the real-time multiprocessor allocation. 89
6.4 Number of tasks done and dropped in each iteration of Sudoku puzzle. 90
6.5 Result of 10 experiments of different approaches and different GA workloads (real-time multiprocessor allocation) in terms the CPU control in the *W* approach. 92
6.6 Result of 10 experiments of different approaches and different GA workloads (Sudoku puzzle) in terms the CPU control in the *W* approach. 93
6.7 Waiting time of tasks in real-time multiprocessor allocation. . . 94
6.9 CPU utilization of the *W* approach in real-time multiprocessor allocation 94
6.8 Waiting time of tasks in Sudoku puzzle. 95
6.10 CPU utilization of the *WCPU* approach in real-time multiprocessor allocation 95
6.11 CPU utilization of the *W* approach in Sudoku puzzle. 96
6.12 CPU utilization of the *WCPU* approach in Sudoku puzzle. . . . 97
6.13 Result of 10 experiments of different approaches and different GA workloads (real-time multiprocessor allocation) 98

List of Tables

3.1 Prediction results of task execution time for three different tasks. Extracted from [6]. 41

5.1 One-way ANOVA test comparing the FT, FP, FPLR and FP-WLS approach against Docker Swarm in real-time multiprocessor allocation case study. 68

5.2 One-way ANOVA test comparing the FT, FP, FPLR and FPWLS approach against Docker Swarm in Sudoku puzzle case study. . . 68

6.1 Example of data collection in each of the node while executing tasks. 82

6.2 One-way ANOVA test comparing the *W* approach against Docker Swarm, Kubernetes, and FTV1 in real-time multiprocessor allocation case study. 88

6.3 One-way ANOVA test comparing the *W* approach against Docker Swarm, Kubernetes, and FTV1 in Sudoku puzzle case study. . . 88

Acknowledgments

I would like to express my deepest gratitude and thanks to Almighty Allah for giving me the strength, blessing, and guidance during my PhD journey and finishing my research.

Dr. Leandro Soares Indrusiak, my PhD supervisor, deserves special thanks for his tremendous advice, constant support, and patience during my research. His enormous knowledge and vast experience have helped me throughout my academic life. I would like to express my gratitude to all of the members of the Real-Time Systems (RTS) research group. It is because of their generous assistance and support that my time in the UK has been enjoyable.

I am grateful to my country, the Kingdom of Saudi Arabia, for allowing me to complete my PhD in resource management in cloud computing to contribute to the country's development. Also, I would like to thank Taibah University for giving me the opportunity to continue my academic life.

I want to show my thankfulness to everyone who has helped and encouraged me to continue on my PhD journey to achieve my goal. These people are exceptional, and they have had a profound impact on both my personal and professional lives.

I would like to thank my parents, Mohammad Alrefai and Fatimah Alrefai, for their unconditional love and support, I shall be ever thankful. I will always be grateful for the tremendous sacrifices they made to ensure that I received an excellent education. Additionally, I want to thank my brothers and sisters for their love and support. Also, thanks to my daughters, Nour and Dalia, for being a great motivator in my life.

Dedication

This thesis is dedicated to my wife, Dr.Tahani Aljohani, who has been a continuous source of inspiration and support throughout my academic and daily life. I am grateful for your presence in my life. My deep love, gratitude, and respect for you.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References. Some of the materials in this thesis appeared in the following published paper (see page xiv).

Thamer Alrefai

October, 2021

Publications

1. T. Alrefai and L. S. Indrusiak, “Management of container-based genetic algorithm workloads over cloud infrastructure” *17th ACM International Conference on Computing Frontiers 2020, CF 2020 - Proceedings.* pp. 229–232, 2020.

Chapter 1

Introduction

Cloud computing is defined as an internet-based service that provides on-demand and pay-as-you-go resources [7]. These resources are based on different models, such as infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). These models can provide resources, including applications, platforms, and physical machines. Once the resources are available, a user can request to use them from a cloud provider [8]. The release of the resources is based on a service level agreement (SLA), which is a contract between the user, the provider, and quality of service (QoS), which is a performance measurement that needs to be met to fulfil the SLA requirements [9].

Application domains, such as healthcare and engineering, often need a computational workload to be executed within specified QoS requirements [10][11]. The outcome of the application typically bears financial value for the organisation or is a prerequisite of another activity that does. Such applications are typically executed on high-performance computing infrastructures or are costly resources concerning computer infrastructure, staff, and energy consumption. Therefore, effective resource management is necessary to make decisions about allocating computational resources that meet user-driven QoS requirements [12].

One specific type of load that often appears in engineering applications is optimisation. In many cases, optimisation software uses meta-heuristics such as genetic algorithms (GA) [13]. GAs are known for optimising solutions in various domains, such as smart factories [14] and embedded multiprocessors [15]. Example parameters that can be input into a GA are, the number of generations, the initial population, and the set of application-specific parameters used by its fitness evaluation function. We often need to achieve the desired fitness solution for a specific problem, with this fitness fulfilled by a given QoS requirement [5].

The resources are made available from the cloud provider where they are placed in a shared pool of resources, thus allowing users to select the most suitable resources to fulfil their requirements. As the demand for particular resources changes, a resource manager handles the change to fulfil the user's demand [16]. *Resource management* is therefore defined as the procedure for allocating resources, such as a physical machine, storage, and networking. Resource management is achieved by providing the user with ways to meet the QoS requirements. Resource management involves several important steps such as, monitoring the infrastructure resources, performance prediction, and value estimation of the resource [17].

Resource management plays an essential role in allocating resources in cloud computing. Therefore, a considerable amount of research looks at different areas within resource management and presents different ways of allocating resources [18]. One way of addressing resource management in different areas is the technology stack, which is a resource management example procedure that already exists in cloud computing (shown in Figure 1.1). The technology stack shown in the figure consists of several layers which includes a cloud or grid as the infrastructure layer, cloud monitoring, performance prediction, energy estimation, value estimation, and the resource allocation approach as the other layers. The information from the lower level is needed to allocate resources and meet user demands.

The benefit of having a technology stack is in the allocation of the right resources for executing the application. For example, if there is a slot of time available in the resources, the resource manager can use the slot to execute a task that can fit in the available space. Using the technology stack also assists in estimating when the task is most likely to finish and determining the required CPU and memory to execute the task. The technology stack can also help to control whether the task can have a financial value or not while the task is being executed. As the providers receive a request from the user, the financial value they get from executing the tasks starts at a maximum value. However, as the task reaches the deadline, its value decreases which results in a lower financial value in executing the task and or sometimes ends up paying more than needed. To help understand the motivation of the research presented in this thesis, it is needed to explain each layer of the technology stack. In the literature review chapter, each of these layers will be explained in more detail.

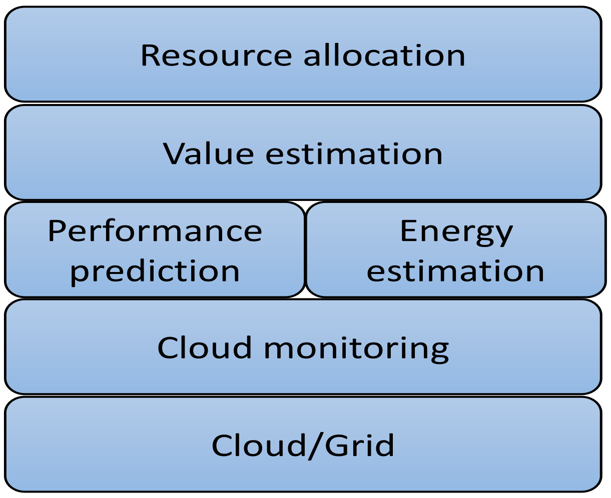


FIGURE 1.1: Technology stack

- Cloud monitoring:

While resources are provided by the cloud provider and change over time, the technology stack's monitoring layer helps with decision-making during resource management. As the cloud platform is large and complex, monitoring different layers of cloud computing is challenging due to the large amount of data collected and analysed, which can cause overhead in the system. Thus, in order to reduce the overhead caused by monitoring the resources, it is useful to determine the right monitoring metrics such as CPU and memory [19].

Studies considered monitoring resource use in their research to track the behaviour of the resources. Studies, such as [2], have tackled one of the issues within the monitoring area. Du and Li [2] used information collected from resources to detect any abnormal behaviour that could cause changes in resource usage and result in poor QoS. They considered CPU use as a metric. However, they also added additional metrics to help detect abnormal behaviour in the resources such as reading and writing operations per second and networkIn and networkOut in bytes (i.e., measure of amount of data moving across the network).

As the monitoring layer of the technology stack is responsible for collecting data from the infrastructure, the next layers which are performance prediction and energy estimation layers, use the information collected from the monitoring component. These layers help to determine whether the allocated resource is able to meet the user expectations based on the predicted value and the energy consumption [20].

- Performance prediction:

As mentioned earlier, the cloud user and the cloud provider are the two main actors in cloud computing. They both can benefit from performance prediction. Performance prediction is able to provide optimised solutions on cloud usage to maintain the QoS requirement. On the other hand, the cloud provider can also benefit from performance prediction layer as it prevents the resource manager from using extra resources or allocates fewer resources to the user.

Several related studies addressed performance prediction issues, including [6] and [3]. Both tried to predict application execution times but for different targets. Kim et al. [6] aimed to improve the response time and cost of resources. The input parameters of their model were job deadline and application workflow. Their approach was to use the past executed tasks to predict the execution time of new tasks using k-nearest neighbours (kNN) with similar samples. Mariani et al. [3] aimed to help the user to select the best cloud configuration for their application using the random forests (RFs) machine-learning approach. The RF approach is a collection of several regression trees where each tree is generated to fit the behaviour of the target performance metric using a randomly selected training data subset.

- Energy estimation:

Cloud computing involve large data centres requiring a high amount of energy to operate them. This problem increases the potential of carbon emission. Estimating energy consumption can help resource management to allocate resources by determining the right amount of resources without over-provisioning (providing more resources than needed) [21]. Both [22] and [23] estimated the energy value, and used an approach to reduce the energy use by merging VMs into one or using Dynamic Voltage and Frequency Scaling (DVFS) to reduce the amount of energy that a VM uses.

- Value estimation:

Value estimation from the technology stack will benefit from performance prediction and energy estimation since it can decide if the current task has value. Therefore, it can determine whether the process should continue to allocate resources to the task, as described in [23]. The value of a task tends to decrease towards zero as time increases. The more time it takes to allocate a task, the less is the value of the task to the user. Additionally, if the value is zero, the task should be rejected since there is no benefit from executing it.

- Resource allocation:

Once resource allocation has enough information from the lower level, it can allocate the new task to a suitable resource, considering the QoS and SLA. It is the cloud provider's responsibility to ensure that the right resources are allocated to the submitted task from the user.

Many researchers have considered QoS-based approaches in their work to determine whether a new task is schedulable which means that worst case response time of the task must be less than its deadline. Singh et al. highlighted several dynamic resource allocations [5]. One dynamic resource allocation is the guaranteed admission control approach, which considers two factors when allowing a task to be scheduled: task execution time and task deadline. Additionally, this approach ensures that all applications admitted into a system will meet their respective deadlines without interrupting other running applications. The worst-case execution time (WCET) of the application and its tasks must be less than the deadline for it to be schedulable using admission control. If so, then the application can be admitted for execution. Otherwise, the application is not schedulable, and the allocation will not proceed [5].

1.1 Motivation

The previous section introduces and explains the technology stack as an example of a resource management process in cloud computing. There remain areas of the technology stack that need further investigation. As data centres provide a large number of resources, managing these resources can be challenging. Thus, the benefit of using monitoring, is the ability to track and analyse the infrastructure to provide a good understanding of the resource status. Monitoring can also help detect abnormal behaviour, which may cause some resources to become unavailable [24]. Further, from the performance prediction perspective, different resource types and the complexity of the application pose a challenge to meet user expectations [25]. Additionally, the resource manager can provide a more accurate result during resource allocation by predicting the performance measurement. More so in the case of a change in resource demand. In that case, performance prediction can detect it and scale resources to the right amount to prevent any SLA violations, leading to a decrease in profits as providers lose their users [26].

As previously mentioned, resource management is responsible for executing the application sent from the cloud user. In a situation where a container orchestrator (Docker Swarm or Kubernetes) has been used to deploy and manage the execution of the application, the resource scheduling process is part of the orchestrator. It decides where to allocate the task. Thus, container-based orchestration systems allow applications to be executed in shared resources with fast and flexible deployment. One platform that can be used to deploy workloads is Docker. This platform is an abstraction to help organise the workload, hide the details, and deploy the application in an isolated environment [27]. Docker Swarm and Kubernetes are considered state-of-the-art container-based orchestration systems [28].

Providing resources for an incoming application is essential in obtaining the desired results based on QoS metrics. Therefore, providers use virtual machines (VMs) to execute user applications. Similar to VMs, container-based orchestration systems can be used for workload deployment. One feature of container-based systems is that the container uses the operating system's kernel and runs as an isolated process instead of using a hypervisor. This also is the case in VMs. Another feature is that the application is packed in an image for the user to execute. They can have as many versions of that application as needed. Further, container-based systems do not require fixed resources, such as CPU and memory. Instead, the container can adjust the resources as needed [29].

As we consider a container-based orchestration system for managing the GA load in this thesis, the typical way that a baseline such as Docker Swarm or Kubernetes manages the load, is by using a spread strategy to allocate the task to an existing node. An application can be deployed on a shared cluster of virtual or physical machines connected to the swarm master node using Docker Swarm. The swarm master node acts as the manager of the cluster, allocating an incoming job, while also executing jobs [30].

Docker Swarm uses several strategies to allocate incoming jobs to nodes, one of which is the spread strategy. The spread strategy selects the node with the minimum number of containers deployed on it [30]. Figure 1.2 illustrates a Docker Swarm cluster with four nodes. The first node has been set as a Swarm master, responsible for receiving and allocating the incoming task based on the current scheduling strategy.

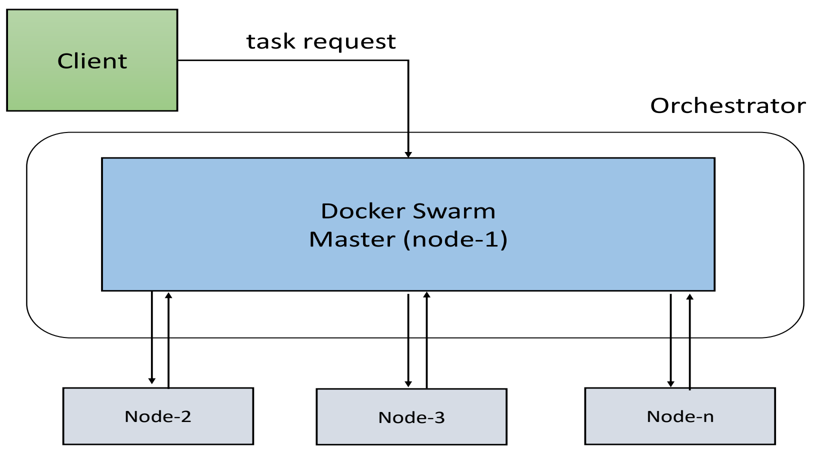


FIGURE 1.2: Docker Swarm architecture.

Kubernetes has a similar structure to Docker Swarm with respect to the master node and worker nodes, except that the master node does not execute any of the tasks. Additionally, the configuration of the cluster is more complex than the Docker Swarm. Further, an incoming task is placed in a pod, where one or more containers can be created to execute that task [31]. There are no other approaches to deal specifically with the GA load since most methods use the basic strategy.

1.2 Use-cases for managing GA workload

The work presented in this thesis considers the specific nature of the workload posed by GAs when used as optimisation engines. GAs are population-based metaheuristics that emulate the natural selection process to gradually improve the fitness of potential solutions to a specific problem [32].

GAs can be configured to guarantee that the maximum fitness within a population in a given generation will never be worse than the maximum fitness in the previous generations (e.g., passing the best individuals to the next generation unaltered). Such a GA configuration is called elitism. This thesis assumes we are always dealing with elitist GAs.

We identified two specific optimisation problems as case studies to provide compelling experimental work and validate our approach. One considered the problem of allocating real-time tasks to a multiprocessor system. In an application with a given number of real-time tasks and a multiprocessor system with a given number of cores, the GA will try to optimise allocating tasks to cores to meet their real-time requirements (using the fitness function developed in [15]). Therefore, the fitness metric is the number of real-time tasks that the GA has successfully allocated.

The other case study considered the problem of solving a Sudoku puzzle. In addition to the number of generations and the population size as input parameters, the GA will receive a level of puzzle difficulty. Based on the level of difficulty, the GA will then generate this puzzle that it will try to solve. After executing the GA, the output of the GA contains the best-achieved fitness and the best chromosome (appearing as a sequence of symbols).

1.3 Research Problems

This section's purpose is to provide an overview of the primary scientific research problems addressed in this thesis. It will also highlight some research works related to these problems. As this section provides a general overview, Chapter 3 will provide a comprehensive literature survey of the existing work.

As previously mentioned, Docker Swarm can be used by a cloud provider to execute the incoming workload from the user. In a high-level illustration of the research problem investigated, a workload which consists of multiple independent tasks that need to be executed is an important step to improve the resource manager. In this work, a specific workload is considered with real-time tasks described in the previous section. This research mainly evaluates how resource management handles the GA workload consisting of specific QoS constraints, such as a hard deadline and required fitness. In this case, resource management must execute the GA task and meet the user-defined deadline and user-defined fitness level.

1.3.1 The problem of managing the GA workload in container-based technologies

There exists a clear gap between container-based orchestration systems and GA. Many works have considered container-based orchestration systems, such as Docker Swarm and Kubernetes, for performance improvement (e.g., [33]). Although they can provide some improvements, they have limitations, like handling tasks with a specific QoS, such as a deadline. On the other hand, GAs are known for optimising solutions in various domains, such as smart factories [14] and embedded multiprocessors [15]. Example parameters that can be input into a GA are the number of generations, the initial population, and the set of application-specific parameters. We often must achieve the desired fitness solution for a specific problem and have this fitness fulfilled by a given deadline. Thus, container-based orchestration systems are unable to handle tasks with a deadline and desired fitness.

1.3.2 The problem of node interference in container-based technologies

The workload consists of numerous of independent tasks. Sharing the resources between multiple tasks are challenging as they can have a negative effect on meeting the required QoS. As mentioned previously, a container-based orchestration system can be used to deploy and execute the incoming task from the user. However, executing multiple tasks in one node can be challenging. When two or more tasks have been executed in the same node, one task can utilise more resources than others, affecting other tasks from achieving the desired QoS. Docker Swarm is considered a container-based orchestration system. It uses several strategies to allocate incoming tasks to the nodes, one of which is the spread strategy. The spread strategy selects the node with the minimum number of containers deployed on it [28]. Figure 1.2 illustrates a Docker Swarm cluster with four nodes. The first node has been set as a Swarm master, responsible for receiving and allocating the incoming task based on the current scheduling strategy. In this scenario, executing two or more GA tasks at the same time can affect achieving the desired fitness on time.

1.4 Research hypothesis

The previous section introduced the challenges and limitations of existing work related to container-based orchestration systems. Therefore, the main objective of this research is to explore and design resource management techniques for handling a GA workload at specific QoS constraints. Along these lines, the below thesis hypotheses determine the central focus of this research. We formulated our hypotheses based on Section 1.3 as:

- QoS-based resource management can handle GA tasks to meet the application's hard real-time timing and user-defined fitness requirements. This resource management considers executing only one task at a time per cloud node and ensures tasks achieve the user-defined fitness level on time.

- Using a situation that is representative of a real scenario, the node interference used in the resource management can lessen the effect when we execute multiple tasks per cloud node. The node interference considers the tasks' information like response time and fitness achieved to determine which cloud node can execute the incoming task from the user with negative impact on other tasks in the same cloud node.

1.5 Thesis contributions

This thesis contributions section outlines the novel contributions made in this thesis addressing the research problems identified in Section 1.3 based on the hypothesis set in Section 1.4:

- **Management of container-based GA workload:** The proposed resource management uses one GA feature: the number of generations to control the execution of the tasks. As mentioned in Section 1.2, additional features are, such as getting a result similar or better than the previous one. Thus, resource management can execute tasks iteratively to meet the user-defined fitness level by the deadline. This way, resource management can increase the number of successful tasks that meet the user-defined fitness level by the deadline.
- **Handling node interference in managing GA workloads:** This thesis presents a weight-based node interference approach to overcome the limitation of resource management. This approach allows multiple tasks to be executed in the same node with less effect on the running tasks. The approach considers the tasks' information during its execution to determine which node is suitable to execute the incoming task.

1.6 Thesis outline

The remaining chapters of this thesis are organised in the following structure. Chapter 2 provides an overview of the main areas that this thesis addresses. It will introduce each area before providing more details. An overview of cloud computing is provided in Section 2.1, including its characteristics, service models, and deployment models. Further, Section 2.2 will provide an overview of resource management in cloud computing. Some of the key points in resource management are highlighted, and the section introduces resource allocation, scheduling, monitoring, and elasticity. Since this thesis considered container-based technologies to deploy and execute the workload, an overview of the area is provided in Section 2.3. Section 2.4 regards optimisation methods, more specifically GA which is considered to be a meta-heuristics algorithm. The last section of this chapter summarises the important points.

Chapter 3 provides a literature review of various areas within resource management. Section 3.1 shows a system model of key concepts, such as workload, static and dynamic allocation. The workload is discussed in Section 3.2 to understand some characteristics and different ways of modelling the concepts, such as DAGs. It also shows the importance of generating a workload that follows the same pattern as industrial examples to get a sufficient result without affecting any sensitive information. Section 3.3 discusses workload deployment, which describes the platform that can execute the tasks such as Docker Swarm and Kubernetes. Section 3.4 explains the demands of workload profiling since the resource manager can use it in allocation. It also looks at a number of studies that used profiling with different parameters. Section 3.5 explains the use of cloud monitoring and other concepts in monitoring the resources provided and passing information about the performance to check the measurements and predict future behaviours, as further explained in Section 3.6. Further, additional sub-sections highlight the demands on prediction and ways to evaluate performance prediction. Section 3.7 highlights different approaches to resource allocation and scheduling that can be used in the process. The last section is the summary, which explains the key points of chapter 3 and introduces Chapter 4.

Chapter 4, defines the experimental platforms, metrics, and methods. It explains an overview of the encapsulation of the GA approach that are used in this thesis. Section 4.2 discussed the application model that the system needs to handle and explains the GA case studies and their application specific parameters. In Section 4.3 the proposed orchestrator that is used in the experiments is explained. Section 4.4 relates to the metrics and how these metrics are used during the evaluation. This is followed by Section 4.5 which describe the experimental method of the work presented in this thesis. The last section is the summary, which explains the key points of chapter 4.

Chapter 5, provides different approaches for managing the container-based GA workload over cloud infrastructure. The chapter starts with an introduction to the work followed by section 5.1 which compares different allocation techniques, including FT, FP, FPLR. Section 5.2, provides details of evaluation of the approaches which include experimental setup and results. Finally, section 5.3 provides a summary of the key points that are discussed in the chapter.

At the beginning of chapter 6, the key concepts used in the chapter including node interference are introduced. Section 6.1 will explain the weight-based node interference approach, how different features used in the approach are calculated, and how these features are normalized. These features represent the slack time which is calculated as the difference between the response time and the deadline. The other feature is the difference in the fitness of the task. Section 6.2 presents the evaluation of the approach which includes the experimental setup and discusses the experimental results. The last section is summary which provides a summary of the key points that are discussed in the chapter.

The last chapter (chapter 7) presents the conclusions of the research presented in this thesis and the identified future works. This chapter will review the main contribution of the thesis as well as the findings in order to determine how well the research problems have been addressed and whether or not the hypotheses presented in this research were valid. This chapter also includes some recommendations for future research.

1.7 Summary

To summarize, this chapter introduces the process of resource management in cloud computing and the benefit of having a resource management that can efficiently allocate the available resources to a workload that is submitted from the user. In this thesis a GA workload is considered which consists of numerous of independent tasks. Thus, one of the benefits of using resource management is using resources in a reasonable way without over or under provision. A further benefit is the availability which provides an additional resource to an incoming workload by scaling the infrastructure and adding more resources.

This chapter also introduced a technology stack which is an example procedure of the resource management that already exists in cloud computing. The advantages of the technology stack is in assisting the resource management for determining when the task is most likely to finish and will allow other tasks to use the resources. The aim of explaining the technology stack in this chapter is to understand the motivation of the research and each of the layer will be explained in more details in the next chapter.

As the resource management in cloud computing is important, this thesis provides several approaches to manage the multiple instances of the GAs running as containers over cloud environment which are: fitness tracking, fitness prediction and fitness-prediction-based linear regression. These approaches only assume one task will be executed at a time per cloud node. Thus, allowing tasks to share resources is frequently beneficial for resource efficiency. For this reason, a node interference is considered to share the same resource and prevents having a negative impact on the running tasks.

Chapter 2

Background

Resource management is needed to allocate and free the resources, keep track, and manage the resources and services, for different providers and users. Thus, this chapter offers an overview of resource management in cloud computing, including resource allocation, scheduling, monitoring, and elasticity.

Since a cloud environment is used in this research, this chapter also provides an overview of cloud computing, which covers its characteristics, service models, and deployment models. A key characteristic of cloud computing is looking at on-demand self-service as both user and provider are able share and request resources or services without interacting with each other. Further, cloud computing uses different service models, such as SaaS, PaaS, and IaaS, to interact and choose a service. These different service models can be deployed either in public, private, or hybrid clouds.

In order to provide suitable resources for the incoming task, cloud providers use VM to handle the execution of the task. However, one of the drawbacks of using VMs is the overhead that can be caused since each VM needs an additional operating system. Thus, providers have taken advantage of container-based technologies to execute the tasks because this technology provides fast and flexible execution. Therefore, the following section is about container-based technologies. This includes the difference between virtual machines and containers and key points of Docker containers. The last section of this chapter will overview optimisation methods, including GA, since it is a meta-heuristics algorithm.

2.1 Cloud computing

Cloud computing is an advancement of Information Technology (IT) and a dominant business model for distributing IT resources. As mentioned before, users and providers can get on-demand access to a shared pool and scalable IT resources managed by the cloud provider. Over the years, different domains have utilised cloud computing within their fields to overcome challenges such as healthcare and engineering. One of the challenges is that of security and data privacy, which can be expressed as an unauthorised user accessing the network. A challenge related to the engineering domain is allocation of reliable and flexible resources with fewer costs [34, 35, 36]. Additionally, public health seeks solutions for managing and analysing their data due to the demand for more resources during the COVID-19 (coronavirus) pandemic. Thus, cloud computing is known for providing an IT solution for numerous domains [37, 38].

In the cloud computing section, we provide an overview of the characteristics of cloud computing with different service and deployment models, as illustrated in Figure 2.1. The first column from the right shows the main structure of cloud computing, which include its seven characteristics, four deployment models and three service models. Detailed information is provided in the following sections.

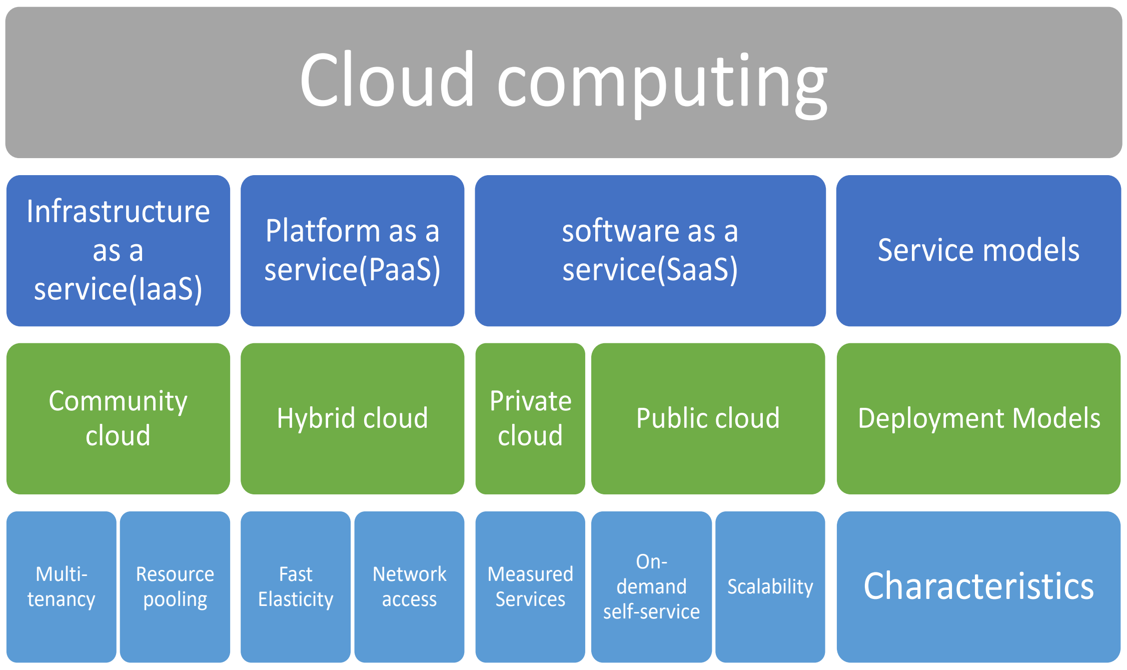


FIGURE 2.1: Characteristics of cloud computing and its models.

Having a cloud environment in this thesis has many advantages, and one of them is low cost. In our work, we rely on several resources to handle the proposed approaches and the execution of the workloads. Providing such physical resources is expensive and not reliable for many reasons. In case hardware resources are used, these resources might run into some issues which lead to permanent loss of resources. Another advantage of using cloud computing is scalability which can resolve the previous issue of using local hardware resources. The same situation can happen to cloud-based resources. However, in cloud computing, the damaged resources can be removed and replaced with new ones to meet the required QoS. Furthermore, security concerns can play a key role in cloud computing as well as local resources. Thus, cloud providers often provide security from outside traffic and prevent any attack on the resources that can reduce the performance of the resources. From the above observations, having cloud environment in this research plays an essential role in achieving a reliable and secure environment.

2.1.1 Cloud computing characteristics

Cloud computing has been used by many users and organisations. Therefore, cloud computing systems provide many attractive characteristics that make future IT applications and services positive [39, 40].

- **On-demand self-service:** Once providers place their resources and service as available, users can choose and select the resources or service they need without interacting with a human.
- **Network access:** This characteristic makes cloud computing unique since it allows users to access cloud resources over the internet at all times and from any device (e.g., smartphones, laptops).
- **Resource pooling:** The resources and the services from the providers are places in shared pools to assist multiple users. Based on the user demand, the resources can be dynamically assigned.
- **Fast elasticity:** Resources and services can be provided to users quickly and flexibly.
- **Measured services:** Resources and services that users currently use are monitored, controlled, and optimised by cloud providers. Cloud computing is a pay-as-you-go model. Therefore, this characteristic ensures the user will pay only for the services used.
- **Scalability:** Providers can add new nodes or remove nodes with minor changes to the infrastructure or the software based on the demand of resources by the providers.
- **Multi-tenancy:** Cloud providers provide service to users. Therefore, multiple users can access the same service simultaneously. Even though

users access the same service; each user is isolated within their customised virtual application instance.

2.1.2 Cloud computing service models

There are several models based on which cloud computing provides its services. Some examples of common cloud computing service models are SaaS, PaaS, and IaaS [41].

The SaaS model allows cloud users to use applications from any cloud provider. The applications are available through a web interface without the need for installing these applications locally. At this stage, the users have no control and cannot manage the cloud's infrastructure or platform. While there are applications available through a web interface, applications such as Gmail and Google Docs can be available on different devices and smartphones. There are several benefits of using SaaS. One of them is that applications are accessible from any device that has access to the internet. Further, users do not need to worry about the infrastructure requirement or using any of the services [39].

The PaaS offers an opportunity for developers to use one of the runtime environments or one of the providers' tools to assist in their development. There are many developers engaged in building a cloud application. Therefore, the developer community is considered strong and able to support the new developers in their application development. Further, the developers do not have to handle the updates and the upgrades related to the infrastructure as it is taken care by the cloud provider [39].

In the IaaS model, cloud providers provide resources like CPU, memory, and operating system to users. IaaS uses virtualisation technology to make it easier to provision and release the resource from the users. The benefit of using IaaS is that users pay for the demand resources only. Also, the user can easily scale the resources up and down based on their requirements [39].

2.1.3 Cloud computing deployment models

Cloud computing must have several models to deploy its resources and services for the cloud users to use them. Each of the deployment models have some restrictions (e.g., public v. private). These models are public cloud, private cloud, hybrid cloud, and community cloud. The first deployment model, the public cloud, is available for general use. Therefore, any user can use any of the resources and services provided by the public cloud. The next deployment model is the private cloud. Private cloud is made accessible for only a single organisation. Therefore anyone within the organisation can access the cloud and use any of its resources. The community model considers multiple organisations that share the same concern to share the cloud infrastructure. The last deployment model, the hybrid model, comprises two or more cloud models [41].

2.2 Resource management in cloud computing

A cloud computing infrastructure is a large distributed system with many processing resources. These resources deal with inconsistent user requests and the consequences of other external factors that are beyond the control of the user and system administrator. The performance, functionality, and computational cost of system evaluation are all influenced by cloud resource management. Also, resource management requires complicated decisions and rules for multi-objective optimization. Several factors make the process of resource management difficult to attain exact information state, such as the complexity of the system, constant and unpredictable interactions [16].

Resource management methods linked to the delivery models of cloud computing differ from one to another. The cloud providers deal with unpredictable and huge workloads in all cases, forcing the concept of cloud elasticity into challenge. Thus, the workload fluctuation in cloud computing is still an issue that researchers are attempting to solve. Therefore, when the cloud providers can forecast an increase in workload, they can reserve resources ahead of time such as in seasonal web applications that might cause fluctuations [42].

The fluctuation of the cloud workload can cause either over-provisioning or under-provisioning of resources. The under-provisioning of resources happens when providing fewer resources for the user than needed. It is the other way around in the over-provisioning case. One of the solutions for such a problem is auto-scaling. Auto-scaling is a mechanism for dynamically adjusting the resources given to elastic applications based on workloads that are received. The goal of the approach is to adjust the resource and assists in making a decision by either allocating, reallocating or releasing the resources to meet the fluctuated workload [43].

As the resources provided by the cloud provider are trying to cope with the fluctuated workload, the resource management can face another challenge when considering efficiency, namely energy efficient, cost minimization, performance optimization, etc. Thus, when the resources are running either at 10% of the

CPU utilization, which considered to be idle, or when they reach around 90% which consider to be closer to the saturation point. In all cases, the power consumption is affected by the use of the power. Therefore, one of the solutions for such an issue is using Dynamic Voltage and Frequency Scaling (DVFS) for efficient use of the resources and can assist in reducing the energy consumption. This technique used to control the frequency and the voltage of the resources to achieve reduce the power consumption while maintaining the QoS requirement [44].

The following subsections of resource management will discuss the key stages of resource management: resource allocation and scheduling, resource monitoring and elasticity.

2.2.1 Resource allocation and scheduling

Resource allocation and scheduling occur when workload consisting of one or more jobs where each job consists of one or more tasks arrive at resource allocation and the scheduling is to be processed. Thus, Manvi and Shyam in [18] have clarified the overlapping concepts between resource allocation and resource scheduling. Resource allocation is defined as assigning incoming tasks to the available resources. Once a new task arrives, one of the resource allocation techniques will handle the allocation process. In contrast, resource scheduling is a timetable of tasks and resources that must be executed at specific times for a specific duration. Also, processing the tasks depends on the dependency of the tasks.

Resource allocation can be either static or dynamic. In static allocation, information about the workload is known in advance, whereas the workload may change during dynamic allocation [5]. One of the disadvantages of static allocation is that it can lead to the over-provisioning or under-provisioning of resources. Thus, having an effective static allocation technique is challenging. Further, the process of resource allocation and scheduling can be based on different objectives. For example, one can allocate the resources to maximise their utilisation and minimise their costs or maximise their profit.

Cloud providers provide users with assured QoS in order to meet their users' needs. SLAs are used to formally manage these QoS contracts. The resources are provisioned forcefully from the cloud data centers to meet the users' SLA needs. This method of resource allocation may result in inefficient resource allocation, which will have a negative influence on resource utilization. Furthermore, it will result in a workload imbalance among the data centres, resulting in some under utilization or over utilization of the resources. Moreover, some of the jobs will have to wait longer than expected to be executed. Thus, one of the solutions for such problems is resource migration. There are several benefits of using resource migration such as scheduling optimization and energy-aware resource scheduling [45].

2.2.2 Resource monitoring

Once the demand for resources increases, efficient cloud monitoring is required to provide accurate and fast information. The benefit of having such monitoring is to distinguish the working resources from the ones turned off. The other benefit is to detect abnormal behaviour that can slow the managing the resources or services. Another benefit of cloud monitoring is that the monitoring information can help provide a good understanding of the required resources during the scaling process, so resource management can either add or remove resources to fulfil the requirement [46].

Not only can resource management take the positive features of cloud monitoring, but cloud users can monitor their usage for different purposes, such as the cost of the resources or a fault in one of the resources. As illustrated in Figure 2.2, there are different components of the monitoring architecture. The monitoring process starts with the monitoring agent that can monitor different resources and services. Once the information is collected, the monitoring agent will report back to the monitoring server, which will issue an alert, such as reaching the threshold of the cost. Another way is when the information is stored in a database such that the front-end system can show reports and graphs related to the resource and services used.

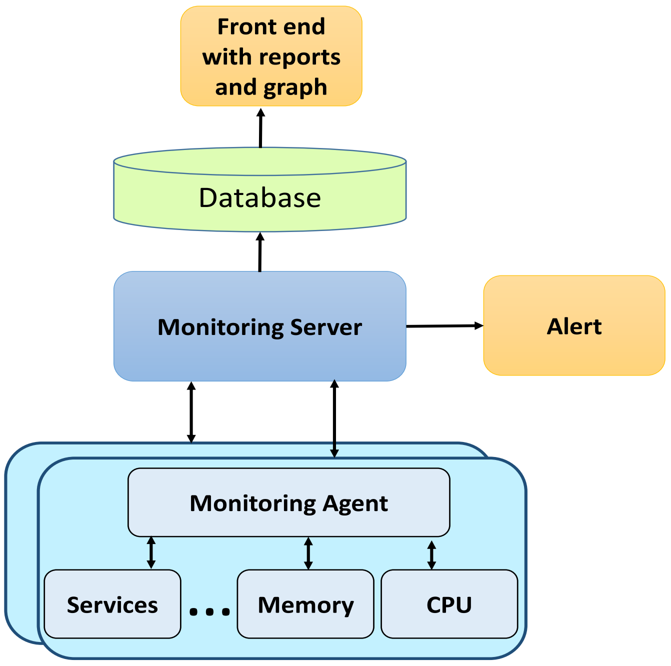


Figure 2.2: Example monitoring architecture.

2.2.3 Elasticity

As the amount of workload changes over time, more resources can be requested or released from the users. Also, some of the current jobs might need more or fewer resources. These situations can be handled via elasticity within cloud computing. The general definition of elasticity in cloud computing is the ability to reconfigure the computing resources of an application based on real-time requests. Moreover, a system must be scalable for it to be elastic. Which means that the system is able to adapt to a sudden workload increase. Thus, there are two kinds of scalabilities. One is vertical scaling, which is the ability to scale up or scale down, such as add more CPU or memory within a node. The other kind is horizontal scaling, which concerns adding or removing computational resources (e.g., node or instances). Figure 2.3 shows the difference between vertical scaling and horizontal scaling [47].

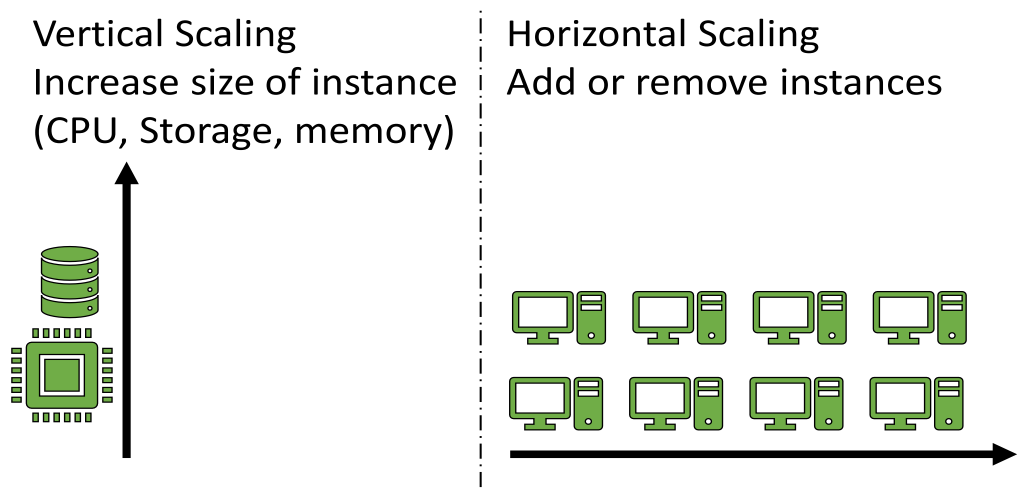


FIGURE 2.3: Vertical scaling VS horizontal scaling.

2.3 Container-based technologies

This section provides the background of container-based technologies and their usage within cloud computing. Traditionally, cloud computing used virtual machines (VMs) as virtualisation techniques, allowing multiple users and applications to use the same resources of a single node at the same time. Figure 2.4 illustrates the architecture of the VMs versus container-based technologies. Different VMs have independent operating systems (OS) on top of the hypervisor. The hypervisor layer is for monitoring different VMs since they all use the same infrastructure and OS. In contrast, container-based technologies do not have the guest OS for each container since they use a container daemon to build, run, and distribute the containers. One of the platforms that can be used to deploy the workload is Docker containers, which use the same architecture of container-based technologies, as illustrated in Figure 2.4.

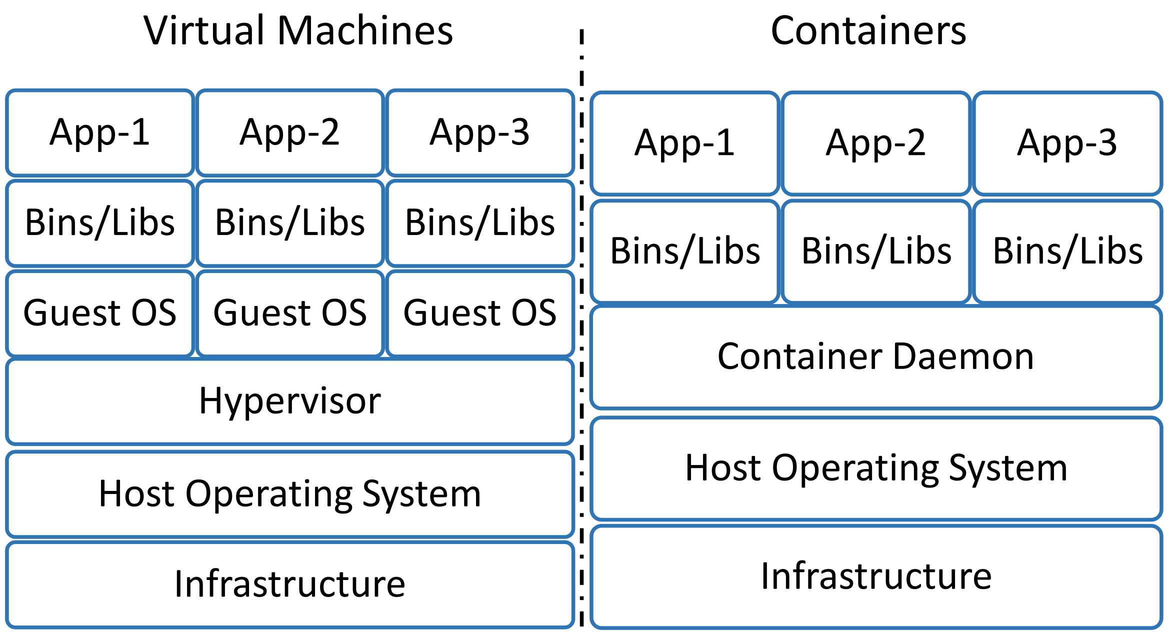


FIGURE 2.4: Virtual machines VS Container-based technologies.

Docker containers are an abstraction to help organise the workload, hide the details, and deploy the application in an isolated environment. Docker containers can be installed on a physical machine. Following this, containers can be added and managed to fulfil the application requirements [48]. There are several components in Docker containers, as described below:

- The Docker engine is placed on the host operating system, and several Docker containers can be built on top of the Docker engine.
- The Dockerfile is a file that contains instructions needed to allow the application to be executed on containers.
- Once the Dockerfile is built successfully, a read-only Docker image is created containing the application to be executed.
- The Docker container is the running component of the Docker image.
- The Docker Hub is the repository where Docker images are stored.

A key factor in the need for Docker containers is to add intelligence to resource allocation. For illustration, assume that there is an existing image that contains a web application. A container can be created to run the application, and additional containers can be created if there is a database service that needs to be run with the application. Each container works in isolation. Therefore, the only way that containers can communicate is by creating a network channel that is specified during the process. As another example, Figure 2.5 shows a Dockerfile used to build a simple image that prints a statement. The image that needs to be built is based on the OpenJDK image version 8 and must specify the working directory and additional commands to run that image. Therefore, building an image can be based on an existing image or from scratch. One benefit of using container-based technologies over VMs is less overhead since the containers do not include the guest OS. Another benefit is increased portability. The application is built inside an image, which can be used and deployed in Linux and Windows systems.

```
FROM openjdk:8

COPY . helloworld
WORKDIR helloworld

RUN javac helloworld.java
CMD ["java", "helloworld"]

```

Figure 2.5: Example of a Dockerfile

Containers have the advantage of introducing less virtualization overhead than VMs because there is no additional virtualization layer. Instead, they are immediately executed on the host OS's kernel. Containers are more efficient and allow for better scalability as a result. However, because containers were not built as a security tool to isolate between untrusted and potentially malicious containers, the lack of a virtualization layer poses new security vulnerabilities due to the lower level of isolation. Containers placed on the same host share a common operating system, making them vulnerable to attacks on shared resources such as the file system, network, and kernel. Another limitation of Docker container is providing a cross-platform compatibility. One important difficulty is that if an application is built to run in a Docker container on Windows, it will not run on Linux, and the other way around. Virtual machines, on the other hand, are not bound by this limitation [49].

2.4 Optimisation methods

There has been increasing demand for cloud computing in various domains, such as engineering, industrial, and business computing. Many of those domains use optimisation tools to improve solutions to their domain-specific problems. The goal of using optimisation is to maximise the result with limited resources. Further, only the best solution is selected from a set of solutions that the optimisation method provides. As previously mentioned in the introduction that this thesis uses a GA-based workload. Thus, this section provides an overview of meta-heuristics and GA, which is considered one of the methods of meta-heuristics [50].

A meta-heuristic is a higher-level heuristic intended to discover, generate, or select a heuristic. It may provide an effective solution to an optimisation problem. Enough information on the problem and computation capacity are needed to achieve such an effective solution. Meta-heuristics provide a set of solutions for an optimisation problem. There are few assumptions that meta-heuristics may assume about the problem they are trying to solve [51].

Most meta-heuristics are either classified as a local search, global search, or a hybrid meta-heuristic. Generally, local search optimisation is considered a more exploitative method by collecting search experience to provide high-quality solutions, such as an iterated local search (ILS) algorithm. On the other hand, the global search is a more explorative method and can extend the search in a wide domain, such as ant colony optimisation (ACO) and GAs [51].

Meta-heuristics are also classified as either single- or population-based heuristics. Numerous solutions are provided in the search process, which decide whether the meta-heuristic is a single solution or a population-based algorithm. A single solution or population-based algorithm must be selected to choose a meta-heuristic for a specific optimisation problem. Single solution meta-heuristics are more exploitation-oriented, whereas population-based meta-heuristics are more explorative-oriented. One of the population-based meta-heuristics is the evolutionary algorithm [32].

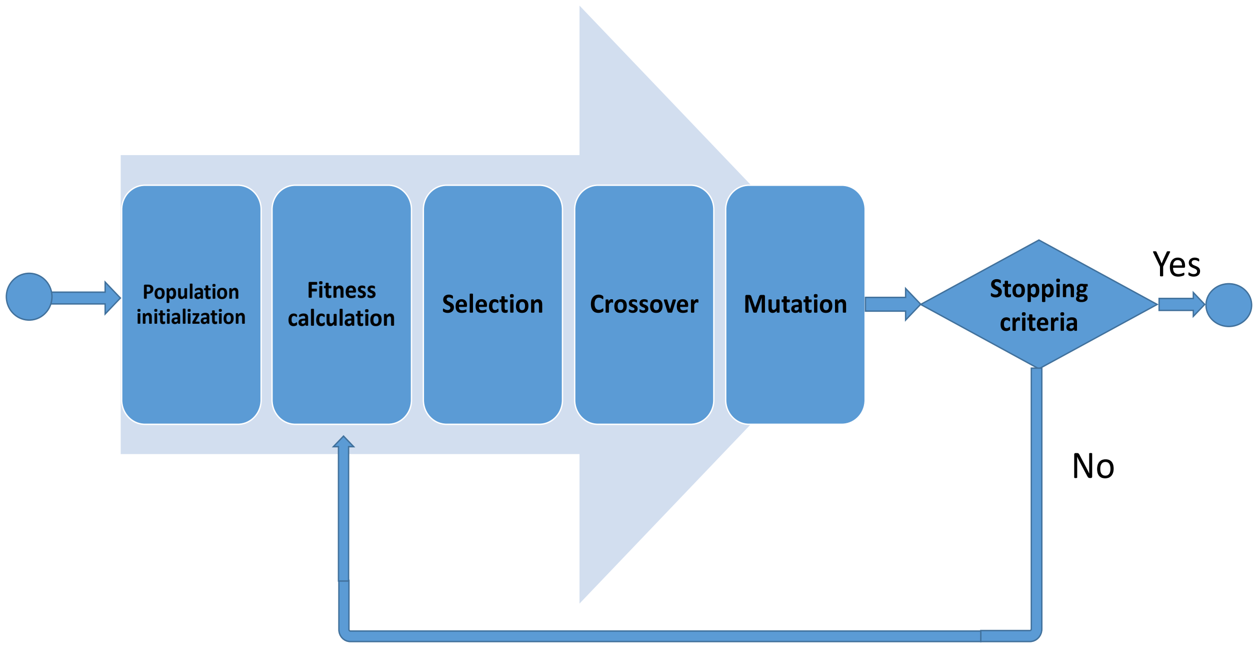


FIGURE 2.6: GA process.

The evolutionary algorithm is a set of algorithms in a global search. It is a population-based meta-heuristic and naturally inspired. One of the algorithms within the evolutionary algorithm is GA. As illustrated in Figure 2.6, the process of the evolutionary algorithm is started initially by generating a set of solutions (population initialisation). Each population has multiple chromosomes and within each chromosome includes multiple genes with representative values as either (0 or 1) or different representation based on the specific type of problem [52].

A fitness calculation (fitness function) decides how suitable a chromosome, or an individual, is compared to others within the population. Thus, a fitness score is given for each chromosome which represents the probability of being selected for reproduction. After that, a selection step selects the most suitable individuals and uses their genes for the next generation. In the crossover step, each of the chosen individuals is a parent. A crossover point is selected at random within the genes. After that, the process reaches a mutation step. Some of the parent genes face mutation at random. This process is repeated until a stopping criterion is met [52].

One strategy that can be applied in the selection stage is elitism. Using this strategy allows the best individuals from the current population to be carried over to the new population. This way, the elitism strategy guarantees the best solution in every iteration until the process reaches the stopping criteria [32].

2.5 Summary

This chapter has provided an overview of the related areas this thesis considered. This chapter started with an overview of cloud computing and its different models and characteristics, followed by an overview of resource management of cloud computing as it is the key area in managing the GA workload. Container-based technology is used to deploy and execute the GA workload. Therefore, an overview of the area is given. The last area is the optimisation method used, particularly GA.

Chapter 3

Literature Survey

3.1 System model

As businesses and industries migrate to cloud-based systems, more issues are being addressed at every layer of the technology stack (Figure 1.1). One reason that businesses migrate to cloud computing is to take advantage of the three models, IaaS, PaaS, and SaaS. The main actors in cloud computing are the users and providers. A **user** is one who submits a task to be executed. A **provider** provides resources such as a **Virtual Machine** (VM) to handle the execution.

Two factors need to be considered when submitting a task or providing resources. These are **Quality of Service** (QoS) and **Service Level Agreement** (SLA). An SLA is a legal contract between the user and the provider based on the user's expectations and requirements. The user can specify several SLA requirements, for example CPU capacity, memory size, availability, and price. QoS is a performance measurement, which needs to be met in order to fulfil the SLA requirements. Examples of QoS are, response time, and throughput [53].

A cloud user can send a **workload** which consists of several jobs to be processed. Each **job** consists of multiple tasks with their dependencies. Thus, a **task** is a unit of work to be done that is indivisible. Figure 3.1 shows the process of allocating the workload. Knowing the availability of the resource can be challenging, so **monitoring** the infrastructure provides information regarding the resource status and its capacity.

Resource allocation is defined as assigning incoming tasks to the available resources. Once a new task arrives, one of the resource allocation techniques will handle the allocation process, based on **workload profiles**, which execute the job in advance on multiple configurations to determine the right resource to execute the job. Resource allocation can be either static or dynamic. In **static**

allocation, information about the workload is known in advance, whereas in **dynamic allocation**, the workload may change during the process. Admission control is an example of resource allocation which uses information from the workload like worst-case response time to determine whether to admit the workload or not based on its deadline [5].

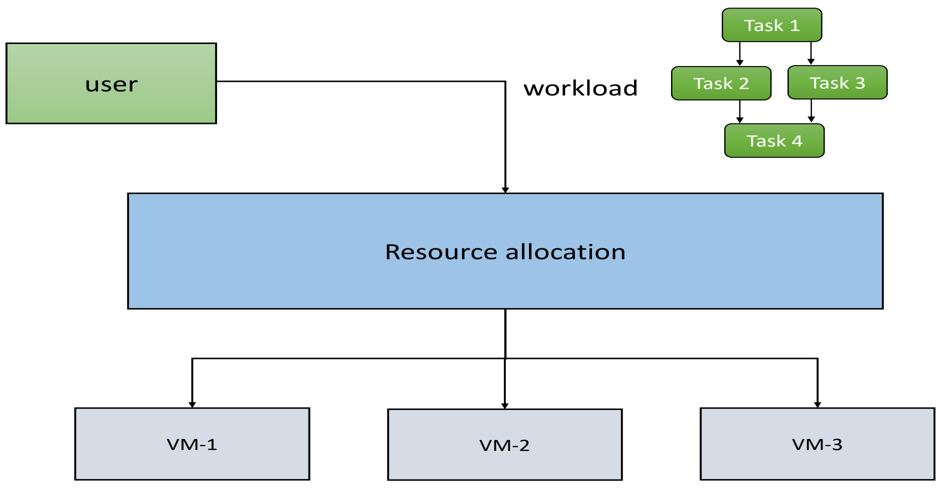


FIGURE 3.1: Allocate the workload to a VM

3.2 Workload model

In order to assist the resource manager in making the right decision when allocating resources, it is essential to understand the GA workload and its behaviour. By understanding the behaviour of the GA workload, different areas can be optimised such as performance, cost, and energy.

Understanding the application behaviour involves several characteristics including, parallelism and scaling, dependencies between tasks, execution time analysis, affinity and value. One of the issues within the workload is the change of application structure over time including the number of tasks and its dependencies. Another issue is related to the different types of resources that are available to run the application which result in unfair allocation of the tasks. Therefore, heterogeneous resources and workload complexity can be considered to be a challenge within a workload model [54] [55]. Based on the measurement of an existing system which can be obtained from an entity such as a log file, it is possible to model a workload that can then be used to generate a synthetic workload. The main goal of workload modelling is to offer performance analysis, simulation of cloud resource management approaches, and allowing cloud providers to improve their systems' QoS without having to create costly large-scale environments [56]. Some researchers, such as work done by Burkimsher et al. in [57], have developed a synthetic workload generator which has the key features of the real workload. The benefit of the workload generator is having better control of the workload and its behaviour. Several aspects need to be considered when generating a realistic workload, for example, inter-arrival times, job size, and the dependency structure of the job.

3.2.1 Application characteristics

One of the key features of an application is parallelism which happens when the task runs on multiple cores and is executed simultaneously to solve the computational issue. Parallelism can improve performance as it uses extra resources to complete the tasks. A parallel application may contain several parts with dependency constraints between them. Thus, the current part will wait for the previous part to finish executing prior to executing the current task. Also, each part can have one or multiple tasks which can be executed on numerous VMs for fast execution and reduce the amount of time that the job can take to finish all the tasks [58]. In order to control the level of parallelism, an auto-scaling approach can be used which will be covered later.

Along with parallelisms, an application might contain some dependencies among the tasks that need to be considered. For example, if task B depends on task A, it means that task B cannot be performed until task A is finished executing. It is important to know whether an application has dependencies or not, because it can affect the overall execution time. Task dependencies can be represented as Directed Acyclic Graph (DAG) structures [59]. There are different shapes of DAG, such as linear, independent chain, and diamond. Figure 3.2 illustrates the different shapes of DAG. The left shape represents linear dependencies, the middle shape represents an independent chain, and the right shape represents diamond dependencies, where nodes are tasks and edges are dependencies between the tasks [60].

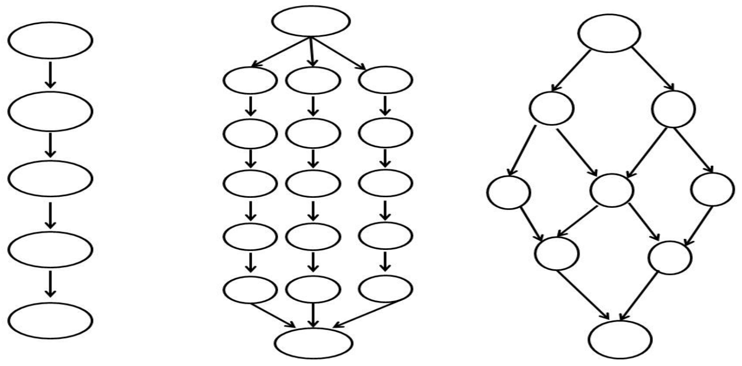


FIGURE 3.2: DAG shapes.

When an existing application contains parallelism and dependencies between tasks, a real-time analysis is a one-way process to determine the amount of time it takes to execute the application and then allocate suitable resources to process it. The benefit of real-time analysis is that it informs about the the upper and lower bounds of the application which can help improve the responsiveness of an application. There are several concepts which are helpful to understand when analysing the time a task takes including the worst, best, and average case execution time [5]. The execution time of a given task is dependant upon the input value of the application, which is why the worst, best, and average are not equal. Worst-case execution time (WCET) is the longest execution of a task, and the best case is the fastest execution time of a task [58]. Figure 3.3 illustrates how important real-time analysis is when executing a task as it can minimise the time it takes during execution [1].

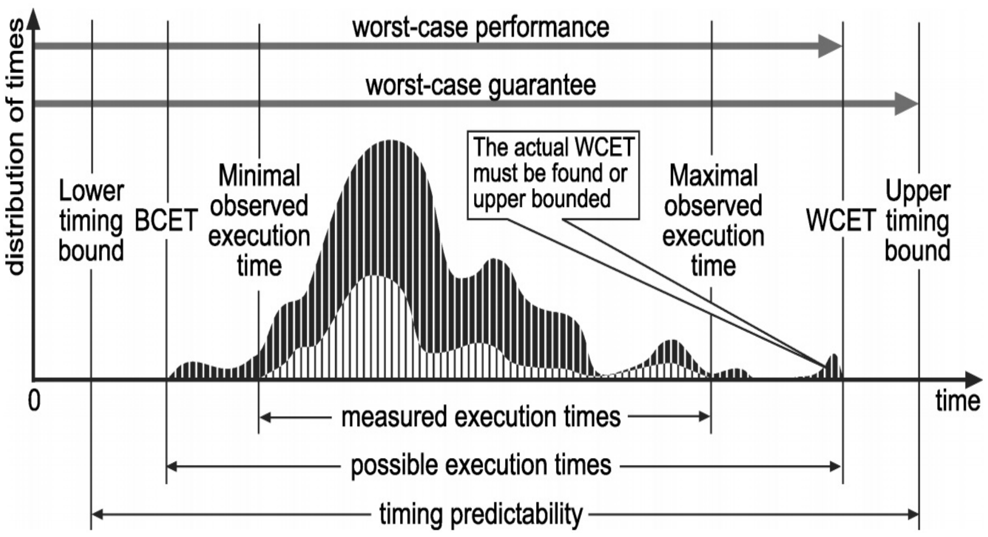


FIGURE 3.3: Timing Analysis. Extracted from [1].

An example application that considers these characteristics is a weather application that is illustrated in Figure 3.4. In this application, 7 tasks are used in order to visualise weather information. Task one is the collecting phase from other sources of information such as radar data. Then, tasks 2, 3, and 4 can be executed in parallel, as the main goal is to calculate the different values including visibility, turbulence, and cloudiness. It is important to complete the execution of the previous tasks in order to start execution of task 5, as it depends on tasks 2, 3, and 4. Task 5 will collect the numerical values and analyse them to predict the weather. The last two tasks represent dynamic modelling and visualising the weather information. The execution time of this application depends on two factors which are, dependencies and parallelism. An example of dependencies is seen in task 7 which depends on task 6, whereas tasks 2, 3, and 4 represent parallel tasks [59].

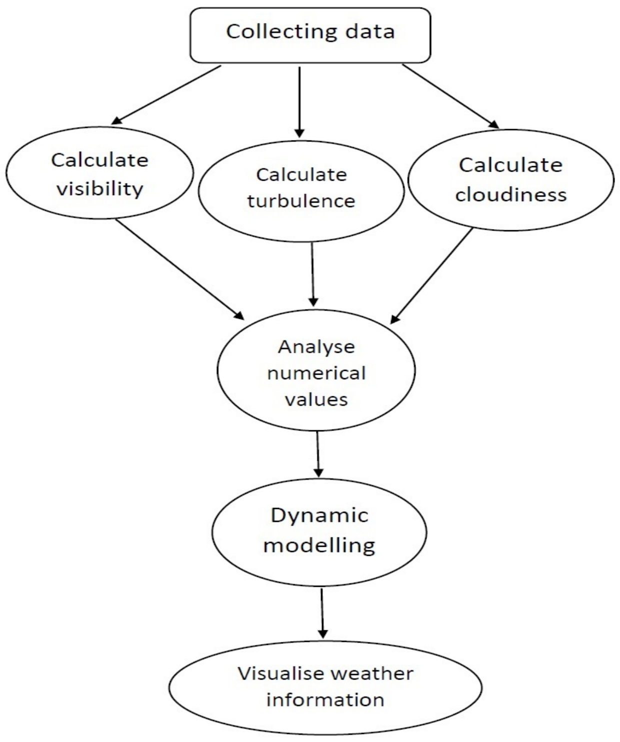


Figure 3.4: weather application as an example

3.2.2 Workload generator

Given the importance of understanding the workload and its characteristics, it is essential to generate a workload that performs similar to a real workload. There are several reasons to support the importance of generating a workload. The first of those reasons for using a generated workload is to avoid seeing any sensitive information. The second reason is the possibility of running hundreds or thousands of experiments and reproducing those experiments whenever needed. Therefore, when the experiment is carried out in a real-world environment, the result will be similar to that achieved before [56]. Curiel and Pont [61] explore different workload generations and describe various design options in order to generate a workload. Two approaches can be considered for providing input to the workload generator. One using the trace log of an existing system, and second using a statistical model. Furthermore, several parameters can be tuned when generating a workload, such as, the arrival time of the job, the size of the job, and dependencies between tasks.

Galindo et al. [62] generate a workload for different load intensities such as memory intensive which require to meet memory capacity for the load to be executed. Although, they generate a workload based on probability distribution, they do not cover dependencies between tasks as demonstrated in the work done by Burkimsher et al. [57]. The aim of their work is to characterise the grid workload of an engineering design department by analysing log files spanning a period of 30 months. Several workload characterisations are covered, such as dependency between tasks, execution time, and variation of arrival times. They observe that the highest rate of task submission is during working hours. One of their workload generation approaches generates a workload with task execution times which is similar to their observations from analysing the log files.

3.3 Workload deployment

Workload deployment is an important stage of resource management since the technology used to deploy and execute the workload may affect the QoS achieved. Thus, a Docker container can be used to deploy and execute the workload to overcome this issue. Docker containers are classified as lightweight because they have a low overhead in comparison to VMs. Docker containers do not require a virtualization layer to run the workload. Instead, they are deployed directly on OS [48].

Within the container-based technologies that handle the workload deployment are Docker Swarm and Kubernetes approaches which are considered state-of-the-art container orchestration systems. These orchestrators are open-source platforms and are able to handle the deployment of the workload as they have a resource management within the orchestrator to decide which virtual or physical machines (node) can execute the workload. Using Docker swarm, an application can be deployed on a shared cluster of nodes that are connected to the swarm master node. The swarm master node acts as the manager of the cluster which handles the allocation of incoming job. The swarm master node can also execute the jobs by itself [28].

Docker swarm uses the following three strategies to allocate an incoming job to the nodes: spread, binpack, and random strategies. The spread strategy selects the node with minimum number of containers deployed on it, whereas binpack selects the node with minimum amount of CPU and RAM available. Figure 3.5 illustrates a Docker Swarm cluster with four nodes. The first node has been set to be a Swarm master which is responsible for receiving the incoming task and based on the current scheduling strategy allocates the task [28].

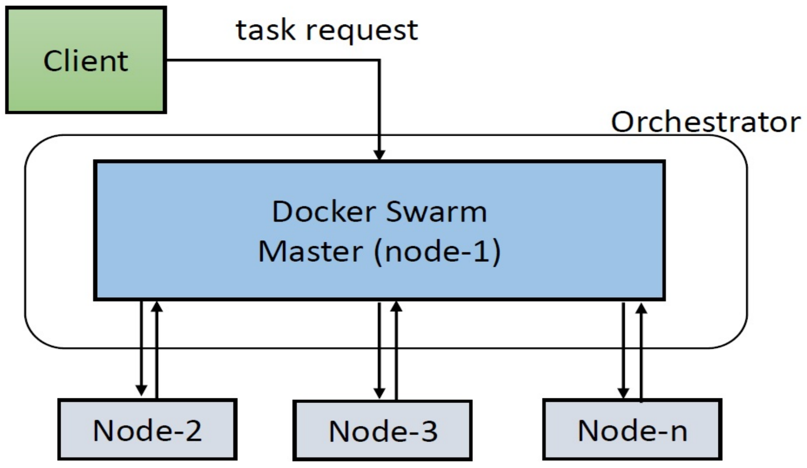


FIGURE 3.5: Docker Swarm architecture.

Docker Swarm uses a two-step process to execute the task. The first step is using filter feature which can be either node filtering like constraint, health, and container slots; or configuration filtering like dependency and port. Then, using one of the strategies, the task is allocated to the right node [63].

Kubernetes has a similar structure to Docker Swarm with regard to the master node and worker nodes. Except that, the master node will not execute any of the tasks. In addition, the configuration of the cluster is more complex than that of Docker Swarm. Furthermore, each node in Kubernetes cluster can run pods where each pod consists of one or multiple Docker containers. As pods are considered to be the smallest unit, Kubernetes can only control the pods [27]. The scheduling process of Kubernetes is based on filtering the nodes in the cluster by taking into consideration the pod requirements. After filtering, the node with the highest score is selected to execute the task. One way of computing the scoring is in finding the node utilization and then the scheduler places the pod to the node with the highest score [64].

A number of studies explore how Docker containers assist the resource management process. Docker containers allow multiple deployment of the same application to provide availability for the service and reduce the waiting time. For example, using the existing cluster configuration to deploy, the Apache Cassandra application needs to run one instance of Cassandra per server. Apache Cassandra is a database that handles large data providing scalability which leads to high availability. Therefore, each server needs to be connected with at least one other server to support availability for the service. In real-world environments, the server has high-end components which could cause under-use of resources as the demand is less than the provided resources. Therefore, deploying Cassandra on Docker containers can solve this issue as Docker allows deployment of an application on multiple instances as shown in Figure 3.6 [33].

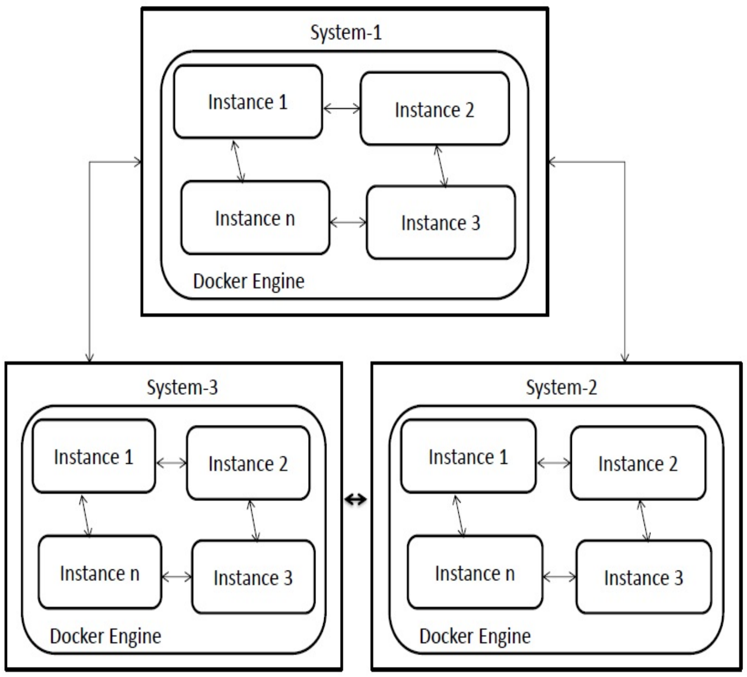


FIGURE 3.6: Use Docker to deploy multiple instances of application

As the application is being deployed on the Docker, the user needs to use the infrastructure in an efficient way to avoid any waste of resources. Thus, a performance measurement can determine when to assist the process of allocating services to users by knowing when an existing resource will become available

[65]. Shirinbab et al. [66] evaluate the performance of the Cassandra application when deployed on Docker versus other VMs. They selected several performance metrics for their comparison between Docker and other VMs such as CPU utilisation, throughput, and mean latency. Their results show that Docker containers have lower overheads compared to VMs.

As Docker Swarm contains a resource manager to handle the execution of the tasks, the scheduling strategies may not serve some businesses as they might focus on specific requirement such as increasing their profits. Liu et al. in [63] propose a new scheduling strategy based on a multi-objective optimization container to improve the QoS performance. Their work takes into account several factors like CPU usage, memory usage, and the time spent transmitting images over the network. The authors develop a measuring technique for each critical component to establish a scoring function for each one and then integrate them into a composite function to determine the most suitable node to deploy the containers needed to be assigned in the scheduling process [63].

3.4 Workload profiling

Understanding resource characteristics including storage, memory, and network capability, during resource management can assist in providing the right amount of resources needed for the given requirements. Using workload profiling during resource allocation can assist resource management as a profile can provide good understanding of the workload characteristics such as dependencies and parallelism [67]. *Workload profiling* is the process of analysing changes in the workload that was previously executed. Given a metric such as execution time, workload profiling needs to be optimised when executing on different platform configurations. Workload profiling thus can be used to estimate job execution times and the amount of resources needed when a job becomes available [68].

According to [69], there are several stages in profiling such as, data granularity, monitoring, storing, and processing. At the data granularity stage, the main goal is to define the metrics that need to be profiled. In addition, resource management can benefit from this data and provide a better QoS with better response time, throughput, and cost for users.

There are a number of studies in the literature using a profile-based approach within resource management, for example [23] and [70]. The goal of Singh et al. [23] is to allocate resources in a way that optimises energy and value using profiling and non-profiling approaches. The data used during the profiling approach was from previously executed applications. This data was collected for each task, such as voltage/frequency level, value curve, arrival time, and energy consumption. During resource allocation using a profiling method, this data, along with the High Performance Computing (HPC) platform, was input into the algorithm to perform the allocation. Their algorithm uses the input parameters explained above and the result allocates incoming tasks as having a positive value. The algorithm consists of several parts which are listed below.

- It monitors the execution of tasks, once there is an change, it updates the resources.
- It monitors incoming tasks to place them into a queue.
- It collects all the bids from the users to choose the maximum bid.
- It computes the value and energy of unscheduled tasks when using resources.
- Based on the profiling result, the algorithm can select the maximum value per energy consumption and its value, energy, allocation, and voltage/frequency levels.
- It schedules the task and updates the resources.

Profiling the resource utilisation can be useful in order to determine the effectiveness of the resource allocation when executing a workload. There are some utilisation parameters that can be considered such as CPU and memory. Profiling the resource utilisation can be used when migrating a VM or powering a new one [17]. Dezhabad et al. [71] developed a new classification scheme for workloads based on resource consumption. The researchers employed a hierarchical clustering approach to generate three workloads and resource demand profiles for low, moderate, and high-demand applications. The presented technique assists cloud providers in resource allocation optimization and profit improvement.

3.5 Cloud monitoring

Cloud monitoring is the process of collecting data about resources and managing them. Several studies have considered cloud monitoring to be a crucial step in resource management as it monitors the infrastructure layer for different metrics such as resource use, the amount of energy used, and availability. This checks if there are any faults in the resources or the physical VM is off and needs to be turned on [72]. Several monitoring tools are currently used for different monitoring purposes, such as Amazon cloud watch, which is used only for Amazon resources. As reported in [72] this tool has some limitations including the model being designed for centralised models not ensuring the availability of resources.

In terms of usage, there are some usage parameters that can be monitored such as CPU and memory. These parameters can assist resource management during the process including VM migration and powering on of new physical machines. Two scenarios can happen during resource management, under-use, or over-use of resources. Over-use happens by allocating more resources than needed. Whereas under-use happens by allocating less resources than actually needed [17].

An example paper on monitoring resources is Du and Li [2] which presents an automated tracking, orchestration, and monitoring (ATOM) framework to monitor resource usage in an IaaS system. They also use a novel tracking method to continuously track important system usage metrics with low overheads. Reducing overheads is done by collecting data from each Virtual Machine (VM) every minute and then reporting it to the Cloud Controller (CLC) through a Cluster Controller (CC). Furthermore, ATOM tracks several metrics from each VM user such as CPU use and network I/O [2].

As Figure 3.7 shows, there are several components in the ATOM framework, one of which is a tracking component. In this component ATOM adapts the optimal online tracking algorithm for one-dimensional online tracking inside the monitoring service on NCs. Another component is Monitoring (anomaly detection). ATOM adds this component in CLC to analyse tracking results by the tracking component which provides continuous resource usage data in real-time. It uses a modified PCA method to continuously track the divided subspace and automatically detects anomalies by identifying notable shifts in the subspace. Also, this component adjusts the tracking threshold from the tracking component dynamically based on data trends and false alarm rates. The last component is Orchestration (introspection and debugging). When a potential anomaly is identified by the monitoring component an introspect request along with anomaly information is sent to the orchestration component on NC to raise an alarm to cloud users for further analysis [2].

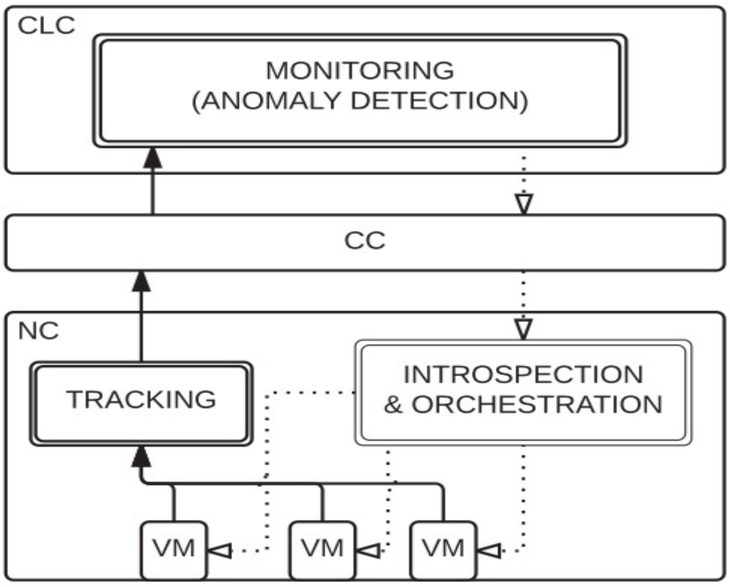


Figure 3.7: ATOM framework. Extracted from [2].

Moreover, Docker swarm [73] was used as a container-based technology to execute the workload. It consists of several nodes one of which acts as the master node. Within the master node, a monitoring component collects information from the existing nodes. This information can be the number of containers in each node, CPU utilization, or memory utilization. Monitoring these nodes helps the resource manager within the master node to allocate the task to a suitable node based on the resource allocation policy currently used.

The other use of the monitoring within the resource management is in monitoring the resources in terms of energy consumption. Enes et al. [74] develop a platform that controls a power budget to limit the amount of energy spent by users, apps, and individual instances. Their power budget platform was built by combining and extending numerous tools such that the CPU shares of containers can be scaled down or scaled up to minimise or increase energy consumption accordingly. One of the tools that is used in their platform is POWERAPI which is a tool that monitors the infrastructure and reports back to the platform about the existing energy consumption.

3.6 Performance prediction

The idea behind prediction is in forecasting the future behaviour of specific applications based on the given information in order to provide the right resources during the resource management process. The goal of using performance prediction is to facilitate the QoS that the user expects. There are several issues that can be avoided by predicting performance, such as over-provisioning or under-provisioning [75].

There are several scenarios that can be improved through the use of performance prediction. One of them is assuming the application needs of another VM. Therefore, the process of turning on a VM is time consuming, and the user will notice the time it takes to run the new VM. Likewise, in cases where a prediction model has not been used, providing the right amount of resources can be challenging and may result in under-provisioning or over-provisioning [26].

3.6.1 Why do we need performance prediction?

As mentioned previously, performance prediction is a crucial step in resource management for a number of reasons. One reason is for optimising the cost of resource usage. Users and providers can obtain this benefit by estimating future usage and ways to reduce cost. There are several ways to reduce the cost when using prediction, such as the cost of energy, networking, and maintenance. One of the studies that uses performance prediction to optimise the cost is by Kim et al. [6] whopropose an end-to-end elastic resource management system for scientific application on a public IaaS cloud.

As there are different costs for every VM from cloud providers such as Amazon, one of the strategies Kim et al. use is related to calculating the performance and cost ratio to determine the value of the task (which can be calculated as - minute cost * execution time). Resource evaluation of the performance and cost ratio aims to satisfy the deadline for each task and reduce the total execution cost for all tasks. As the prediction model for task execution time produces several VM options, the approach evaluates VMs in order to select the most cost efficient approach which meets the deadline. For instance, if there is a task (task A) with a one-hour deadline, the prediction results for task execution time on four different types of VMs are as shown in Table 3.1. Using the cheapest VM (m1.medium) of the three (m1.medium, m1.large, and m1.xlarge) is the best in terms of cost efficiency because m1.medium has the lowest (best) value for the performance-cost ratio [6].

TABLE 3.1: Prediction results of task execution time for three different tasks. Extracted from [6].

VM info		Prediction result		
Type	Min. Price	task A	task B	task C
m1.small	$0.00125	80 min.	80 min.	80 min.
m1.medium	$0.00248	50 min	55 min	40 min
m1.large	$0.00498	40 min	25 min	20 min
m1.xlarge	$0.00997	25 min	11 min	10 min

Another reason for using performance prediction is for optimising the resource use. Knowing the behaviour of an application may help improve the process of allocating the task as the prediction can determine when the resource is free and the next tasks in the queue can be allocated. Also, an accurate prediction is needed to avoid any conflict when sharing the resource between two applications which can lead to a decrease in the QoS [26].

3.6.2 Performance prediction approaches

The previous section discussed the reasons for using performance prediction. There are several studies that use one of the performance prediction approaches. One of the approaches is a proactive approach, which uses prediction to scale the application. Previous studies have considered using this approach but they have used different techniques for their models. However, one of the reasons researchers opt for a proactive approach is its fast reaction to requests by cloud platforms when there is are variable resource needs. For instance, Calheiros et al. [76] use an ARIMA model and their focus is on request patterns where they try to accurately predict the number of future requests by the user. Their work is designed for the short-term so the predictions are quicker. In order to update the model on the fly they apply feedback from the latest observed loads.

On the other hand, Kim et al. in [6] take a proactive approach using local linear regression to improve the cost and performance. There are several strategies that the system uses, namely an accurate and dynamic task execution time predictor, a resource evaluation scheme that balances cost and performance, and an availability-aware task scheduling algorithm. The prediction module has additional sub-components. These include the LLR predictor, which estimates the execution time for an input task on different types of VM. And the task history repository, which stores predicted and real time execution results and provides samples to estimate the task execution time.

Nadeem et al. [77] propose a novel method based on a neural network to predict the workflow execution time in the grid to benefit from the available resources. They apply principal component analysis (PCA) to eliminate the less important information and make the prediction more accurate. The prediction model collects the actual performance observed from executing training applications on a set of cloud configurations provided by the cloud providers. The training application they consider for this model is the NAS parallel benchmarks and the cloud configuration parameters used in the study are the number of nodes, the number of cores per node, and the amount of RAM per node. The model is then released to the user who can query it for performance predictions for the target application.

A radial basis function neural network (RBF-NN) is used in their model. The RBF-NN consists of three-layered neural networks. The input layer is the first layer and takes a set of workflow attribute values as input to the network, such as dependencies, problem-size, grid-sites, scheduling policy, and grid-site states. In the second layer each neuron i calculates the Euclidean distance d_i between an input set A_i and the centre ($d_i = ||A_i - c||$) applying the Gaussian function as a radial function. Therefore, there are weights w_i resulting from the second layer which are passed to the third layer. The third layer is the output layer, which communicates with a hidden layer through weights. Then, the layer approximates the function as a linear combination of neurons [77]. One of the drawbacks of RBF-NN is in trying to distinguish which attribute is less important.

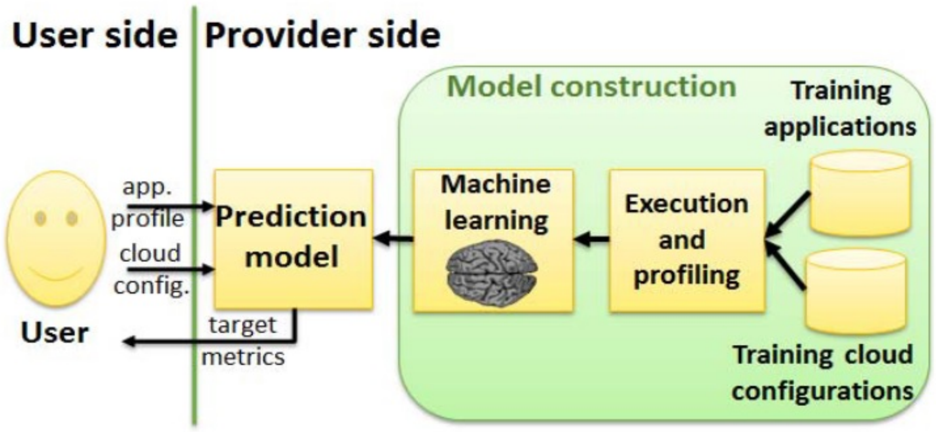


FIGURE 3.8: Overview of the proposed methodology. Extracted from [3].

Mariani et al. in [3] also consider applying random forests (RFs) as one of the machine learning techniques to implement their prediction models. The RF approach is a collection of several regression trees where each tree is generated to fit the behaviour of the target performance metric using a randomly selected subset of training data. Therefore, their goal is to use a prediction model based on machine learning to help the user select the best cloud configuration for their application as shown in Figure 3.8, [3].

The prediction model takes a cloud configuration and application profile as input. Then the output of the model is the expected speed, execution time, and cost. In addition, the model collects the actual performance observed from executing training applications on a set of cloud configurations which are provided by the cloud providers. The cloud configuration parameters that Mariani et al. [3] consider are the number of nodes, the number of cores per node, and the amount of RAM per node. The model is then released to the user who can query it for performance predictions for their target application.

In cloud systems, failure is a concerning issue. Minimizing the consequences of failure and producing accurate predictions with enough latency remains a demanding research challenge as large-scale systems grow in scale and complexity. This involves the implementation of a proactive and effective failure management strategy aimed at reducing the impact of failure within the system. Mohammed et al. [78] proposed an effective technique for failure prediction using time series and machine learning approaches. Their goal was to create a model that could reliably anticipate the failure of systems and applications.

Within the performance prediction there is an online curve fitting which helps service providers to predict performance when the nature of the task is unknown, or partially known. Service providers endeavour to improve the quality of service by allocating sufficient resources to all tasks, minimising idle resources and attaining server-wide load balance. The only way to achieve this goal is to collect historical data and generic metrics such as CPU cycles which can predict performance at a given time and task. After a series of operational cycles, the historical data of the server may be used to model a function that can be used to predict performance. This method is known as online curve fitting, which is vital in cloud resource allocation due to the fluctuating nature of user requirements [79]

Online curve fitting begins with coming up with data point approximation functions. In this case, the typical performance limits for a given task are estimated to yield data points that can form a curve, expressed as a function. For instance, the data approximation function may be developed by estimating the ideal lower timing bound, best-case execution time (BCET), minimal observed execution time, maximal observed execution time, worst-case execution time and the upper timing bound. Plotting these estimated data points on a graph yields a curve. Then, the unknown data points may be estimated by fitting them on the curve [80].

Online curve fitting is applicable in performance analysis because the data collected from a server at different loads represent a form of the continuous differentiable surface with a known structure and the constant factors can be predicted as variables in the model. Michael and Lily present a new approach based on extreme learning machine (ELM) and particle swarm optimization (PSO). In their study, a neural network-based approach is proposed to address the issue of curve fitting, one of the classic numerical analysis issues. This can be understood as an alternate strategy based on advanced learning that opposes the numerical solution analysis method. The benefits of both the PSO optimization approach and the ELM algorithm are incorporated into their method [81].

3.6.3 Prediction evaluation metrics

Once the prediction model has finished processing data and provided the prediction value, it is essential to evaluate the model to find out its accuracy either by using the success rate or by using error metrics.

As accuracy is one of the evaluation metrics for a prediction model, several papers outline different ways of calculating accuracy. For example, in [82] the authors use two metrics to identify the accuracy of the prediction. One is calculating the success rate. The success rate refers to the number of accurate predictions compared to the total number of predictions. Furthermore, sometimes there is a difference between the actual value and the predicted value. Therefore, there are several formulas that can be used to find errors in prediction such as Mean Squared Error (MSE) and Root Mean Squared Error (RMSE).

Cost and profit can also be used for the evaluation of a prediction. As Jiang et al. [83] discuss Cloud Prediction Cost (CPC) to calculate the cost of prediction error which can be of two types, the cost of the SLA violation, and the cost of resources in idle mode.

In brief, the evaluation of the prediction model is essential to ensure that the accuracy of the model meets the QoS and does not violate the SLA agreement.

3.7 Resource allocation and scheduling

Cloud computing has become a common approach for processing and executing applications on a pay-as-you-go basis. As the demand for cloud-based applications grows, it is becoming more challenging to allocate resources effectively based on user requests while still meeting the SLA between providers and users. Furthermore, resource heterogeneity, the unexpected nature of demand, and the diverse goals of cloud users make resource allocation in the cloud environment even more challenging. As a result, both researchers and professionals have begun significant research efforts to efficiently tackle the multiple issues associated with cloud resource allocation [8].

Therefore, in order to allocate resources effectively and meet the QoS, there are numerous mechanisms that can be used to perform the allocation and the scheduling [5] and [84]. One of the ways of classifying resource allocation approaches is based on optimisation objectives (such as the QoS objective) or the pricing objective.

QoS objective:

Under the QoS objective there are several resource allocation approaches such as QoS-based, priority-based, and guaranteed admission control. The QoS-based approach allocates the resources that are available for a given period of time using a discovering intermittently available resources (DIAR) algorithm. Huang and Venkatasubramanian [85] use the DIAR algorithm in a multimedia environment. Because their target is a multimedia environment, their work focuses on providing continuous time when allocating the task. Therefore, limited network bandwidth on the actual server might have an effect on the continuity of the allocation. Even when using multiple servers during the process, if they do not provide continuous allocation, the user might suffer poor QoS. The parameters they consider as an input are, processor use, storage capacity and bandwidth, requested video object identifier, start and finish time of the request, and the request type [85].

Along with a QoS-based approach, a priority-based approach is also considered to be under the QoS objectives. As shown in Figure 3.9, there are two stages within the priority-based approach. The first stage of priority-based approach is that the dispatcher receives applications from users and then sends them to a pool of priorities. In the second stage the resource allocation algorithms schedule the applications using an objective function and allocate resources which support the least resource demanding applications. The targeted resources are programs that are shareable between the applications, hard disks, printers, and memory. The resource allocation looks into the priorities from high to low and allocates the resources to the application. Then, an objective function is used to evaluate the next application in terms of its cost. In terms of allocating the resources, it uses Dijkstra's algorithm which finds the maximum bandwidth path between the resource allocation and the target [4].

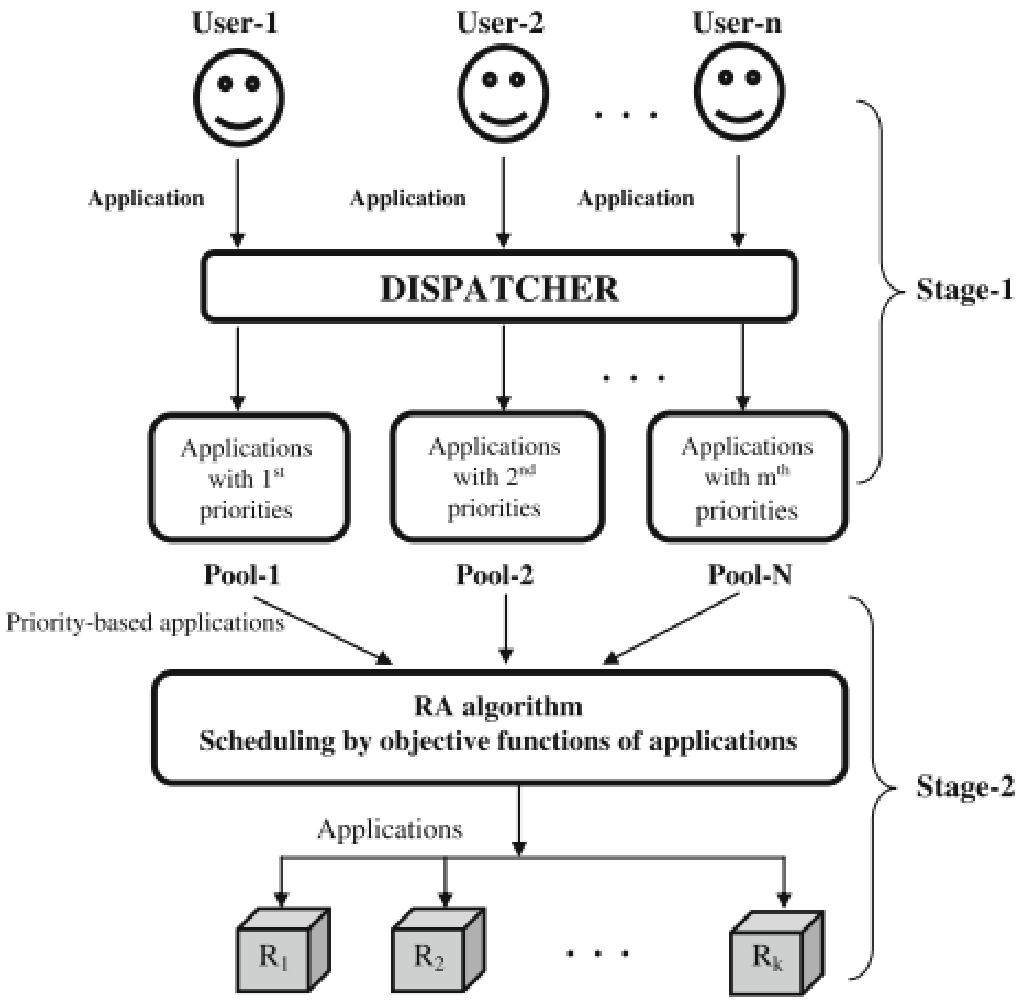

User-1
User-2
User-n
Application
Application
Application
DISPATCHER
Stage-1
Applications with 1st priorities
Applications with 2nd priorities
Applications with mth priorities
Pool-1
Pool-2
Pool-N
Priority-based applications
RA algorithm
Scheduling by objective functions of applications
Stage-2
Applications
R_1
R_2
R_k

Figure 3.9: Priority-based resource allocation. Extracted from [4].

Singh et al. [5] survey different resource allocation strategies. One of the approaches covered is guaranteed admission control which considers two factors when admitting a task to be scheduled. These are task execution time and deadline which are related to the QoS objective. This approach ensures that all applications admitted to a system will meet their respective deadlines without forcing other running applications to miss theirs. Figure 3.10 shows an analysis of WCET against deadline. If the WCET of the application and its task is less than the deadline, the application is considered to be schedulable and can be admitted and allocated to the right core to be executed as shown in Figure 3.10. Otherwise, the application is not schedulable and the allocation will not proceed.

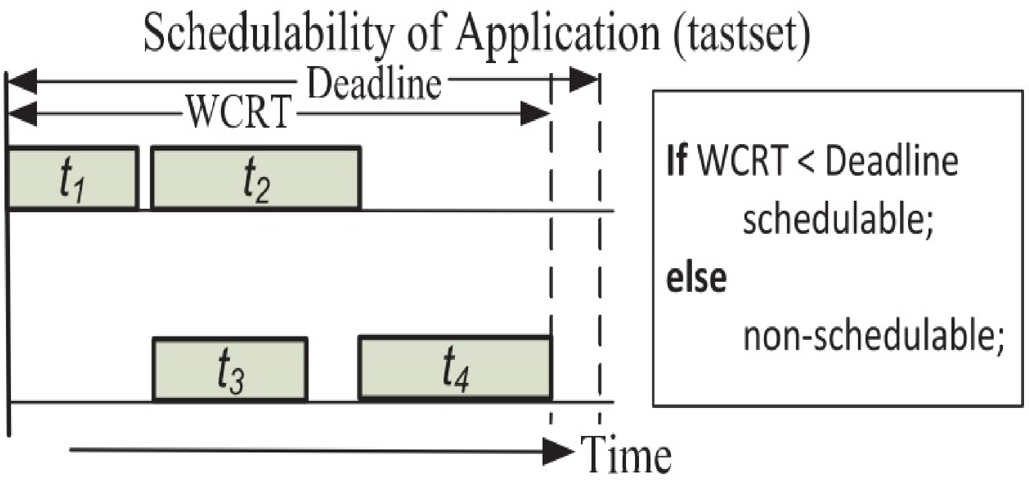


FIGURE 3.10: Guaranteed admission control. Extracted from [5].

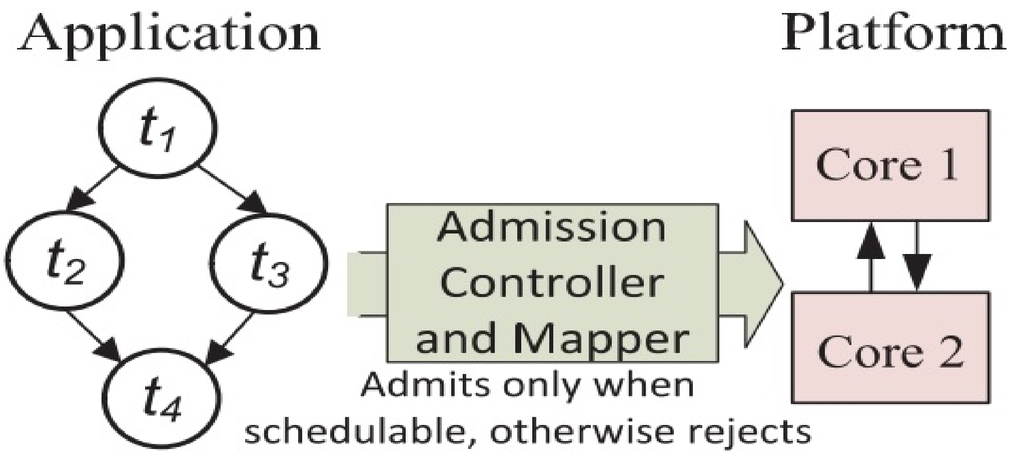

Application
Platform
t_1
t_2
t_3
t_4
Admission Controller and Mapper
Admits only when schedulable, otherwise rejects
Core 1
Core 2

FIGURE 3.11: Guaranteed admission control. Extracted from [5].

Multi-processor real-time scheduling entails designing a system with multiple CPUs to share a load or a set of tasks in a distributed way. This schedule may be achieved through symmetric or asymmetric scheduling. Asymmetric scheduling in a cloud infrastructure would entail a master processor that distributes, or assigns all I/O tasks to other processors in the same VM. The other processors only will execute the user code but deliver outputs in significantly faster timelines. This mode of scheduling is ideal for a single-user setting, such as a dedicated cloud server for a given organization [86].

In real-time system, three different real-time system types can be distinguished depending on the consequences of missing a deadline. In a hard real-time scheduling, any deadline that is missed is considered as a system failure. This scheduling is widely utilized in engineering or health systems where failure to meet timing requirements results in the loss of lives or products. In firm scheduling, it does not compromise the proper behavior of the system yet the task's late completion is useless. Whereas in soft scheduling, it is possible to often miss deadlines, and as long as tasks are completed on time, the results are still valuable. Up until the deadline, completed tasks could be worth more and less as time goes on [87].

Dziurzanski and Singh in [88] have considered a firm real-time scheduling in their work. They proposed a DVFS-based (dynamic voltage and frequency scaling) feedback control technique for a firm task allocation on Multiprocessor Systems-on-Chips (MPSoC) systems to increase energy efficiency. The newly created admission control algorithm performs a schedulability analysis taking into account the states of the prior platform and rejects the task that are expected to miss their deadlines.

According to Qureshi et al. in [4], a swift scheduling mechanism is a dynamic scheduling technique that combines the advantages of heuristics approach. This approach is used when it cannot reach an optimal solution and thus uses the shortest job first (SJF) method which finds the smallest execution time and higher priority in scheduling. The swift scheduling takes the incoming tasks into queue. Using a heuristic function, the scheduling technique selects a task from the queue. Reducing the waiting time in the queue is the main goal of the technique. As it mentioned in the paper that swift scheduling shows a good result with minimum cost and time comparison with first-come first-serve and shortest job first methods.

There are numerous studies on managing GA execution in the cloud for example [89] and [90]. In an industrial domain scheduling a new request as soon as possible is an important step to achieve the QoS required by the user. Thus, Shuai et al. [89] propose an approach that allows a multi-objective GA to be executed on multiple sub-populations (islands). Their goal is to increase responsiveness and profit when a new manufacturing order arrives or there is a change in the factory state. Therefore, the research considers real-world smart factories. Salza and Ferruci [90] propose an approach in distributing GAs by using a master-slave model where the master places the individuals in a request queue which then distributes them in a round-robin fashion to the slave nodes. Once the slaves have finished processing the individuals, they place them in a response queue and return to the master node.

Devarasetty and Reddy in [91] introduce an improved resource allocation optimization technique by using a GA that takes into account the goals of reducing deployment costs and increasing QoS performance. The presented algorithm takes into account various user QoS requirements and allocates resources within the budget constraints. Total time and maximum budget for deployment of application are examples of users' requirements. In the chromosome representation, they consider user QoS demands as well as available resources. Thus, the input parameter to the GA model is the users' QoS requirements and the output result of the GA model is a suitable resource for the user.

Pricing Objective:

The hybrid resource allocation technique is proposed by Shah et al. in [92]. This technique allocates grid resources by combining DLT and least cost method (LCM) in a way as to minimise the total computational cost. Specifically, hybrid resource allocation divides the tasks into equal sized portions and allocates them to the processing resources using the LCM technique.

Several approaches are considered under-pricing and cost objectives, such as hybrid resource allocation, market-based, and combinatorial auction-based. Hybrid resource allocation combines two methods, divisible load theory (DLT), and least cost method (LCM). The hybrid approach goal is to minimise the computational cost by using DLT to divide the tasks into equal sized tasks and then allocate them using LCM.

As resource allocation varies with the optimisation objective, the market-based approach focuses on pricing between the user and the provider. One study that uses a market-based approach is by Singh et al. [93]. They use an auction on a many-core system. The resource manager can receive bids from the cores and those that offer the highest bids are allocated the jobs that have the highest value. Their goal is to maximise the overall value returned from the system.

3.8 Optimization problem models and solvers

Over the last two decades, the optimization field has experienced remarkable expansion. To address a wide range of issues in engineering and management, several novel theoretical, algorithmic, and computational contributions to optimization have been proposed. The most effective approach to allocate limited resources to accomplish particular goals is determined by organisations using optimization models, which are mathematical models. These models can be used for a wide range of issues, from finding out the best path for a delivery truck to discovering the best combination of goods to stock in a store [94].

Although there are many different optimization model types, they all share a few essential characteristics. A collection of decision variables representing different possibilities will be present in an optimization model at the beginning. The decision variables in a model, for instance, can represent the various routes that could be taken in order to determine the most effective delivery service. Second, a set of constraints that specify the upper and lower bounds for the decision variables will be present in an optimization model. A limitation might specify, for instance, that a delivery truck is only permitted to drive on particular routes or that it must complete all of its deliveries within a specific window of time [95].

An optimization model will also include an objective function that specifies the goal it is seeking to achieve. The objective function in the delivery truck example might be to reduce the overall distance travelled. Though optimization models are useful for a wide range of issues, they work particularly well for issues with a variety of decision variables and constraints. Because of this, optimization models are frequently employed in industrial engineering and other disciplines that address challenging decision-making issues [96].

Genetic algorithms have been applied to optimize resource allocation by scheduling to adapt the task requirements to the available memory in a cloud computing system. Zhao et al. in [97] designed an optimized genetic algorithm to schedule divisible and independent tasks as a function of the computation memory requirements. The outcomes indicate improvement in homogenous systems but no significant improvement in heterogeneous systems. A similar challenge was noted in a nonlinear programming problem, modelled with a mathematical programming language (AMPL) and aiming at optimizing task scheduling on multiple clouds based on a minimal cost constraint [98].

3.9 Summary

From the literature review, different domains within resource management show that information from other components plays a key role in allocating the right resources for the task. This starts with workload and workload deployment which can assist during system testing. In order to avoid using a sensitive data in workload a workload model can be used to extract the workload information so that the experiment can be carried out in a more controlled environment to allow more experiments to be done without the concern of having crucial business data. One of the container-based technologies that is used to deploy the workload is a Docker container along with Docker swarm and Kubernetes as an orchestration system to manage task deployment. Both, Docker swarm and Kubernetes are considered to be one of the state-of-the-art container-based systems [28]. Therefore, we consider both of them as the baseline for our work as it will help can compare our model against it.

Within Docker swarm, a spread strategy is used to allocate tasks to one of the nodes based on monitoring information from the nodes which refers to the number of containers per node. In order to avoid over or under provisioning of resources, workload profiling can be used to determine the resources needed for the current task. In addition, information from the platform about resource use can help determine whether to scale resources up or down as it can reduce the cost of the running of a new machine.

Among the other areas of resource management, resource allocation has a variety of objectives that a provider can determine when allocating resources. Using market-based resource allocation can maximise the revenue of the service provider as it chooses the highest bid. On the other hand, admission control can be used to manage task execution and one of the criteria when admitting or rejecting a task is based on whether the response time is less than or equal to the deadline.

Based on the investigations that have been reviewed, improving the resource management is an important stage in increasing the QoS performance and one of the common metrics is deadline. Most of the works consider deadline and cost as two features for improving the resource management. However, in our work we consider deadline and fitness achieved from the GA as two main features. The aim of our work is to obtain for as many instances as possible, a GA output which achieves a user-defined fitness level by a user-defined deadline.

Chapter 4

Experimental platform, metrics and methods

A real application is required to assist in evaluating resource management presented in this thesis to test the hypotheses previously mentioned in section 1.4. Therefore, this chapter will provide deep knowledge of the applications used in this thesis (mentioned in section 1.2). Once detailed knowledge of the applications is provided, an approach will be presented. Further, a detailed understanding will be gained for handling the GA application within container-based technology, including different parameters the approach received as an input to execute that GA application.

Several parameters can be received as input to a GA application, such as the number of generations, the number of populations, and the application-specific parameters. Application-specific parameters are used for different optimisation problems, such as solving a Sudoku puzzle, where the GA application receives the level of the Sudoku puzzle's difficulty. The resource manager must receive the actual deadline and the user-defined fitness level to compare them after each iteration for the approach to check whether the task has achieved the user-defined fitness level and reached the deadline.

Traditionally, The GA application can be executed using Docker Swarm or Kubernetes since they are considered state-of-the-art container-based orchestration systems. However, we proposed an orchestrator that resource management can use to handle the GA application. Concerning the infrastructure to be used to deploy the orchestrator, we used Amazon AWS as one of the cloud providers to deploy the orchestrator and evaluate resource management.

4.1 Encapsulation of genetic algorithm

We created an approach that containerises the GA so that the resource manager, part of the orchestrator, could instantiate containers to execute the GA application to manage different GAs, and its parameters passed on the user requirement. This is illustrated in Figure 4.1, showing the inputs and outputs of the GA that can be executed in a container. One of the input parameters is a set of application-specific parameters. It is used when the GA application solves a specific application. For example, the GA application is used to solve the Sudoku puzzle. The application must receive the Sudoku puzzle's difficulty level for the GA application to solve that puzzle. Then, the application will generate the puzzle based on that level. The other two parameters (i.e., the initial population and the number of generations) are used for the GA application to start the process of finding a suitable solution based on the given problem.

A Docker image was created to execute the GA inside a container, which contained the actual GA application and allowed the image to receive any inputs that might be used when a container is instantiated. After the container is executed, the output of the GA application will be processed. The final population will contain a set of solutions that were determined to contain the best solution for the problem that the GA tried to solve. Each individual of the final population has a fitness value to define how good a solution is within the population.

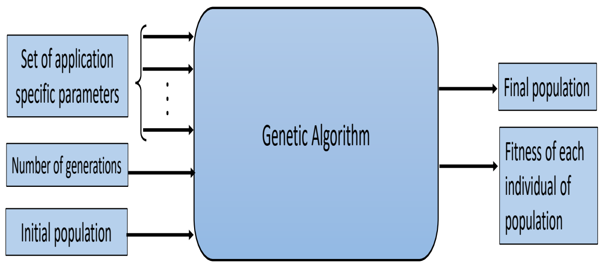

Set of application specific parameters
Number of generations
Initial population
Genetic Algorithm
Final population
Fitness of each individual of population

Figure 4.1: Inputs and outputs of the GA.

4.2 Application model

The application model can formally describe the work that needs to be done by the system. Generally, a workload consists of several jobs, and each job consists of several tasks. In our work, we consider a workload as consisting of one job and numerous independent tasks. Further, we consider real-time tasks based on the case studies presented in section 1.2: (i) allocating real-time tasks to a multiprocessor system and (ii) Sudoku puzzles. Thus, each task contains several parameters. Some of which are application-specific. Meanwhile, others are related to the GA, such as the number of generations and the initial population. Each of these tasks was a single unit without dependencies.

Beyond the number of generations and the initial population, two parameters related to the first case study are the X-dimension and Y-dimension. These parameters are the number of cores in a multiprocessors system. For example, if X and Y are 3 by 4 (3 X 4), the system represents the core as a grid that is 3 by 4. In addition to these parameters, there is Navs, which contains the number of real-time tasks to be allocated into the multiprocessors system. On the other hand, the second case study receives only one specific parameter, which is the probability besides the GA parameters. This parameter is the level of the Sudoku puzzle's difficulty. The higher the level of the probability is, the more difficult the puzzle will be. Once the GA application receives the parameter, it will generate the puzzle to be solved.

Several parameters are common between the case studies to distinguish each task in both case studies. One of them is a unique task ID so that the result of that task can be clearly read and evaluated. Other parameters that exist in both case studies are the fitness required and the deadline in seconds. Both parameters are fixed values and generated for the resource management to execute the task and achieve the fitness required by the deadline. The tasks the resource management handles do not arrive at the same time but one after the other. Thus, the last parameter is the task waiting time. This parameter specifies the time that the tasks need to wait before it sends to resource management.

4.3 Proposed orchestrator

In the proposed approach, the orchestrator consists of several components dedicated to managing the deployment of the GA workload. This is illustrated in Figure 4.2. Every component of the orchestrator has its own roles and responsibilities in handling the information related to the task or the node, such as the number of tasks in a node, tracking the tasks' response times, and the fitness achieved. First, the client can submit numerous GA tasks, $T=\{t_1, t_2, ..., t_c\}$, which each provide a set of application-specific parameters, the fitness required, and the deadline. The tasks arrive randomly every S seconds. Second, once the resource allocation component receives the task, it requests information from the node observer component about the cluster. This information might be the number of tasks in a node, CPU utilisation, or others that the resource allocation algorithm needs to allocate the incoming task to a suitable node.

The node observer receives updated information about the cluster from the Docker manager. This helps make the right decision when allocating the task to a node. Once the resource allocation module allocates the task to a node, the Docker manager handles the execution of the task on the specified node and collects the results. Thus, the Docker manager has a set of n nodes $N=\{N_1, N_2, ..., N_n\}$ that execute the task in a Docker container based on the tasks' information and each node created as a Docker machine.

In this thesis, as the resource allocation module sets the number of generations and initial population for the GA, each of the following chapters has its own assumption of the tasks and the allocation technique. They will be introduced in each chapter.

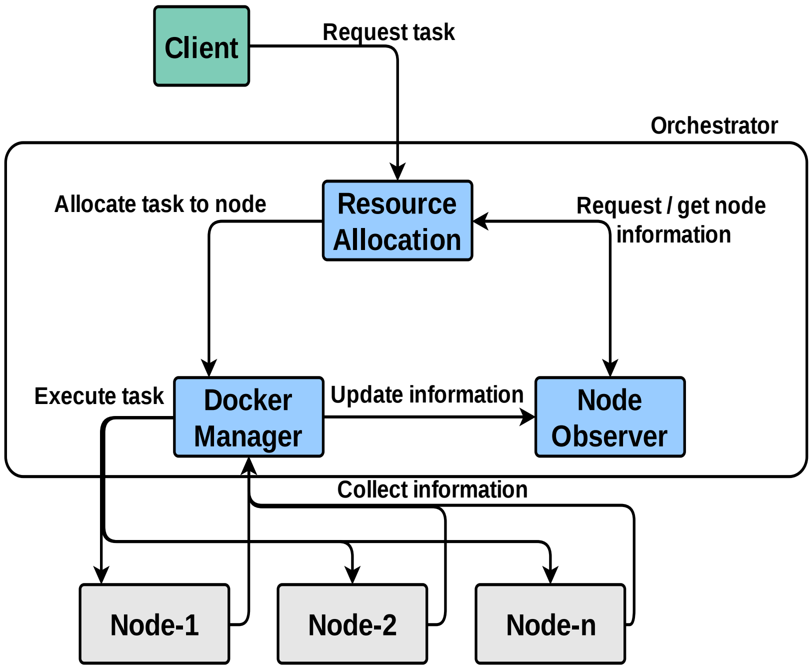


FIGURE 4.2: Proposed orchestrator.

4.4 Metrics

In this thesis, we consider a hard deadline and achieving the user-defined fitness level. Thus, resource management aims to maximise the number of tasks executed on time and achieve the fitness required. Therefore, this section is related to some observed properties of the workload when the orchestrator executes it. The resource allocation policies used may affect the tasks' results on the response time, or the fitness achieved. Moreover, the metrics used in this section can be employed in a resource management approach and for comparing and evaluating different resource management policies.

Complete task response time:

As the tasks may go through several iterations during their execution, the complete response time of a task is calculated from the arrival of the task until the end of its execution when the task terminates.

Execution of a single iteration:

The execution of a single iteration is calculated from when it arrived until the end of its execution at that iteration. This is because tasks may go through multiple iterations throughout their execution.

Task waiting time:

The task waiting time is the time from the arrival of the task until the resource allocation allocates one of the available nodes to it.

Fitness achieved:

Once the resource allocation allocates the task to a node, the Docker manager will handle the execution of the task. Once the task is finished running the iteration or it completely finished the execution, the result of the task will be collected. One of the results collected concerns the fitness achieved, which can determine whether the task has met the user-defined fitness level or not.

Number of successful tasks:

Once the tasks finish executing and the results are collected, a task is considered successful if the complete response time is less than or equal to the user-defined deadline. Further, the fitness achieved must be greater than or equal to the user-defined fitness level.

4.5 Experimental method

We considered the Docker Swarm and Kubernetes as the baselines to compare the proposed approaches for making a fair comparison. The baselines and proposed approach are assessed at the same load level to ensure a fair comparison.

The reason for choosing these baselines is that they are considered state-of-the-art container-based technology. Thus, we created a cluster of 12 nodes in the baselines and approaches. Further, the baselines and approaches will receive the incoming task randomly every S second (i.e., 10–20 seconds). The workloads are based on the case studies mentioned in section 1.2, where we generated 20 workloads (10 workloads for each case study), and each workload contains 150 tasks. Testing the approaches and the baselines with different workloads will help provide a greater understanding of the result during the evaluation. Further, it will help in quantitatively analysing the approaches.

Once the incoming task is received by either the master node (from the baseline) or resource allocation (from the orchestrator), one of the allocation policies will allocate the task to a suitable node (e.g., the spread strategy in Docker Swarm). In this thesis, we proposed several allocation policies to handle the allocation of incoming tasks. Once the task has finished the execution, the result is written in a log file, and the container is removed.

In our work, we conducted a series of experiments to assess our approaches. AWS EC2 instances (type t2.micro) used for these experiments have 1 VCPU and 1 GiB memory, and the operating system is ubuntu. The first step is to create two instances of t2.micro. This instance is for the user to send the task and the other instance is for the resource manager to receive and allocate the tasks.

On the resource manager instance, we need to install the Docker engine to create a Docker machine and handle the execution of the tasks. Once the installation is done, then the experiments presented in this research can be reproduced by following the steps below:

1. In the resource allocation java file ("ResourceAllocation.java"), we need to choose the approach that needs to be used in the experiment.

2. Run command
   java -cp .:lib/* Orchestrator.MainProcess
   in the terminal to start the resource manager which then starts by creating the nodes (Docker machines) that will be used in the experiment. After each node creation, the applications will be downloaded using "ConfigNode.sh" which handles the downloading on each node.

3. Once step two is done, on the user instance we need to run the command
   java Userpart.Client

which will connect to the resource manager instance and start sending the tasks.

All the required files along with detailed instructions to reproduce the results, are available at https://gitlab.com/ta835/resource-management

4.6 Summary

In conclusion, this chapter introduces a key point that the rest of the thesis depends on. An encapsulation of the GA is introduced so that resource management can instantiate as many Docker containers as needed. Each Docker container can have a different configuration based on this thesis's task information and case studies. Aside from the baselines, the approaches presented in this thesis will use the proposed orchestrator implemented using Java. We generated 20 workloads for both case studies to evaluate the approaches and compare them against the baselines. We collected two metrics to be used in the evaluation during the experiments: the response time and the fitness achieved.

Chapter 5

Management of container-based genetic algorithm workloads over cloud infrastructure

Cloud infrastructure has been widely used to support engineering applications. The fast and efficient execution of software tools can contribute to prompt responses and solutions to engineering problems. One specific type of load that often appears as part of engineering applications is optimisation. In many cases, optimisation software uses meta-heuristics such as GAs [13]. GAs provide optimisation solutions in various domains, such as smart factories [14] and embedded multiprocessors [15]. The number of generations, the initial population, and a set of application-specific parameters are examples of parameters that can be input for a GA. Additionally, a solution with the desired fitness for a specific problem is often necessary, with this fitness fulfilled by a given deadline. A prediction feature can be used to determine the fitness by a given deadline when executing the GA workload. Therefore, this chapter proposed two approaches to investigate this hypothesis.

In a situation where a container orchestrator (Docker Swarm or Kubernetes) has been used to deploy and manage the execution of the application, the resource scheduling process is part of the orchestrator and decides where to allocate the task. Thus, container-based orchestration systems allow applications to be executed in shared resources with fast and flexible deployment. One platform that can deploy workloads is Docker, an abstraction to help organise the workload, hide the details and deploy the application in an isolated environment [27]. Docker Swarm and Kubernetes are considered state-of-the-art container-based orchestration systems [28].

Improving resource management is considered an essential part of managing a specific GA workload. This allows the task to meet the deadline and achieve the fitness level required by the user. We assume that only one task is executed at a time per cloud node in this chapter. A queue can reserve the tasks until enough resources are available to process them. Therefore, this chapter proposes a novel approach to manage different GA workloads and compare them against the baselines. Additionally, the chapter examines how well this approach improved the number of tasks executed on time and whether the tasks achieved the required fitness level. There is an iteration process when executing the task. Therefore, we considered iteration analyses. In addition, because performance prediction has been used in this chapter, an evaluation technique must be used to determine the efficiency of the approach. One of the techniques for evaluating the prediction approach's effectiveness is to use a prediction error analysis, which will be done to evaluate the prediction approaches.

5.1 Comparing different allocation techniques

Based on the different models mentioned previously, the resource allocation module receives the incoming tasks from the client, and it requests from the node observer the node that is available to handle that task. Based on this information (i.e., the incoming task like deadline and the information from the cloud platform like number of task in a node), resource allocation can decide on which node a task is submitted for execution. Various allocation decisions are possible. This chapter proposes several approaches and compares them to the Docker Swarm spread strategy and Kubernetes as baselines. Additionally, for the baselines, we evaluate the following approaches: fitness tracking (FT), fitness prediction (FP) and fitness prediction-based linear regression (FPLR).

5.1.1 Fitness tracking (FT):

This approach aims to keep track of the achieved fitness and compare it with the user-defined fitness requirement. Additionally, the approach aims to improve the number of tasks, which *a*) are executed on time and *b*) achieve achieve the fitness required by the user.Thus, the task will continue executing tracking until it reaches the deadline or has achieved the user-defined fitness requirement.

Furthermore, the task goes through several iterations. It has a fixed number of generations at each iteration to track the time taken to execute the task, and the fitness achieved, as illustrated in Figure 5.1. Once the task is received, either an available node will execute the task or it is placed in a queue. After the task is allocated to a node, the task starts at the initial iteration and stores the time it takes, and the fitness achieved. It further sets the maximum number of iterations that the task can go through. At each iteration, the task will continue to execute for a fixed number of generations using the best results from the previous execution. This is because every time we execute the GA application, it gives similar or better results than the previous time. This process continues until the task reaches the deadline, the fitness achieved is higher than the required fitness, or it reaches the maximum number of iterations.

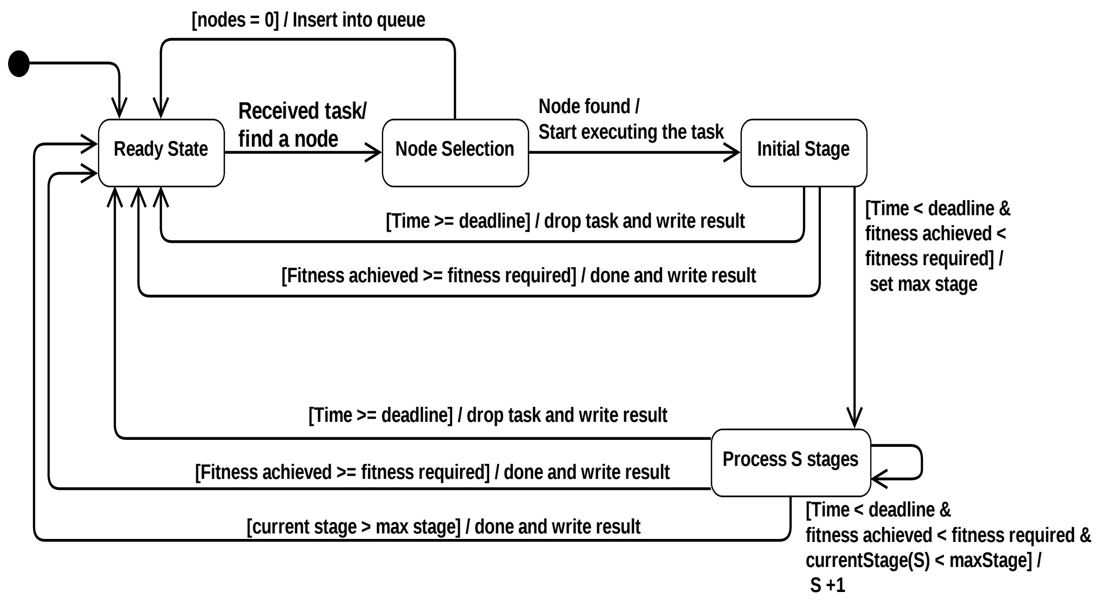

[nodes = 0] / Insert into queue
Received task/ find a node
Node found / Start executing the task
Ready State
Node Selection
Initial Stage
[Time >= deadline] / drop task and write result
[Fitness achieved >= fitness required] / done and write result
[Time < deadline & fitness achieved < fitness required] / set max stage
[Time >= deadline] / drop task and write result
Process S stages
[Fitness achieved >= fitness required] / done and write result
[current stage > max stage] / done and write result
[Time < deadline & fitness achieved < fitness required & currentStage(S) < maxStage] / S +1

FIGURE 5.1: State machine of fitness tracking approach (FT).

5.1.2 Fitness prediction (FP):

In FP, although we are executing a fixed number of generations at each iteration, we use a polynomial prediction to predict the fitness achieved at a given time similarly to FT. In this section, we assume a second-degree polynomial prediction model. The prediction is used to find the relationship between the independent variable (time taken) and the dependent variable (fitness achieved).

Predicting the fitness value of a problem-solving process based on a given deadline is the aim of fitness-required forecasting for quadratic cases. The mathematical model for the quadratic case is illustrated in equation 5.1. As the new data points become available when running the approach, the coefficient values become updated and then can predict the fitness required at a given deadline.

$$F(t) = C + m1 \times t + m2 \times t^2 \tag{5.1}$$

Where:

- *F(t)* is the fitness value at time t which means at a given deadline.

- C is the intercept.

- *m1* is the coefficient associated with the linear term, indicating the rate of change of fitness with respect to time.

- *m2* is the coefficient associated with the quadratic term, capturing the curvature of the fitness curve over time.

In this approach, we use the prediction to predict the fitness achieved by the deadline. Additionally, for the FT approach conditions, each iteration will check whether the fitness predicted is better than or equal to the fitness required based on the previously observed current task data. Based on Figure 5.1, we set the predicted fitness as the fitness required to go to the next iteration and then collect more points that can be used in the prediction. After the second iteration, the prediction value is updated based on the observation points from the first and second iterations.

5.1.3 Fitness-prediction-based linear regression (FPLR):

The prediction used in the previous approach (FP) was a polynomial prediction based on the observation data as the task goes through the iterations. In doing this, all observation data collected from each iteration are considered when updating the fitness prediction. Concerning the fitness prediction for FPLR, a linear regression prediction is used to forecast the fitness achieved at a given time. The FPLR approach uses predictions based on the last two observed data points to predict the fitness for the next iteration.

Although the relationship between the dependent and independent variables is best approximated using a polynomial prediction, one or two outliers in the data might significantly impact the result of the nonlinear analysis On the other hand, using the linear regression approach is less complex concerning its implementation and provides satisfactory outcomes. On the negative side, the linear regression assumes that the relationship between the dependent and independent is a straight line. Further, linear regression is too simple to describe the complexities of the real world.

$$F(t) = C + m1 \times t \tag{5.2}$$

We are interested in predicting the fitness required of a problem-solving process based on the computing time (t) spent on the problem in the context of fitness function forecasting in a linear case. We apply a basic linear model as illustrated in equation 5.2. Where:

- *F(t)* is the fitness value at time t which means in our case at a given deadline.

- C is the intercept.

- *m1* is the coefficient that represents how the fitness value changes with respect to computation time t.

We can use a technique like linear regression to determine the values of coefficients C and *m1*. During the problem-solving process, the technique will collect data points which result in updating the estimated value for C and *m1* iteratively.

Thus, each iteration checks whether the fitness predicted is better than or equal to the fitness required based on previously observed current task data. As illustrated in Figure 5.1, we set the predicted fitness as the fitness required to go to the next iteration, with additional data observations collected for the prediction. In processing the iterations, an additional condition is also used (the fitness prediction) to determine whether the task can achieve the required fitness by the deadline. After the second iteration, the prediction value is updated based on the observation points from the first and second iterations.

5.1.4 Fitness-prediction-based weighted least square curve fitting (FPWLS)

FPWLS uses one of the curve fitting approaches which is weighted least square to predict the fitness achieved at a particular time, much like FP. Similar to the FP, we assume a weighted least square of the second-degree polynomial prediction model in this section. As the name suggests, Weighted Least Square is a method for accounting for the influence of each data point by giving each data point the appropriate amount of weight in relation to the parameter estimate.

Like other Least Squares, weighted Least Squares is an effective technique that uses weights that are inversely related to the variance. It is similar in that it can offer many kinds of simple-to-understand statistical intervals for estimation, prediction and optimization. One of the benefits of weighted least squares is its capacity to handle regression scenarios where the quality of the data points varies. Another benefit, it works well for getting the most information out of small data sets. The major disadvantage, it requires knowing the exact weights. Results from weight estimation might be unreliable, especially when working with small samples. In our work, each data point is considered to be important since we are dealing with a small data set.

In this method, we forecast the fitness achieved by the deadline using WLS. Similar to the previous approaches, the fitness predicted for the FT approach conditions will also be compared to the fitness required based on the previously observed current task data in each iteration to see if it is better or equal. We set the predicted fitness as the fitness necessary to proceed to the next iteration based on Figure 5.1 and then gather more data that can be used in the prediction.

5.2 Evaluation

This section will examine the different approaches and baselines concerning how well they improve task execution. First, an experimental setup will be discussed, followed by an experimental result. In the experimental result, there will be several evaluations of the result. For example, one could compare different approaches regarding the number of successful tasks in each approach. Since two approaches are used for the prediction, a performance prediction error will be used to compare their predictions. Then, an iteration analysis will be done to check the behaviour of the tasks when they go into an iteration process.

5.2.1 Experimental setup

In this experiment, we used the proposed orchestrator in Figure 4.2, as mentioned and explained in Chapter 4. Thus, we created a cluster of 12 nodes for both the baselines and the proposed approaches that could execute the incoming workload. As mentioned in Chapter 4, the user can submit a workload to be executed in one of the created nodes. Then, resource allocation can receive an incoming workload randomly every S second (i.e., 10–20 seconds) and allocate the task to the available node. In this chapter, we assume that the node can execute only one task per cloud node.

Since we have a deadline for managing the GA workload in this chapter, the experimental work aims to evaluate different approaches such as *FT, FP* and

FPLR and compare them against Docker Swarm and Kubernetes baselines. Additionally, the first experimental hypotheses (related to managing the GA workload) will be investigated in this section of the evaluation. During the evaluation, we consider two metrics: the response time and the fitness achieved. The importance of collecting these metrics is to find the number of successful tasks. In our work, we can say that the task is considered a successful task only when it achieves user the defined fitness level by the deadline.

Regarding the workload, we consider two specific kinds of GA workloads: allocating real-time tasks to a multiprocessor system and Sudoku puzzles. Thus, the GA is fed with problem-specific parameters needed for these case studies. The number and types of tasks to be allocated and interconnects of the multiprocessor are example parameters related to the real-time multiprocessor problem. However, the difficulty level of the puzzle is an example parameter related to the Sudoku puzzle.

When generating the previous workload, we generated real-time tasks with a task ID, x-dimension, y-dimension and navs. The x-dimension and y-dimension are the number of processors in a multiprocessor system, and navs contain the number of real-time tasks. Further, we generated the level of difficulty for the Sudoku puzzle optimisation problem. These parameters will be passed to the genetic algorithm for an application-specific domain. Additionally, we generated the fitness needed to be met for each task when the task finished executing and a fixed deadline in seconds.

Our experiment used Amazon Web Services EC2 instances to deploy our orchestrators and run the experiments. The node used was t2.micro (1 VCPU and 1GB memory), running on the Ubuntu Linux operating system. Since we have two main parts of the orchestrator (the user who sent the workload and the rest of the orchestrator), there were two EC2 nodes for each client and the orchestrator. Therefore, the client could send a task for the resource allocation to receive. We created additional 12 nodes to handle the execution of the tasks to have a fair comparison between the proposed approaches and the baselines.

5.2.2 Experimental results

We considered two case studies (real-time multiprocessors allocation and Sudoku puzzles) mentioned in section 1.2. In our work, we ran 10 experiments on each case study. Thus, we executed 150 tasks on the FT, FP, FPLR and FPWLS approaches in each experiment and the Docker Swarm and Kubernetes.

We collected two metrics: the response time and the fitness achieved. The results show that the approach we implemented (FPWLS) performed better than the FT, FP and FPLR approaches and the Docker Swarm and Kubernetes in the first case studies. In the second case study, the result of FPWLS shows that the approach pull-back compared with the other approaches. In more detail, the median observed results of FPWLS in the first case study were higher than the median observed results in the other approaches, as shown in Figure 5.2 and 5.3.

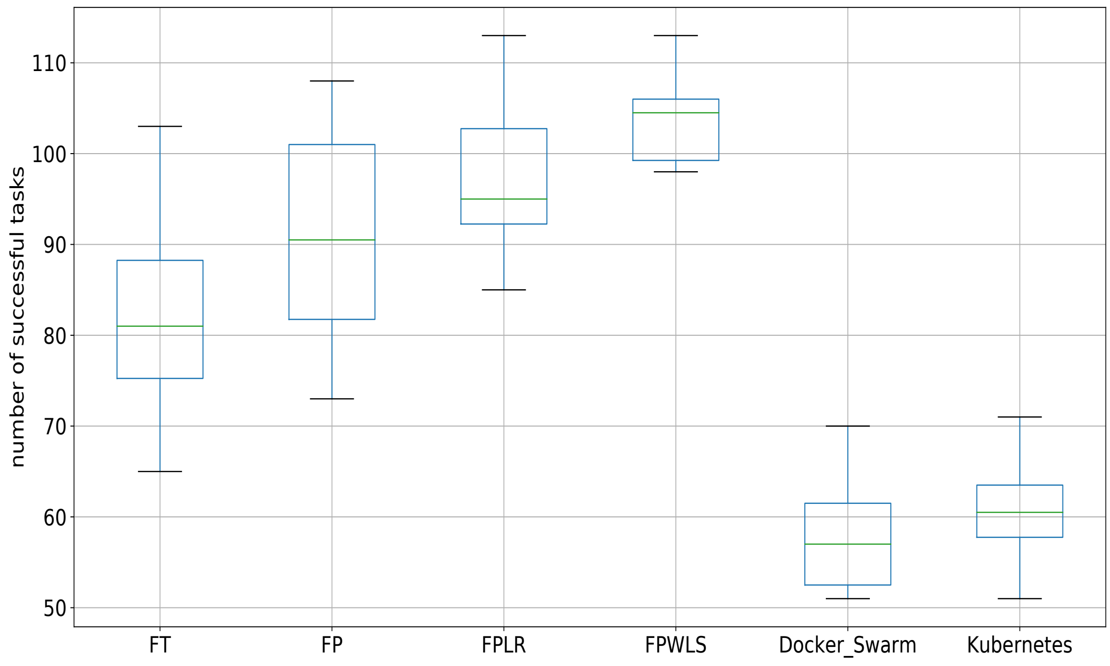


FIGURE 5.2: Results of 10 experiments of different approaches and different GA workloads (real-time multiprocessor allocation) concerning the number of successful tasks.

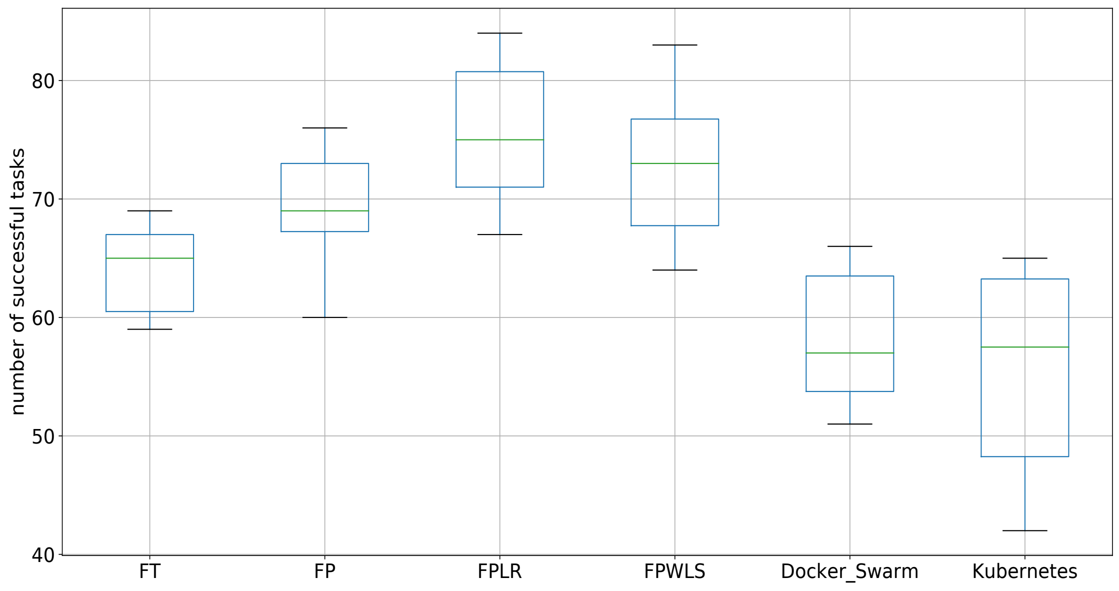


FIGURE 5.3: Results of 10 experiments of different approaches and different GA workloads (Sudoku puzzle) concerning the number of successful tasks.

As shown in Figure 5.2, the FT approach has 16% more successful tasks than the Docker Swarm and Kubernetes, whereas the FP approach has achieved 21% more successful tasks. In addition, the FPLR has achieved 26% more successful tasks. Further, the FPWLS achieved a higher percentage than FT, FP and FPLR, with 29% more successful tasks. Conversely, FPLR achieved a higher percentage of successful tasks with 13% than FP with 9%, FT with 5% and FPWLS with 11% more successful tasks (Figure5.3).

The reason for the results in the Docker Swarm and Kubernetes (state-of-the-art container-based orchestration systems) is that they both execute the task in a single execution. Therefore, the container executed and reported back the result, regardless of the condition. However, in FT, FP, FPLR and FPWLS, we forced the GA containers to run in iterations to keep track of and predict the fitness, considering the tasks' deadline. Once the results were obtained, we compared them with the desired results. Additionally, the approaches could tune the number of generations passed to the GA container to ensure they did not exceed the current task's deadline. Having these features over the Docker Swarm and Kubernetes allows us to achieve more successful tasks as shown in the previous results.

Approach	f-ratio	P-value
FT vs. Docker Swarm	39.72	p-value is $<$ .00001
FP vs. Docker Swarm	61.28	p-value is $<$ .00001
FPLR vs. Docker Swarm	138.90	p-value is $<$ .00001
FPWLS vs. Docker Swarm	174.64	p-value is $<$ .00001

Table 5.1: One-way ANOVA test comparing the FT, FP, FPLR and FPWLS approach against Docker Swarm in real-time multiprocessor allocation case study.

We compared the results using one-way analysis of variance (ANOVA) tests to statistically demonstrate the difference between the results obtained from the experimental approaches. As seen from the previous result that Docker Swarm and Kubernetes achieve similar results in both case studies, we use Docker Swarm to compare our results against it when analyzing the result using the ANOVA test.

Thus, the f-ratio is the variation between the approaches' mean values based on the f-ratio. The p-value can be obtained and compared to the significance level (0.01, 0.05, or 0.10). As illustrated in table 5.1, the result shows that the F-ratio is 39.72 when comparing FT against Docker Swarm whereas the p-value (p-value $<$ 0.00001) is less than the significance level of 0.05. Therefore, at the 95% confidence level, it can be observed that there is a significant difference among the other approaches when we compare them against Docker Swarm.

Approach	f-ratio	P-value
FT vs. Docker Swarm	7.81	p-value is .011949
FP vs. Docker Swarm	19.20	p-value is .00036.
FPLR vs. Docker Swarm	41.66	p-value is $<$ .00001
FPWLS vs. Docker Swarm	30.85	p-value is .000028

Table 5.2: One-way ANOVA test comparing the FT, FP, FPLR and FPWLS approach against Docker Swarm in Sudoku puzzle case study.

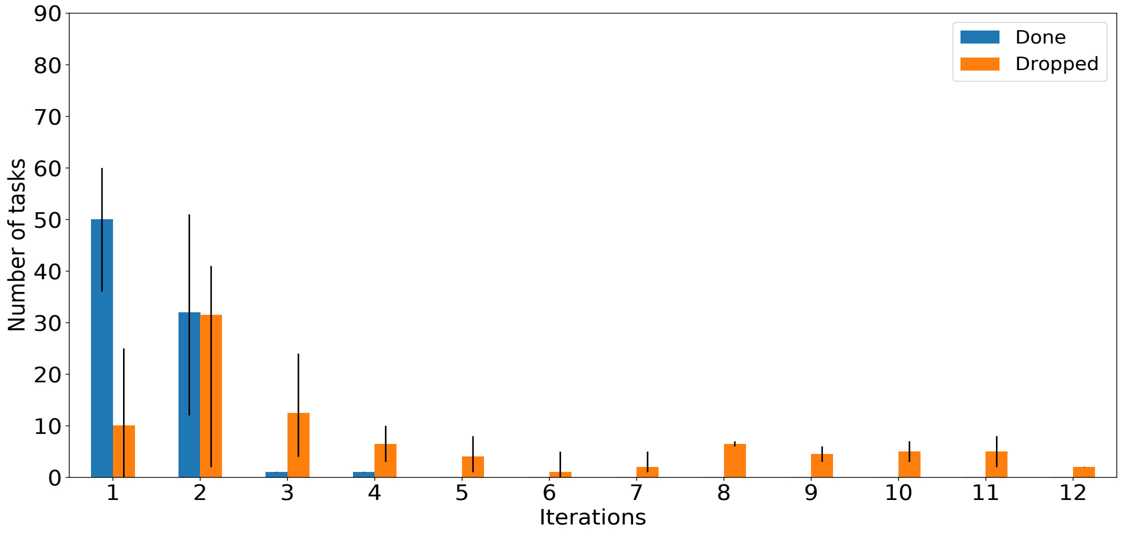
On the other hand, we used the ANOVA test for the second case study (Sudoku puzzle). We compared FT approach against Docker Swarm. As illustrated in table 5.2, the result shows that the f-ratio value is 7.81 when we compare FT approach with Docker Swarm, whereas the p-value is 0.011949 which is less than the significance level of 0.05. Therefore, at the 95% confidence level, it can be observed that there is a significant difference among the approaches. Similarly, it can be observed that the results of the other approaches are significant when comparing them against Docker Swarm.

Further iteration analysis of the approaches was carried out using both case studies: real-time multiprocessor allocation and the Sudoku puzzle. The iteration analysis is related to the number of iterations that the tasks go through to achieve the desired result. Several points can be observed in Figures 5.4, 5.5, 5.6 and 5.7. As illustrated in Figure 5.4 and 5.5, the FPLR approach could reduce the number of iterations compared to the FT, FP and FPWLS approaches. The reason for reducing the number of iterations is that some tasks were dropped earlier due to the fitness prediction condition in the FPLR approach. In addition, the number of observed data points that are considered in the prediction models (e.g. FP, FPLR and FPWLS) are small which may have some effects on the number of tasks done or dropped at each stage.

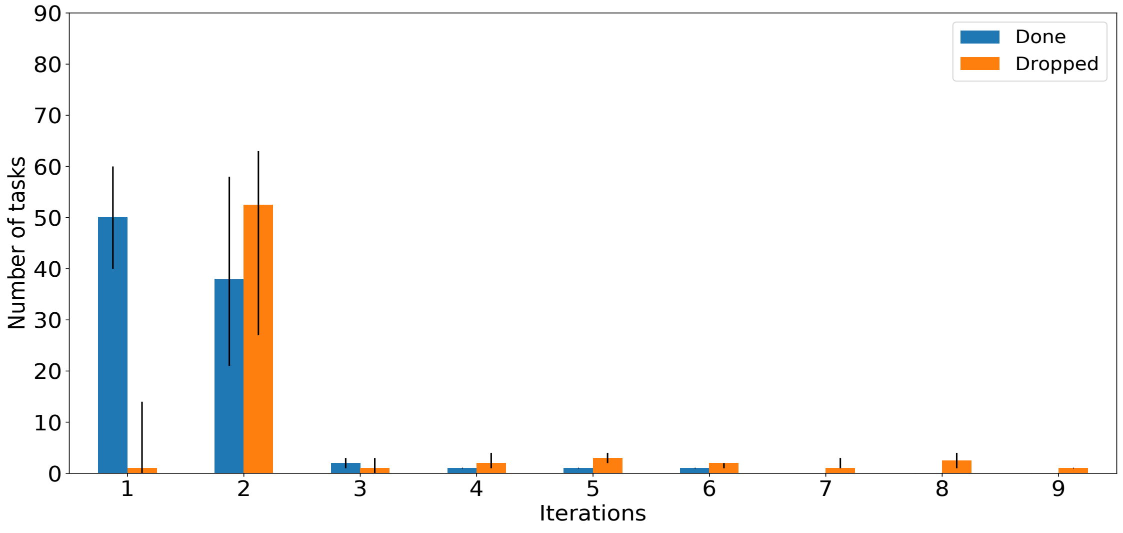
Conversely, the same analysis has been applied to a second case study, showing that the FP approach has reduced the number of iterations that the task can go through compared to the FT, FPLR and FPWLS approaches. As shown in Figure 5.6 and 5.7, most of the approaches did not reduce the number of iterations as was seen in the previous case study (Figure 5.5). This is because some tasks take longer (concerning the number of generations and iterations) to solve the Sudoku puzzle. Further, related to the puzzle's difficulty, there is a low number of difficulty levels of the puzzle, needing a larger number of generations to find a solution. Furthermore, FPLR has a smaller number of tasks beyond the eleventh iteration compared to the FT approach.

The GA applications that we are dealing with in this research are configured to achieve similar or better results. Therefore, as explained previously, some tasks can be dropped when using FP, FPLR or FPWLS, reducing the computational cost of running the task. Based on the result achieved, FPWLS has achieved 29% more successful tasks in the real-time multiprocessor allocation whereas 11% more successful tasks in the Sudoku puzzle than the FT and FP approaches while it is closer to FPLR approach with 13% more successful tasks. Additionally, the prediction approaches used in FP, FPLR and FPWLS can identify whether the task can reach the user-defined fitness based on the observed data points of the current task. Based on these approaches, the task can continue to execute or be dropped early.

Since FP, FPLR and FPWLS use predictions to forecast the fitness achieved at a given iteration, we used the root mean square error (RMSE) and mean absolute error (MAE) measurements to evaluate the performance of the prediction

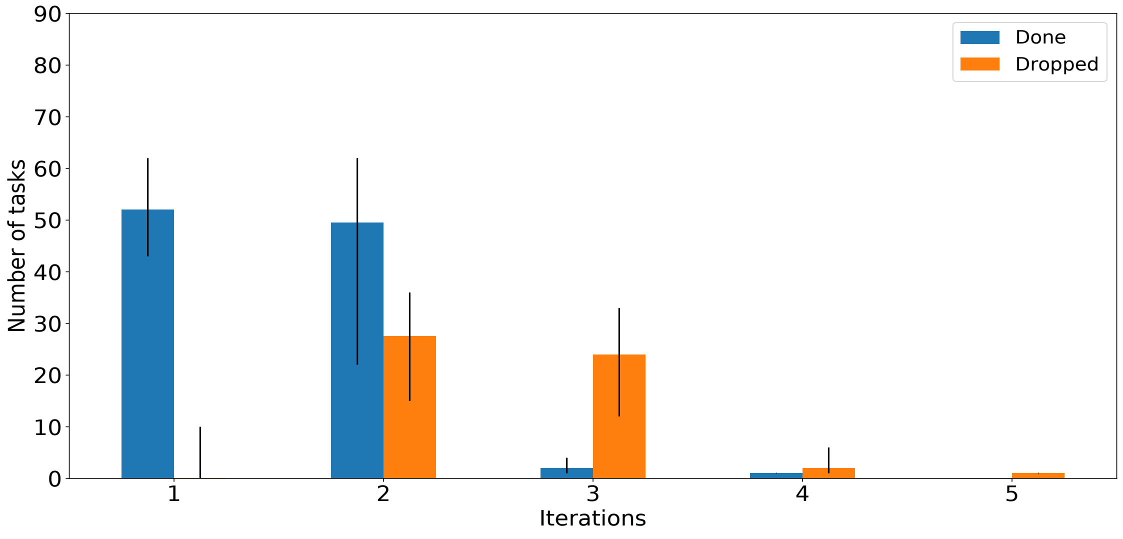


(A) FT approach.

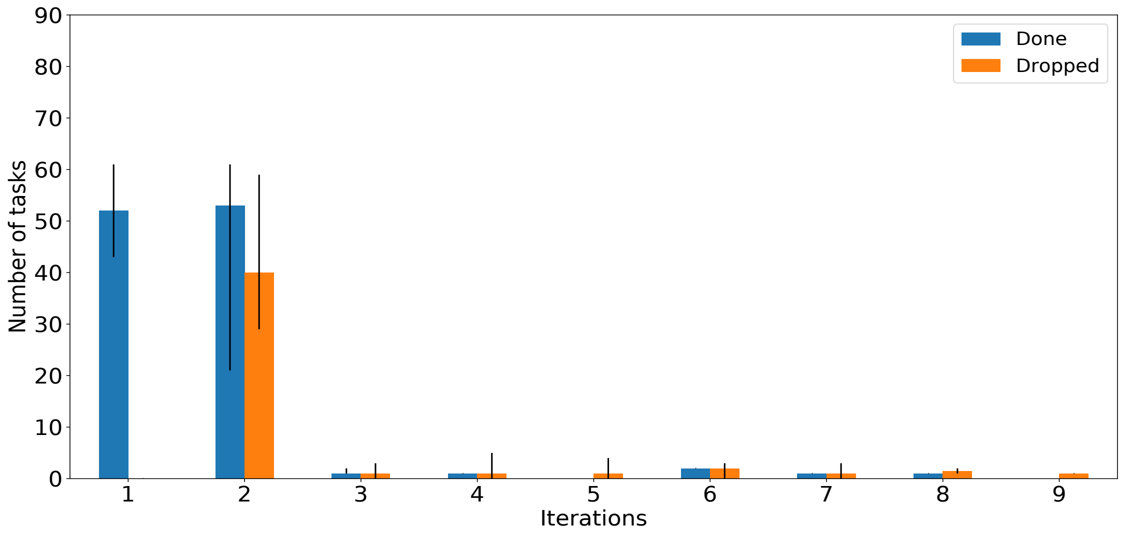


(B) FP approach.

FIGURE 5.4: The number of tasks done and dropped in each iteration of the real-time multiprocessor allocation in FT and FP approaches.

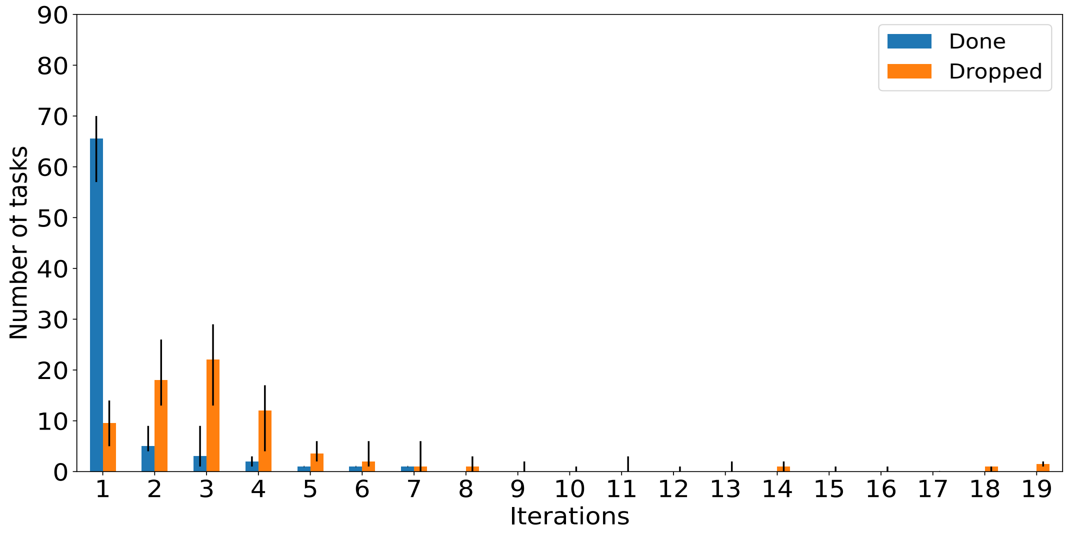


(A) FPLR approach.

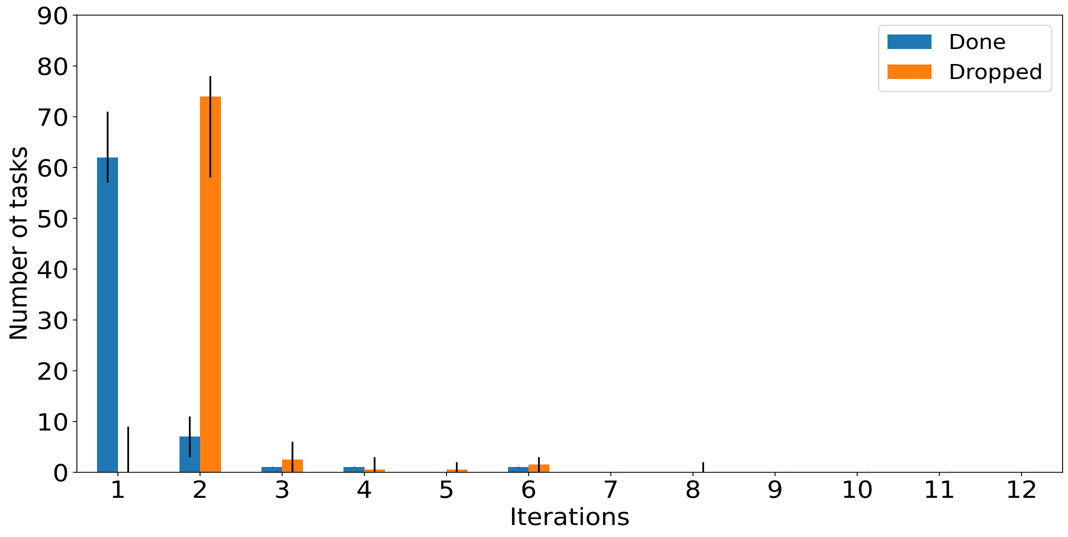


(B) FPWLS approach.

FIGURE 5.5: The number of tasks done and dropped in each iteration of the real-time multiprocessor allocation in FPLR and FPWLS.

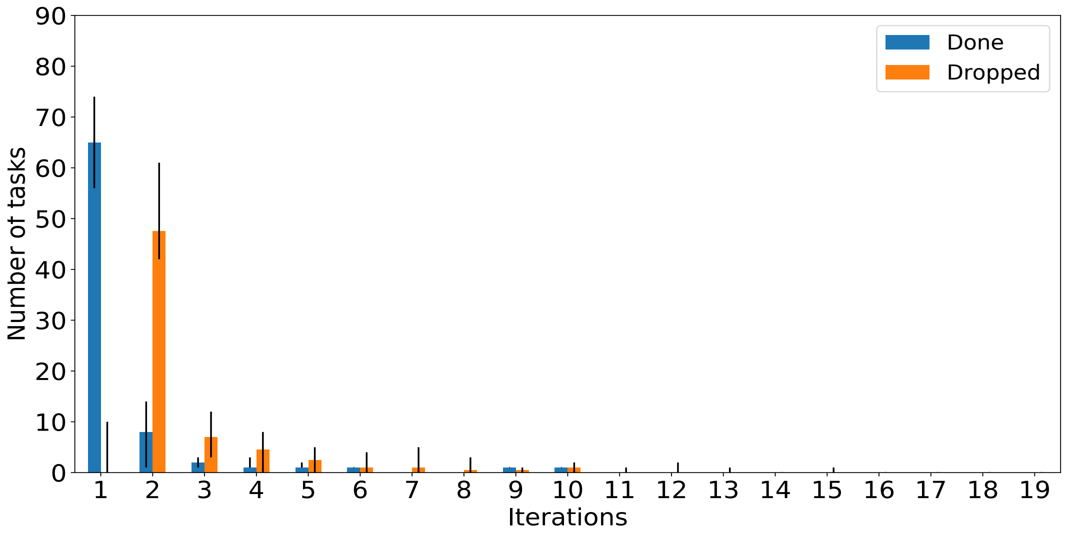


(A) FT approach.

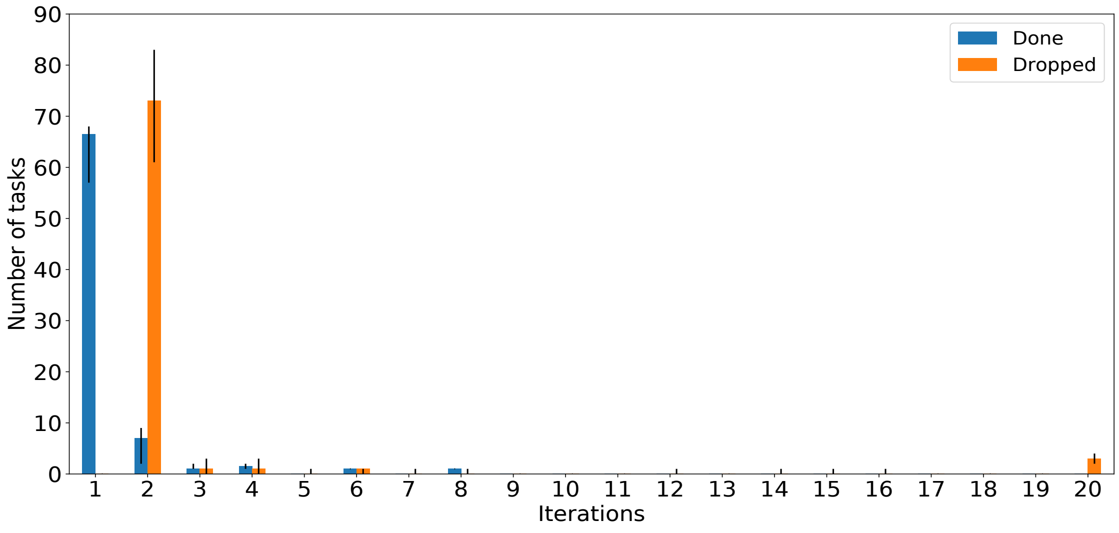


(B) FP approach.

Figure 5.6: The number of tasks done and dropped in each iteration of the Sudoku puzzle in FT and FP approaches.



(A) FPLR approach.



(B) FPWLS approach.

FIGURE 5.7: The number of tasks done and dropped in each iteration of the Sudoku puzzle.

approaches. RMSE and MAE measure the difference between the predicted value at a given iteration and the achieved value at a given iteration, as shown in Equations (5.3) and (5.4). A lower value of RMSE and MAE demonstrates a better result. In addition, they show how far the predicted values are from the actual values [99].

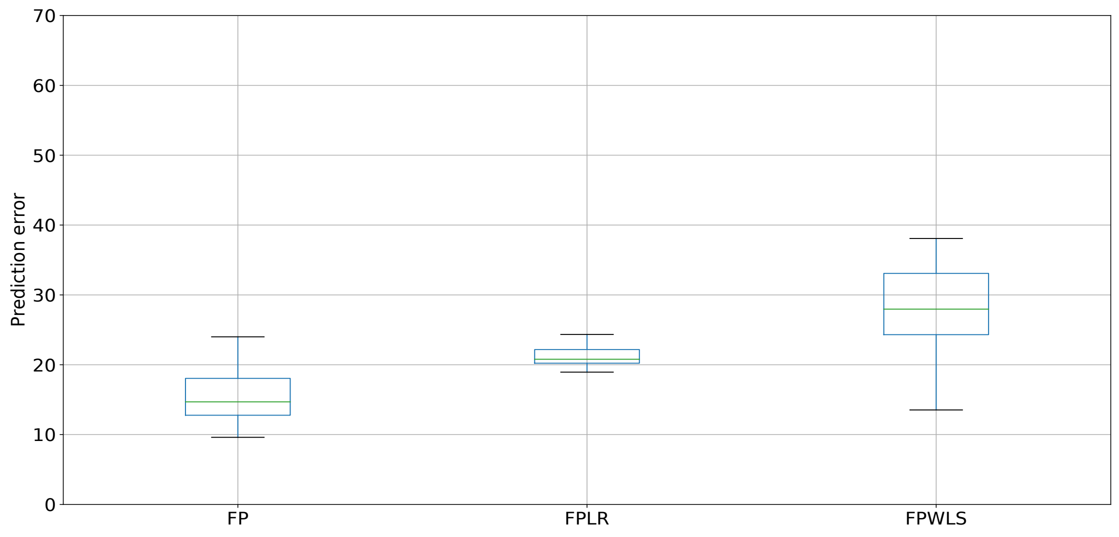
$$RMSE = \sqrt{\frac{\sum_{j=1}^{n} \left(x_j - \hat{x}_j\right)^2}{n}} \tag{5.3}$$

$$MAE = \frac{1}{n} \sum_{j=1}^{n} |x_j - \hat{x}_j| \tag{5.4}$$

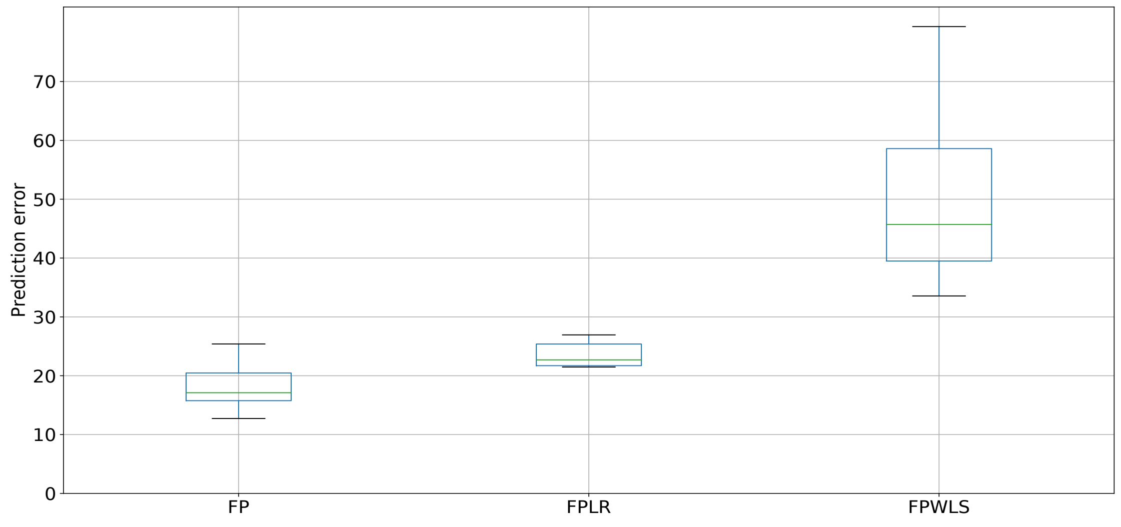
where x_j is the observed value, and $\hat{x}_j$ is the predicted value. Thus, the prediction error unit in the figure is the fitness function. As shown in Figures 5.8 and 5.9, the results of the 10 experiments of the FP, FPLR and FPWLS show that FP has a lower median than the FPLR and FPWLS approaches in both applications (real-time multiprocessor allocation and the Sudoku puzzle). Further, the results demonstrate that FP has better accuracy in both MAE and RMSE. The reason for such results is that FPLR uses a linear regression prediction, which is optimistic concerning giving an early prediction result. Since we have small data-points to be used in the FPWLS approach which is one of its drawbacks, the approach performed worse than other approaches in terms of prediction error.

Since the MAE and RMSE look into the prediction error and compare the approaches, these measurements did not show the change in the prediction error across the iterations. This can be shown in Figures 5.10 and 5.11. In both figures, the delta fitness is taking the absolute value of the difference between the fitness achieved and the fitness predicted. Further, the prediction has not been used during the first or second iteration. Therefore, the actual prediction errors start from the third iteration. Figure 5.10 shows that the FP and FPWLS approaches achieved a similar result concerning the delta fitness across the iterations, whereas the FPLR approach starts with a high delta fitness and gets lower in further iterations.

On the other hand, Figure 5.11 shows that the tasks in both approaches reach the maximum number of iterations. Further, the delta fitness starts high and slightly gets lower as the tasks go on in further iterations. Additionally, the median of delta fitness of the FPLR approach achieves better results than FP

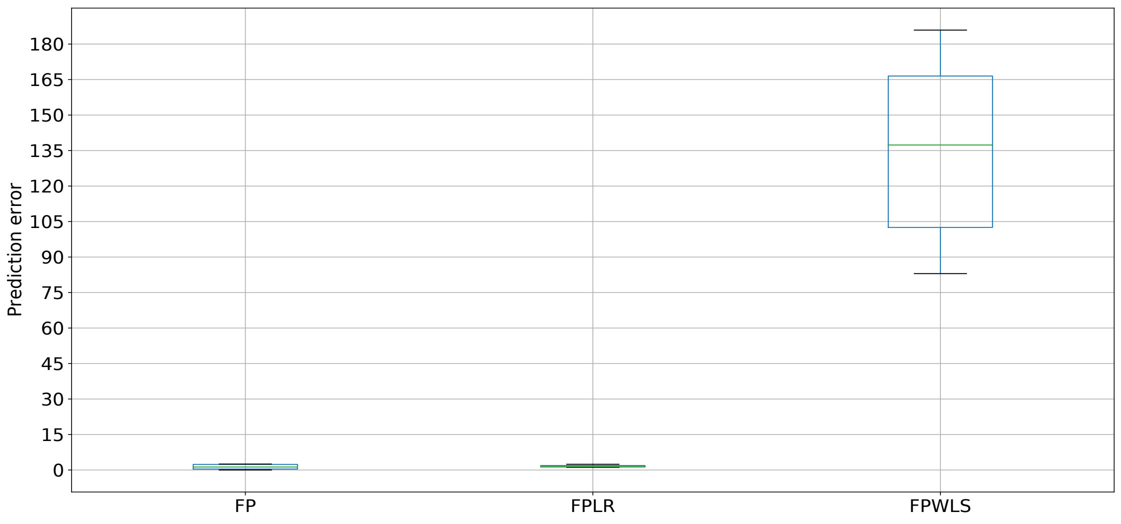


(A) MAE prediction error.

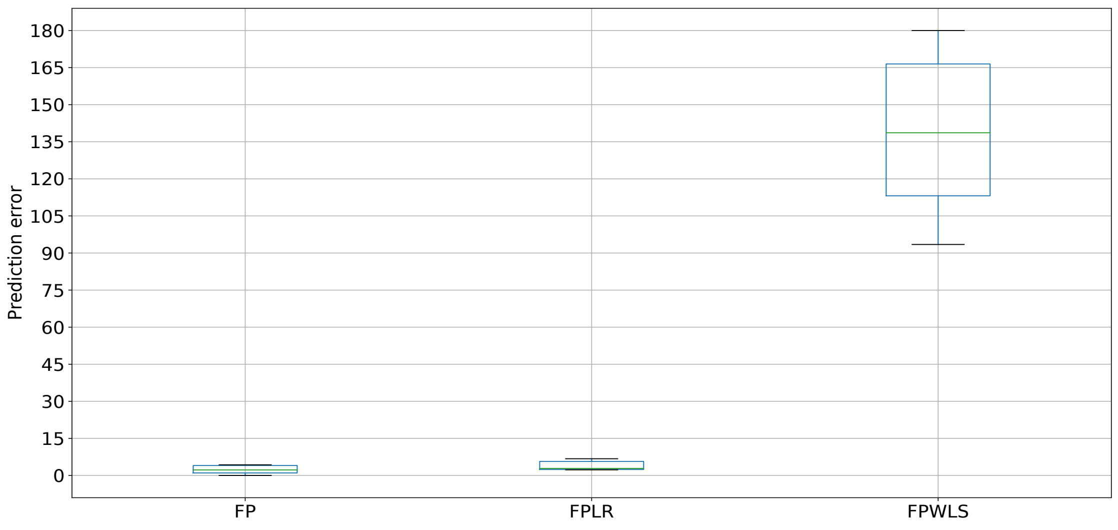


(B) RMSE prediction error.

FIGURE 5.8: Prediction errors of the real-time multiprocessor allocation.

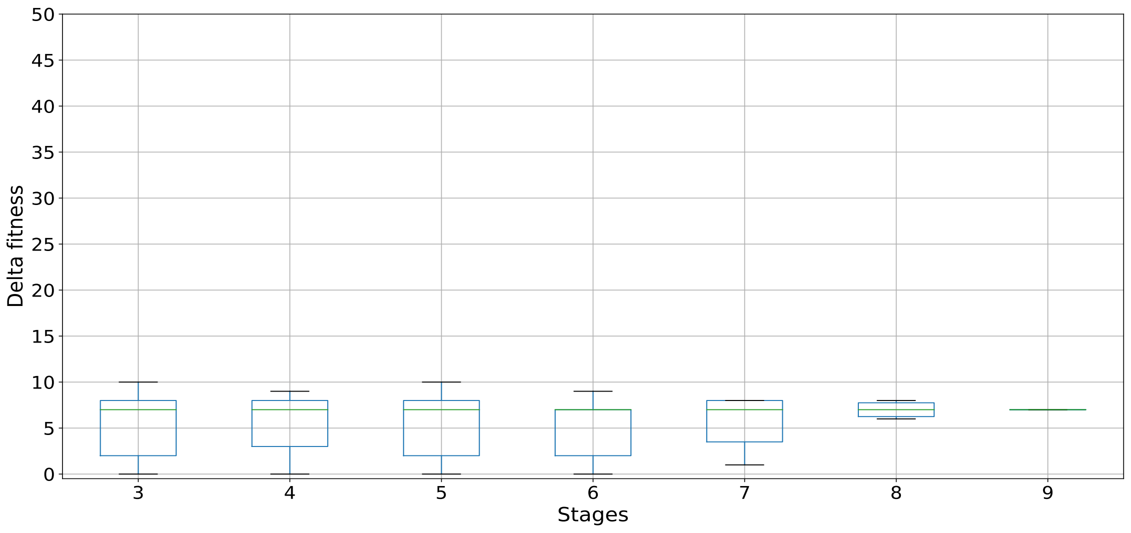


(A) MAE prediction error.

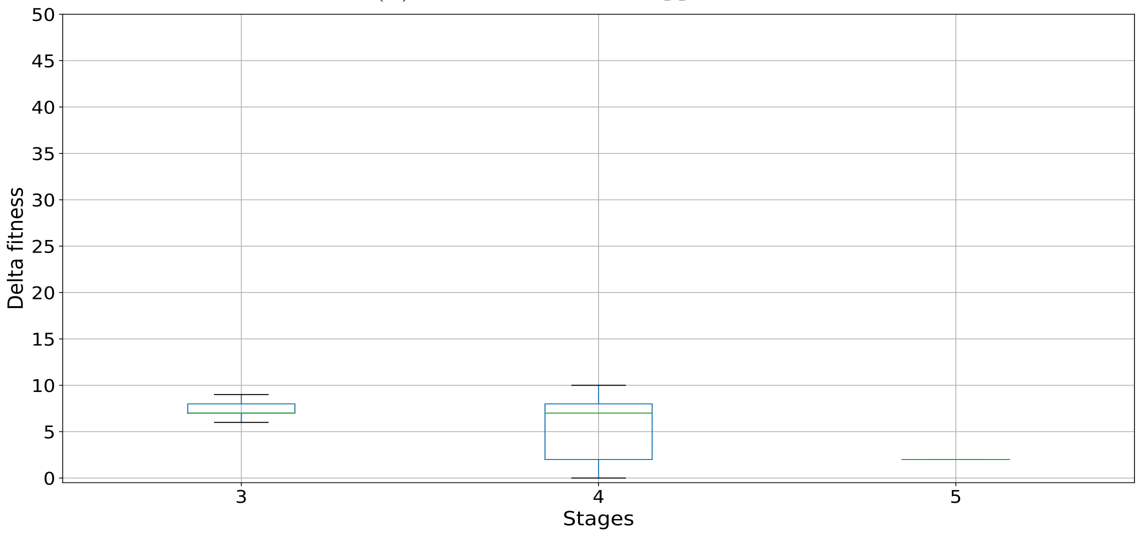


(B) RMSE prediction error.

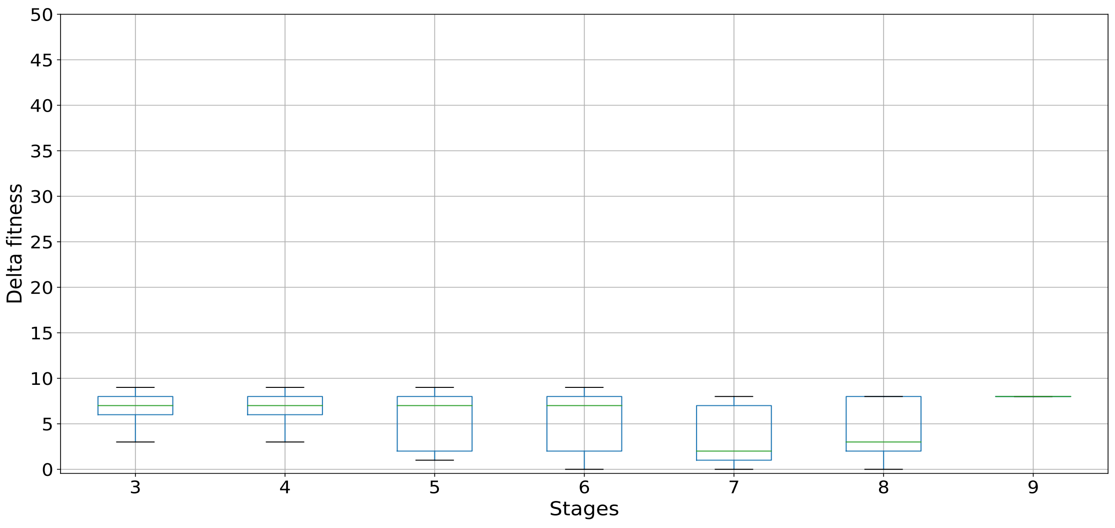
FIGURE 5.9: Prediction errors of the Sudoku puzzle.



(A) Delta fitness of FP approach.

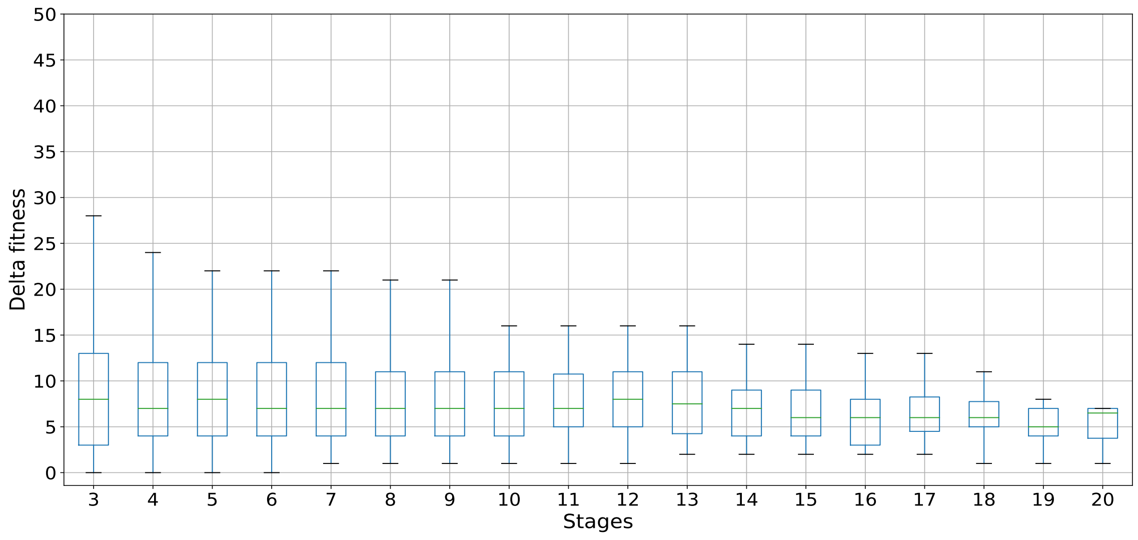


(B) Delta fitness of FPLR approach.

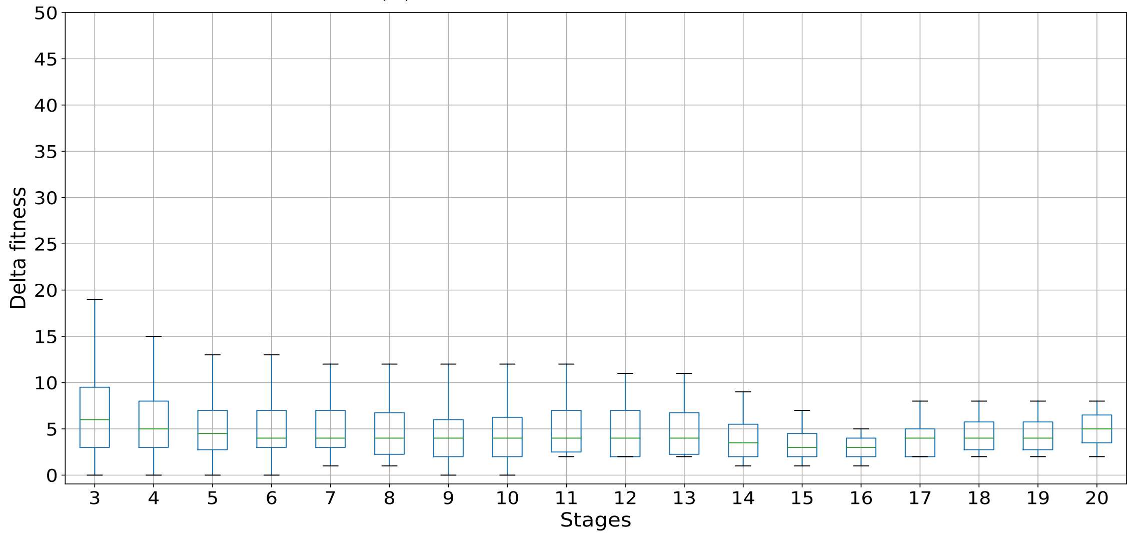


(C) Delta fitness of FPWLS approach.

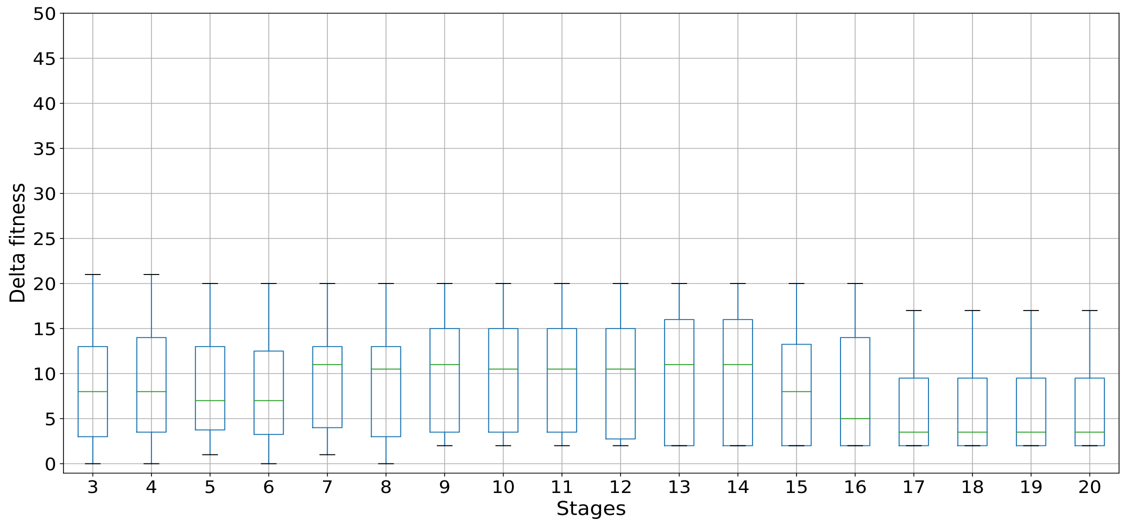
FIGURE 5.10: Delta fitness of the real-time multiprocessor allocation.



(A) Delta fitness of FP approach.



(B) Delta fitness of FPLR approach.



(C) Delta fitness of FPWLS approach.

FIGURE 5.11: Delta fitness of the Sudoku puzzle.

and FPWLS. The figures show that the FP and FPWLS approaches averages across the iterations fall above 5 for the fitness achieved, whereas the FPLR approach falls below 5 for the fitness achieved.

Designing a system that simultaneously fulfils the majority of real-time tasks on time and achieves the user-defined fitness level is challenging due to the initial assumption that the workload is unknown in advance. The goal of this research is to propose solutions that outperform the state-of-the-art container-based orchestration systems (e.g. Docker Swarm and Kubernetes). Both Docker Swarm and Kubernetes can handle the GA tasks but they only execute them without taking into consideration the user-defined fitness level and the timing constraints. Hence, we utilize these features in our work in such a way as to improve the result.

Furthermore, the idea behind these approaches is to develop a solution that is close to the baseline in order to have a fair comparison. In addition, our interest in the prediction models used is to drop the tasks that might not achieve the user-defined fitness level by the deadline even if there was no change in the fitness. By doing this, we allow the tasks that are in the waiting queue to be allocated to a node and start the execution which results in reducing the waiting time.

5.3 Summary

In general, the work in this chapter attempts to manage the GA workload in a container-based environment by considering two case studies (the real-time multiprocessors allocation and the Sudoku puzzle). To this end, we have proposed the FT, FP, FPLR and FPWLS approaches to improve the number of tasks executed on time and achieve the fitness required by the user. After running the 10 experiments and comparing FPWLS against the FT, FP and FPLR approaches, the FPWLS approach achieves better results and improves the total number of tasks executed on time while attaining the required fitness. Although prediction models usually require a suitable data sample to be used during the training of the models, the prediction model that has been used in FP, FPLR, and FPWLS is able to predict given the data points provided from executing the task.

Chapter 6

Handling node interference in managing genetic algorithm workloads

Chapter 4 discussed the orchestrator's model used in our experiment and the containerisation approach, which allows the resource manager to instantiate as many GA containers as needed. Further, the chapter explains the metrics collected to be used in the evaluation. On the other hand, Chapter 5 examined the management of container-based GA workloads over cloud infrastructure. However, one limitation that can be observed from this chapter is that resource management cannot handle node interference. Therefore, only one task can be executed per cloud node.

However, this chapter presents a weight-based node interference approach. The node interference happens when two or more tasks are executed in the same cloud node. This approach is designed to handle node interference and allow multiple tasks without negatively impacting previously allocated workload. The approach considers collecting information from the workload during the execution to achieve this and uses it to determine which node can achieve the goal of the approach.

There are two metrics that the approach used to handle node interference: slack time and fitness achieved. Thus, the proposed approach in this chapter is evaluated using a similar evaluation as the previous chapter. Experimental work is undertaken on each of the features and is compared to the baselines (Docker Swarm and Kubernetes), and the FT approach from the previous chapter.

6.1 Weight-based node interference approach

Our algorithm intends to find a feasible allocation for the GA workload to meet the user-defined QoS constraints. In other words, the response time of each task must not exceed the user-defined deadline, and the fitness achieved from executing the task must not be lower than the user-defined fitness level. In this subsection, we propose a metric called weight to represent the workload handled by a particular node at each point in time. We then use that metric to guide our allocation process so that new workloads are placed on the nodes more likely to handle them without negatively impacting previously allocated workload.

In our work, node interference happens when there is more than one task executed in a cloud node. As these tasks share the same resources, each task executed in the same node can be affected owing to the sharing of resources. Therefore, the weight of each node considers the task's fitness achieved and slack time, which is the time difference between the deadline and response time. As the deadline and the achieved fitness of the task are important in our work, the task goes through several iterations. Each iteration has a fixed number of generations used to track the time taken to execute the task (response time) and fitness achieved.

Concerning allocating an incoming task to a node, the resource allocation will always allocate the incoming tasks to an idle node if an idle node exists. In case all the nodes are busy executing other tasks, we use the weight of each node. We calculate the normalised slack time of the task (NST_{jk}) to obtain the weight of the nodes, as illustrated in equation (6.1). Simultaneously, we find the normalised difference in the fitness of the task (NDF_j), as shown in equation (6.2). We used $maxFA_{jk}$ and FA_{jk} to find the NDF to avoid dividing it by zero value as FR_j can be zero. Furthermore, as shown in table 6.1, three nodes are an example, and each node has some tasks under the process of execution. The table represents the information of currently executed tasks at some point in time with NST and NDF values.

$$NST_{jk} = \frac{D_j - RT_{jk}}{D_j} \tag{6.1}$$

$$NDF_j = \left(\frac{|FA_{jk} - FR_j|}{maxFA_{jk}} \right) \tag{6.2}$$

Node	**Tasks within the node**	D_j	RT_{jk}	NST_{jk}	FR_j	FA_{jk}	$maxFA_{jk}$	NDF_j
node-1	task-1	88	30	0.659	0	6	27	0.22
	task-2	125	3	0.976	9	14	14	0.357
node-2	task-3	130	10	0.923	18	35	35	0.485
node-3	task-4	198	94	0.525	0	8	26	0.307
	task-5	191	8	0.958	9	42	42	0.785

TABLE 6.1: Example of data collection in each of the node while executing tasks.

In the equation D_j is the deadline of the task, and RT_{jk} is the response time at the k iteration. As we are dealing with tasks' response time less than the deadline, we do not need to use the absolute value of D_j and RT_{jk}. Further, $max\ FA_{jk}$ is the maximum fitness achieved, as the fitness achieved is updated after each iteration whereas FA_{jk} is the fitness achieved at iteration k. Also, FR_j is the fitness required. The intuition behind NST_{jk} and NDF_j is finding the node of a task with a larger slack time and less fitness to be achieved, where the effect of the new task will be less on the existing task.

As soon as the previous values are collected from each task j in node i, equations (6.1) and (6.2) will start the normalisation process. As such, we obtain the final result of both NST_j and NDF_j. Subsequently, we apply equation (6.3) to determine the weight of each node in the cluster (W_i).

$$W_i = \sum_{j=1}^{n} NST_{jk} + NDF_j \tag{6.3}$$

Once we have the weights based on equation (6.3), we allocate the incoming task to the node with the maximum weight value. This process ensures that the incoming task will have a negative impact on the existing task when the resources are shared. For example, we can apply and calculate the weight of each node (table 6.1). Then, *node-1* will have a total weight of 2.209, *node-2* will have a total weight of 1.408, and *node-3* will have a total weight of 2.575. Based on these weight values of the nodes, the incoming task will be allocated to *node-3* as the node will have the maximum weight among the other nodes.

When executing two or more tasks in the same cloud node, the node might reach a saturation point and at that point, some of the tasks might get more resources than the other tasks closer to their deadline. Thus, We use a fixed threshold of CPU utilisation to avoid the node from reaching a CPU saturation point. Once a node reaches the threshold, we limit the tasks within the node based on the tasks' deadlines. In other words, the task that is closer to its deadline will be allocated more CPU utilisation. We calculate the percentage of the task based on its response time and how long the node executes the task to limit the CPU utilisation of the tasks, as illustrated in equation (6.4).

$$PT_j = \frac{RT_{jk}}{D_j} \tag{6.4}$$

We use algorithm (1) to illustrate the process of limiting the tasks within a node. From line (1 - 7), Docker Manager, which is part of the orchestrator, collects the CPU utilisation of each node in the cluster for every fixed interval by using a Linux command ”$head - n1/proc/stat$” for calculating the CPU utilisation. The information is then passed to the Node Observer to update every node and its CPU utilization. At line 4, the Node Observer checks the fixed threshold against every node in the cluster to see whether the node exceeded the threshold which then flagged every node in the cluster with true if a node exceeded and false if the node did not reach the threshold. Subsequently, from line (8 - 13), the Resource Allocation checks every node with a true flag to limit every task within that node based on equation (6.4). Assuming node-3 reaches the threshold for the CPU utilisation, the node will be flagged true, and the Docker Manager will use one of the Docker options, which is ("$--cpu =< value >$") to specify the CPU available for the task. This option allows the resource manager to limit the CPU usage of a task. Taking an AWS EC2 instance with 1 CPU as an example, the value range from 0.0 to 1 which means that $--cpu = 0.6$ reflects that the task will guarantee maximum of 60% of the CPU utilization.

Considering table 6.1, task-4 will guarantee maximum of ≈ 0.47 of the CPU, while task-5 will guarantee a maximum of ≈ 0.05 of the CPU. In this case, the task closer to reaching the deadline will be allocated more CPU to provide a better chance to achieve the fitness required before the deadline. On the other hand, once the CPU utilisation of the node is back to normal (less than the threshold), all the tasks within that node will return to their normal state with full access to the CPU.

For more clarification, the approach discussed in this section has several features involved to assist resource management, such as NST_j, NDF_j, W_i and W_i with monitoring node utilisation. Each of these features can be used individually to analyse their result. Since we are dealing with timing constraints, we use node utilization to distribute the CPU among the existing tasks in the node based on the percentage of the task.

Algorithm 1: CPU utilization control.

```
Input: CPU utilisation of $node_i$.
Result: Limit the CPU utilisation of each task.
1  nodeFlag = $< node_i, false >$;
2  for $node_i$ in Nodes do
3      collect CPU utilisation for $node_i$;
4      if CPU utilisation for $node_i > 80\%$ then
5          nodeFlag = $< node_i, true >$;
6      end
7  end
8  if nodeFlag == $< node_i, true >$ then
9      for $task_j$ in $node_i$ do
10         Compute $PT_j$ from equation (6.4) for each task;
11         limits the CPU usage for $task_j$ based on $PT_j$ value.
12     end
13 end
```

6.2 Admission control for firm real-time task

Task allocation and scheduling algorithms that use admission control have been shown to be useful in the case of dynamic workloads in container-based systems because they can boost platform utilization and meet timing requirements. On real-time systems, three real-time types can be distinguished based on the effects of missing a deadline: hard, where any deadline violation may endanger the correct behaviour of the entire system; firm, where it does not harm the correct system behaviour but the late completion of the task is worthless; and soft, where a late task still has consequential damage for the system [88].

In this section, we propose an admission control based on slack time to improve the number of successful tasks that achieved the user-defined fitness level by the deadline. In this section, we are assuming that we have a firm deadline. In case there is an idle node new tasks will be allocated to it. In case all the nodes have currently executing tasks, this technique uses the information from each task and then allocates the new task to a node with a minimum slack time.

The main goal of choosing the minimum slack time is to allow other nodes with higher slack time to finish executing the tasks and obtain a positive value by achieving the required result. From the previous equation (equation 6.1) we can formulate a new equation to calculate the normalized slack time of every task in a node as shown in equation 6.5.

$$NSTAD_i = \sum_{j=1}^{n} \frac{D_j - RT_{jk}}{D_j} \tag{6.5}$$

As shown in the normalized slack time ($NSTAD_i$), the equation D_j is the deadline of the task, and RT_{jk} is the response time of task j at the k iteration. After allocating the task to an idle node if exists otherwise to a node with a maximum $NSTAD_i$, the task will start the execution. After each execution, admission control will check the response time of the task with its user-defined deadline. If the tasks still have time remaining and the user-defined fitness level still did not achieve the task will be admitted again for further execution.

6.3 Evaluation

This section will considers several techniques and baselines to observe how effectively they improve task execution. An experimental setup will be discussed first, followed by a discussion of the experimental result. There are numerous evaluations of the experimental result in the report. An example is comparing different ways based on the number of successful tasks in each methodology. There are features involved in calculating the weight. Therefore, each of the features will be analysed separately. Finally, the combined approach is also analysed. Then, an iteration analysis was undertaken to check the behaviour of the tasks when they go into an iteration process.

6.3.1 Experimental setup

In this experimental work, we follow a similar setup as section 5.3.1. As we consider node interference in this chapter, we assume that the cloud node can execute multiple tasks at the same time per cloud node. The experimental work aims to evaluate different approaches, such as *NST, NDF* and *W*, and compare them against Docker Swarm and Kubernetes as baselines and as an FT approach discussed in the previous chapter. Additionally, the second experimental hypothesis (related to node interference) will be investigated in this section of the evaluation. Moreover, the number of successful tasks, which will be used in the evaluation, can be defined as the task whos response time is less than or equal to its deadline and its fitness achieved is greater than or equal to its fitness required.

6.3.2 Experimental results

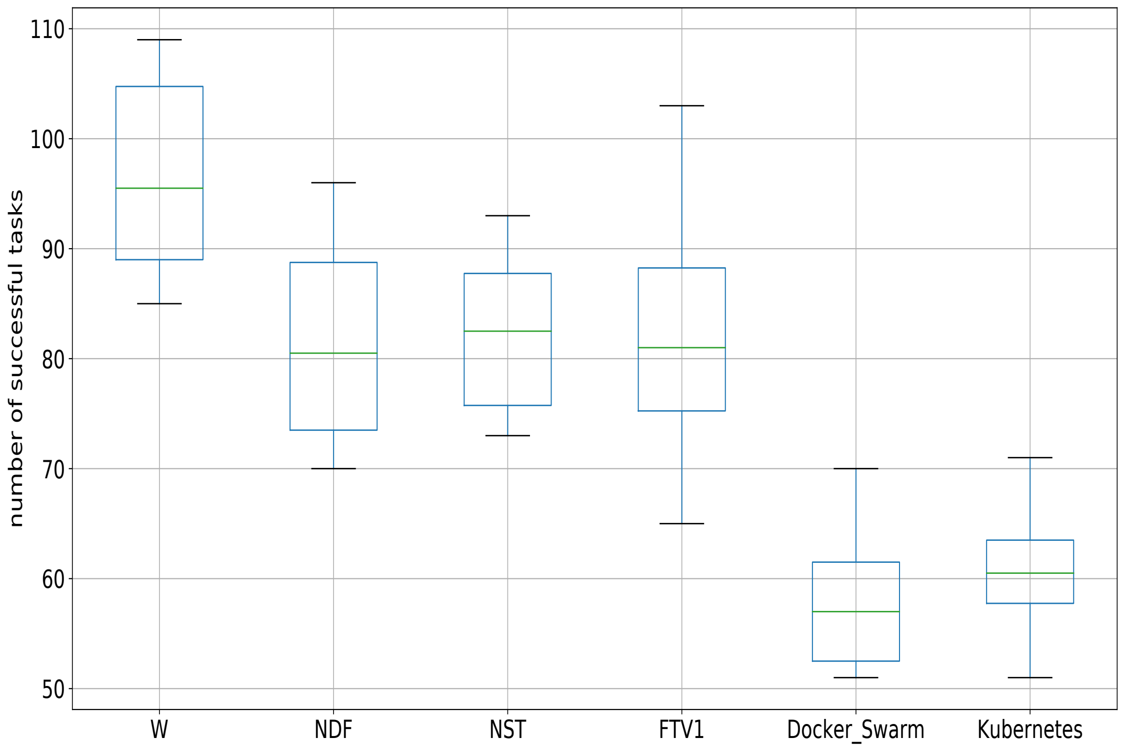


FIGURE 6.1: Result of 10 experiments of different approaches and different GA workloads (real-time multiprocessor allocation) in terms of the number of successful tasks.

As we considered two case studies (real-time multiprocessors allocation and Sudoku puzzle) in our work, we ran 10 experiments on each of the case studies. Thus, in each experiment, we executed 150 tasks on the *W*, *NDF* and *NST* approaches. Then, we compared them against the FT approach from the previous chapter as well as Docker Swarm and Kubernetes as they considered the baselines. The reason for choosing the FT approach and not the others is because FP and FPLR use prediction features. Also, both *W* and FT are rather similar in executing the tasks, except that the cloud node in the previous chapter executes only one task at a time per cloud node, whereas *W* is able to execute multiple tasks at a time per cloud node. We used the same metrics explained in chapter 4 which are response time and fitness achieved. The results show that the approach we implemented (*W*) performed better than the other approaches, including Docker Swarm and Kubernetes in both case studies. The

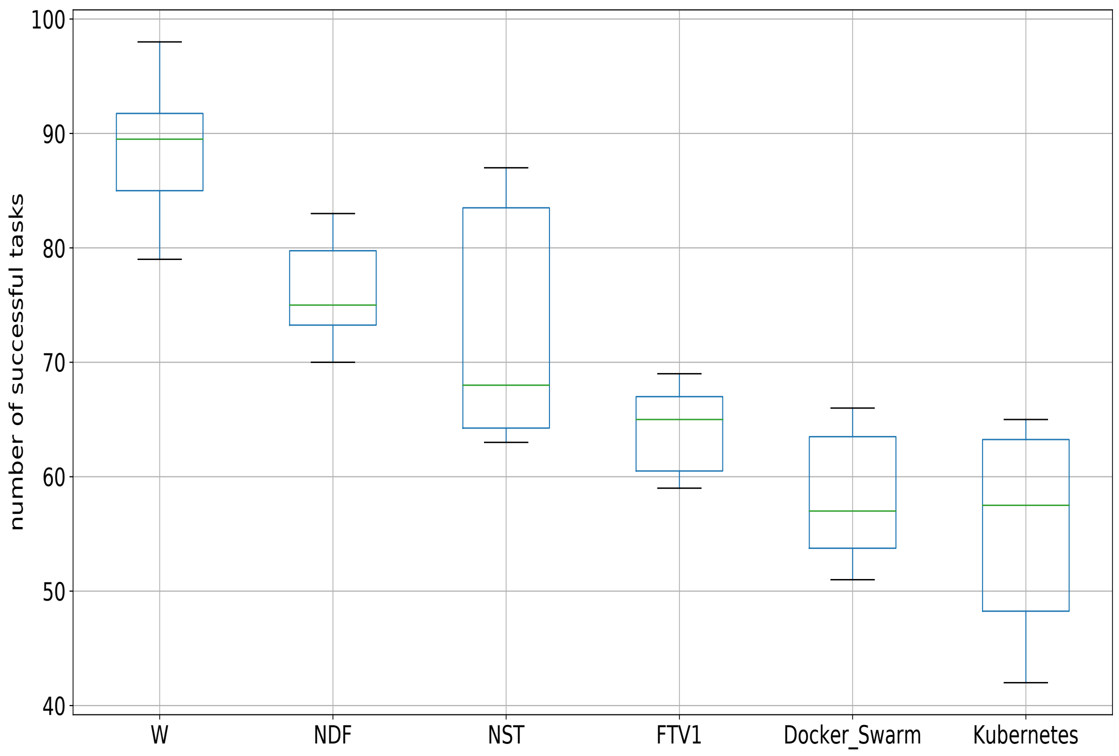


FIGURE 6.2: Result of 10 experiments of different approaches and different GA workloads (Sudoku puzzle) in terms of the number of successful tasks.

median and maximum observed results of *W* in both the case studies are higher than the median and maximum observed results in other approaches, as shown in Figure 6.1 and 6.2.

As shown in Figure 6.1, the W approach has a higher percentage at 26% more successful tasks when compared to Docker Swarm and Kubernetes. Also, NDF achieved 15% more successful tasks, whereas 16% more successful tasks were achieved for NST and FTV1. On the other hand, using Figure 6.2, *W* also achieved a higher percentage of successful tasks at 20% when compared to other approaches. Additionally, NDF achieved 12% more successful tasks, whereas NST achieved 10% more successful tasks, and FTV1 achieved 4% more successful tasks.

Approach	f-ratio
W vs. Docker Swarm	129.54
W vs. Kubernetes	99.87
W vs. FTV1	10.56

TABLE 6.2: One-way ANOVA test comparing the W approach against Docker Swarm, Kubernetes, and FTV1 in real-time multiprocessor allocation case study.

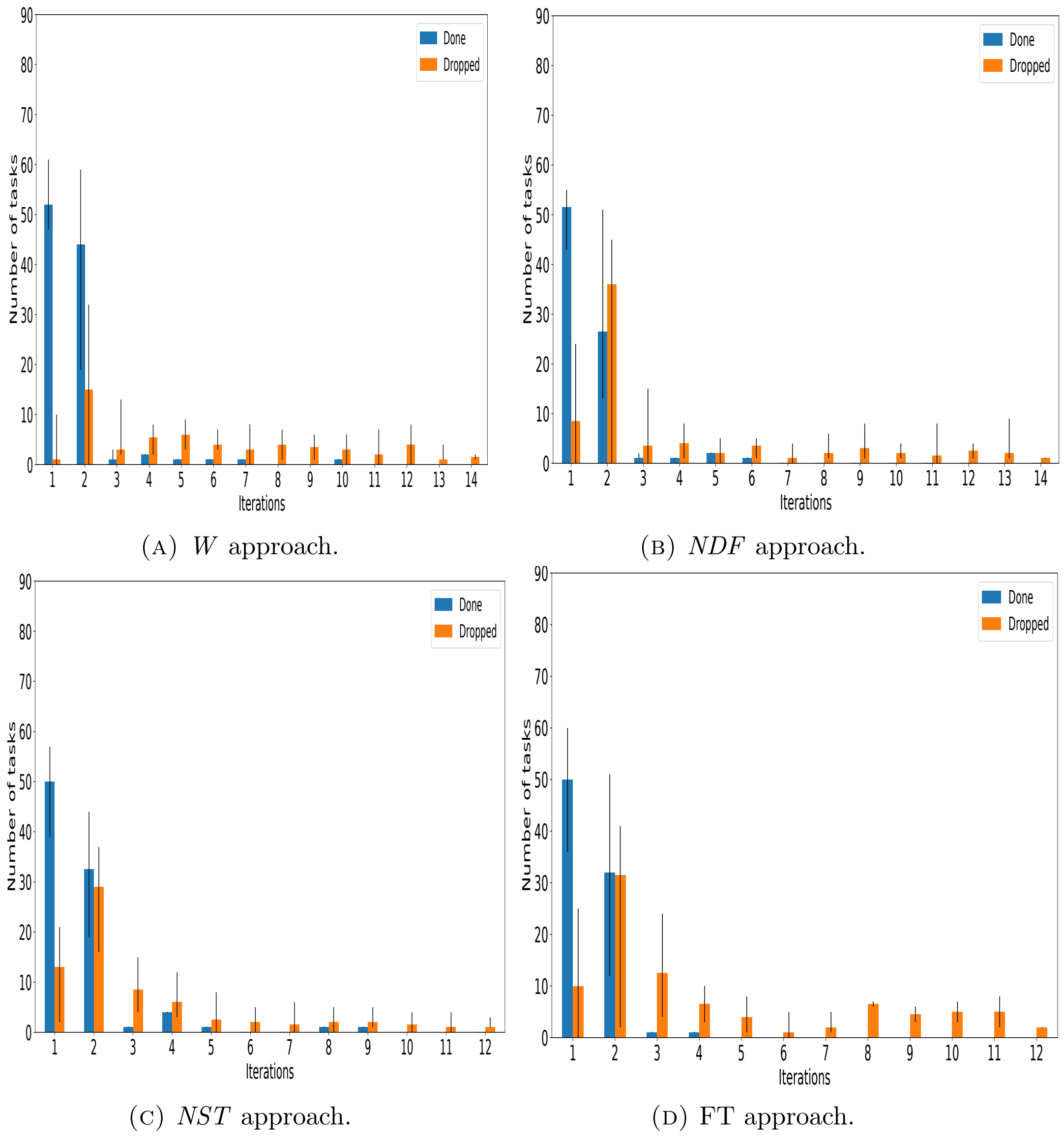
We compared the results using one-way analysis of variance (ANOVA) tests to statistically demonstrate the difference between the results obtained from the experimental approaches. for the same reason as stated in the previous chapter, Docker Swarm and Kubernetes achieve similar results which allows us to choose one of them and compare our result against it. The f-ratio is the variation between the approaches' mean values based on the f-ratio. The p-value can be obtained and compared to the significance level (0.01, 0.05, or 0.10).

As illustrated in table 6.2, the result shows that the F-ratio is 129.54 when comparing W against Docker Swarm, whereas the F-ratio is 99.97 when comparing W against Kubernetes. In both cases, the p-value (p-value < 0.00001) is less than the significance level of 0.05. Therefore, at the 95% confidence level, it can be observed that there is a significant difference among the three approaches. Furthermore, we compared W with FTV1, and the result shows that the F-ratio is 10.56, and the p-value (0.004437) is less than the significance level. Thus, at the 95% confidence level, it can be observed that there is a significant difference between the approaches.

Approach	f-ratio
W vs. Docker Swarm	114.13
W vs. Kubernetes	84.94
W vs. FTV1	89.35

TABLE 6.3: One-way ANOVA test comparing the W approach against Docker Swarm, Kubernetes, and FTV1 in Sudoku puzzle case study.

On the other hand, we used the ANOVA test for the second case study (Sudoku puzzle). We compared W, Docker Swarm, Kubernetes and FTV1. As illustrates in table 6.3, the result shows that the f-ratio value is 114.13 when we compare W with Docker Swarm, whereas the f-ratio is 84.94 when we compare W with Kubernetes. Thus, in both cases, the p-value (p-value < 0.00001) is less than the significance level of 0.05. Therefore, at the 95% confidence level, it can be observed that there is a significant difference among the approaches. Additionally, we compared W and FTV1. The analysis result shows that the f-ratio is 89.35 and the p-value (0.00001) is less than the significance level. Thus, at the 95% confidence level, it can be observed that there is a significant difference between the approaches.



(A) *W* approach.

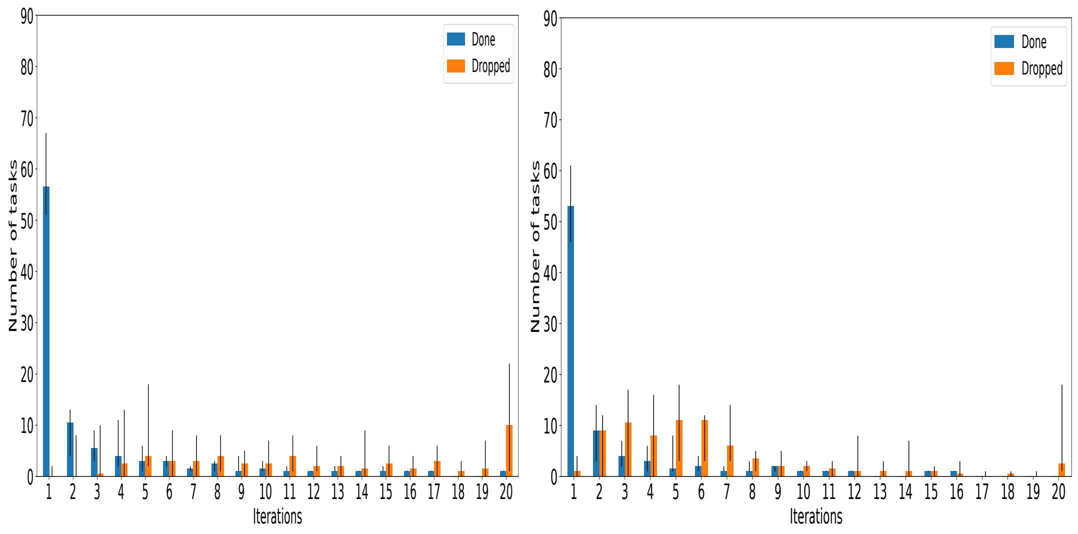
(B) *NDF* approach.

(C) *NST* approach.

(D) FT approach.

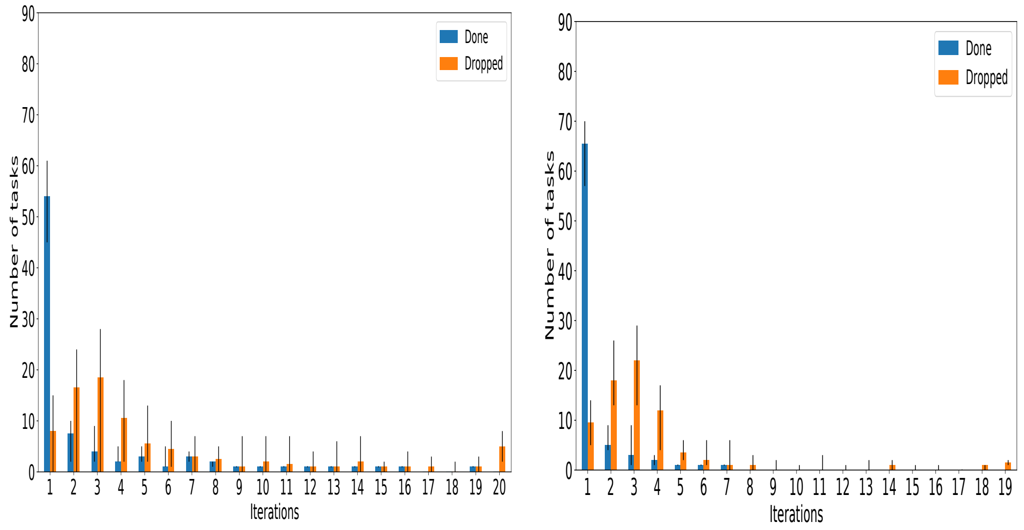
FIGURE 6.3: Number of tasks done and dropped in each iteration of the real-time multiprocessor allocation.

Next, we undertook an iteration analysis of the approaches of *W*, *NDF*, *NST*



(A) *W* approach.

(B) *NDF* approach.



(C) *NST* approach.

(D) FT approach.

Figure 6.4: Number of tasks done and dropped in each iteration of Sudoku puzzle.

and FT. Further, we applied this analysis to both case studies: real-time multiprocessor allocation and the Sudoku puzzle. The iteration analysis is related to the number of iterations that the tasks go through to achieve the desired result. Several points can be observed in Figures 6.3 and 6.4. As illustrated in Figure 6.3, the *NST* approach achieves a similar result as the FT approach. While *W* and *NDF* approaches exceed the number of iterations when compared to the FT approach.

Figure 6.3(A) shows there are more cases of tasks that achieved the required fitness on time (done tasks) when compared to (B), (C) and (D). Meanwhile, there are better cases of tasks (done tasks) in (B) and (C) compared to (D). On the other hand, although the number of iterations is the same in (C) and (D), the number of dropped tasks is slightly different, as can be observed from the 4^{th} iteration until the 9^{th} iteration. Considering the slack time, (C) has improved the number of tasks done which allows tasks to have more time and a better change of achieving the fitness required. Furthermore, as there were no tasks placed in the queue, several tasks took advantage of the extra time and spent it on the tasks to be executed. Taking the *W* approach from the 3^{th} until the 10^{th} iteration as an example, the number of tasks done are shown when compared to the FT approach. Therefore, at least 15 tasks took advantage of the node interference approach and the tasks were either done or dropped before.

We also conducted the same analysis and applied it to the second case study (Sudoku puzzle). Figure 6.4 shows that as we allow multiple tasks to be executed simultaneously, more tasks tend to go through the iterations trying to achieve the required fitness on time, as illustrated in subfigures (A, B and C) and compared with the previous approach (FT). Further, figures (B and C) tend to drop more tasks at early iterations when compared to (A). Thus, we can observe that the *W* approach achieved more cases of tasks done across the iterations when compared to other approaches. Taking the number of iterations from the 6^{th} until the 18^{th} iteration as an example, because the tasks do not have to be placed in a queue in *W*, more tasks spend that time executing and some of them achieved the user requirement when compared to the FT approach.

As the second part of of the experiments use CPU control, we conducted more analysis comparing different threshold when limiting the CPU as illustrated in figures 6.5 and 6.6. Figure 6.5 shows that *WCPU* with 70% threshold achieved a better result when compared to 80% and 90% thresholds. Comparing the same result that we observed previously, *WCPU70%* achieved more successful tasks with 17% when compared to *WCPU80%* with 15% and *WCPU90%* with 9%. Other comparisons can be observed when we compare these results against *W* with CPU control as it achieves 26% more successful tasks when compared to Docker Swarm. Thus, based on the result, some tasks are affected by limiting the CPU during the execution. Therefore, they did not achieve the fitness level by the deadline.

On the other hand, we conducted the same analysis on the second case study as illustrated in figure 6.6. *WCPU90%* and *WCPU70%* achieve similar results when compared against the *W*. They both achieved 20% more success when compared to Docker Swarm. This means that CPU control with 90% and 70%

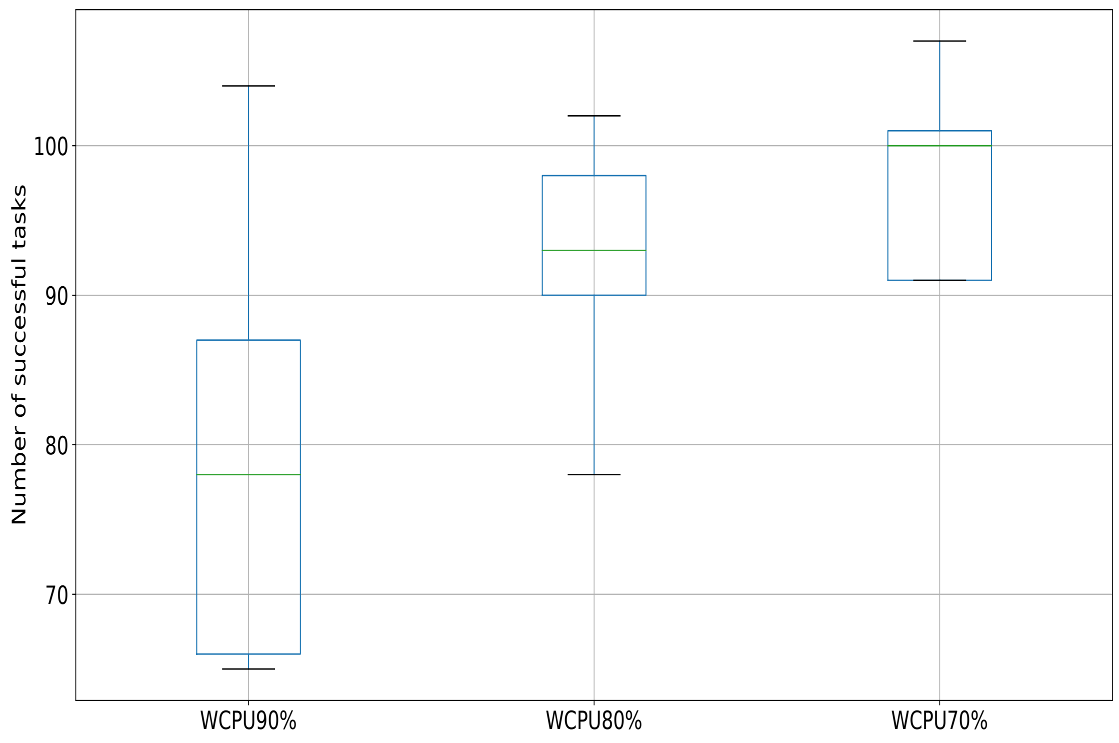


FIGURE 6.5: Result of 10 experiments of different approaches and different GA workloads (real-time multiprocessor allocation) in terms the CPU control in the W approach.

has a slight effect on the execution of the task, whereas *WCPU80%* is affected by the CPU control which achieves 15% more successful task.

One of the limitations of the previous approach (FT) is that every task has to wait in the queue in case all the nodes are busy executing other tasks. However, the approach that we implement can handle node interference. Therefore, resource allocation will allocate the incoming tasks based on the new approaches if there is no idle node. From this point, further analysis was undertaken related to the waiting time of tasks for both case studies, as illustrated in Figures 6.7 and 6.8. In these figures, different colours mean different experiments. It can be observed from both figures that tasks spent more than 5 seconds waiting to be executed, as shown in both figures in subfigure (B). On the other hand, the W approach in Figure 6.7 shows that none of the tasks waited for a node to be executed. Meanwhile, in Figures 6.8, the approach significantly reduced the waiting time.

Additionally, for the previous analysis, we compared *W* and *WCPU80%* as an example concerning the CPU utilisation as we control the CPU utilisation of the tasks as the node exceeded the threshold. We applied this analysis to both of the case studies, as illustrated in Figures 6.9, 6.10, 6.11 and 6.12. Each figure

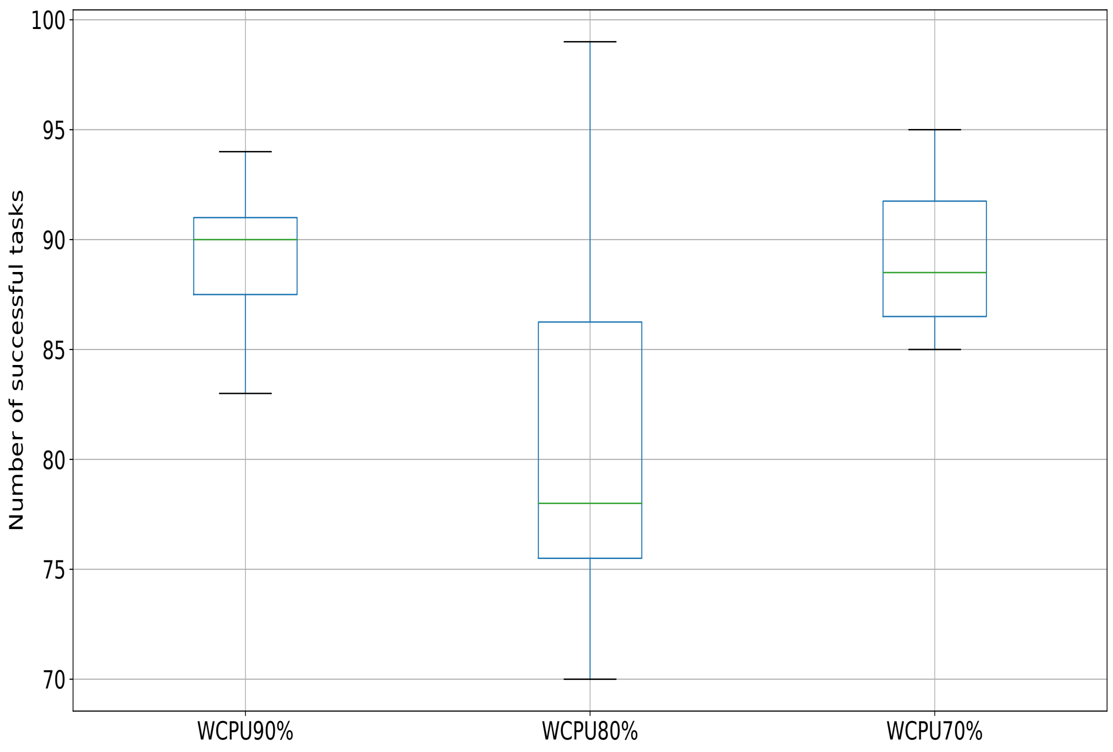
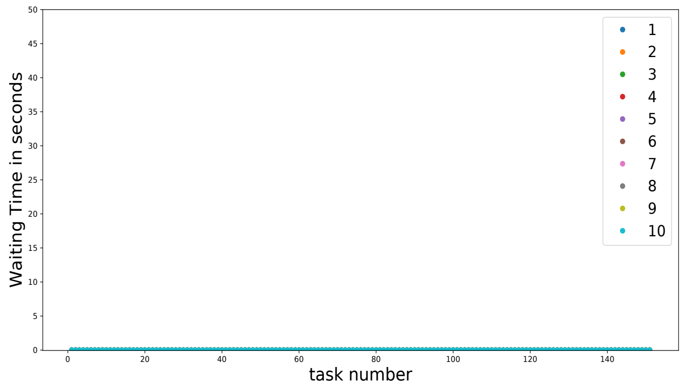
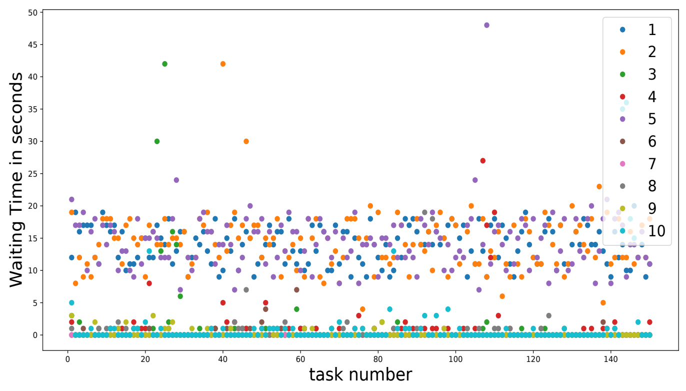


FIGURE 6.6: Result of 10 experiments of different approaches and different GA workloads (Sudoku puzzle) in terms the CPU control in the W approach.

legend represents a cloud node, so each subfigure shows two nodes only to see the difference before and after applying the CPU utilisation control. General, it can be observed from Figure 6.9 that most of the nodes tend to exceed the threshold (80%) of CPU utilisation. On the other hand, Figure 6.10 illustrates that controlling CPU utilisation reduces CPU utilisation in some situations. The reason for this reduction is that when we limit the CPU utilisation, we only apply it after the first iteration of the tasks to collect the information for equation (6.4).



(A) *W* approach.



(B) FT approach.

FIGURE 6.7: Waiting time of tasks in real-time multiprocessor allocation.

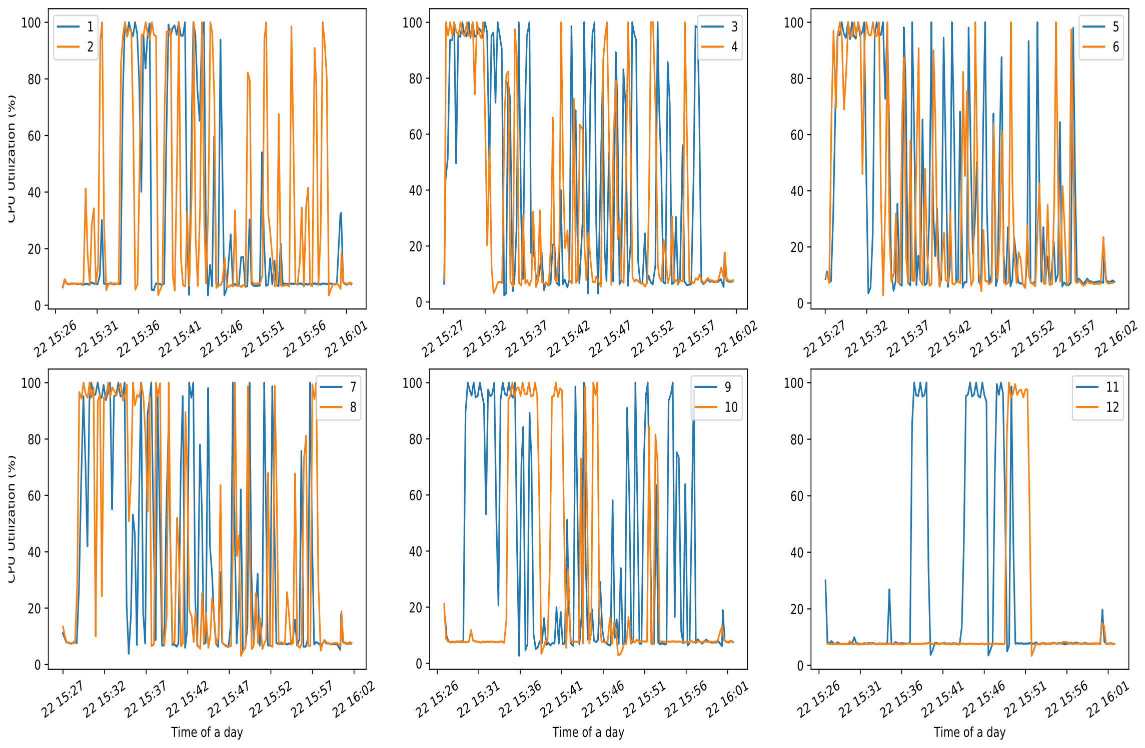
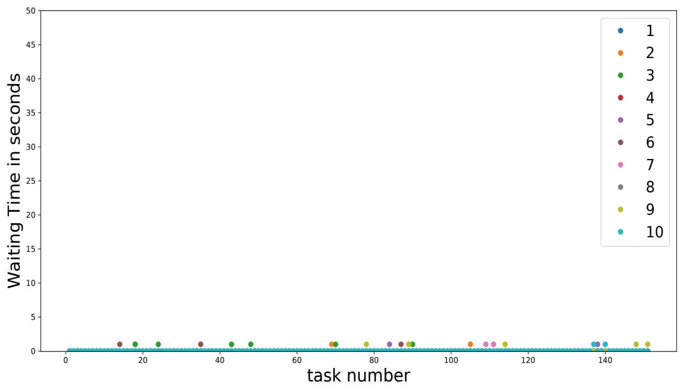
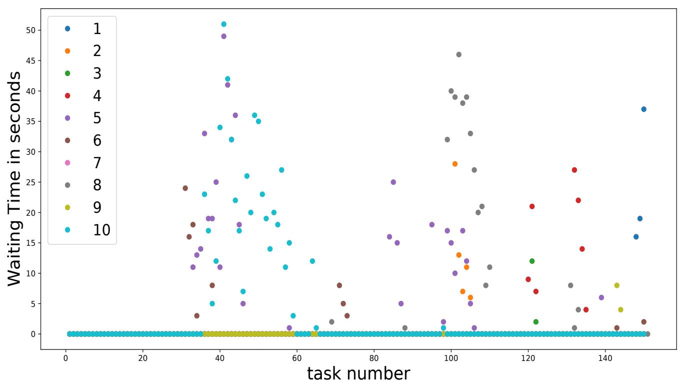


FIGURE 6.9: CPU utilization of the *W* approach in real-time multiprocessor allocation



(A) *W* approach.



(B) FT approach.

FIGURE 6.8: Waiting time of tasks in Sudoku puzzle.

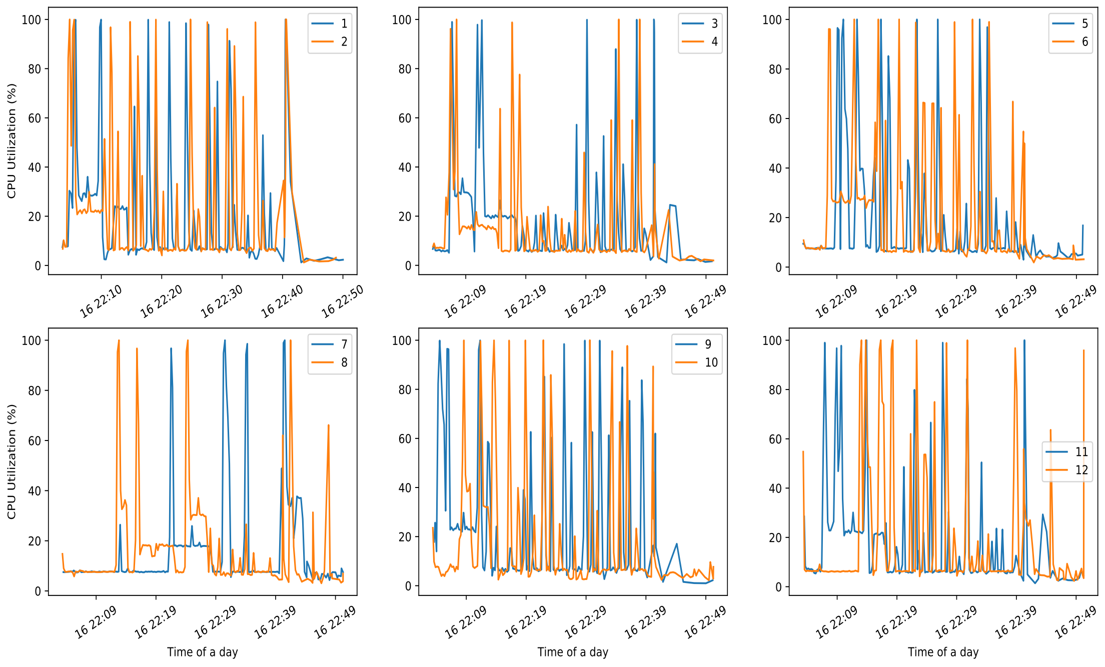


FIGURE 6.10: CPU utilization of the *WCPU* approach in real-time multiprocessor allocation

The same analysis was applied to the second case study (the Sudoku puzzle) provided in Figures 6.11 and 6.12. The same was observed in this case study when we control the CPU utilisation of the nodes. Thus, some tasks might utilise high CPU and finish the tasks before applying some control over the node exceeding the threshold.

Moreover, it can be observed that some nodes have been idle for some time in both case studies. The explanation for this lies in the process of allocating the tasks to a node. More specifically, the allocation technique is the first look for an idle node to handle the incoming task. If all the nodes are busy, then we apply the node interference approach. When more nodes become idle, the allocation technique will choose a random node. This randomness of choosing the idle node can result in some nodes becoming idle for some time.

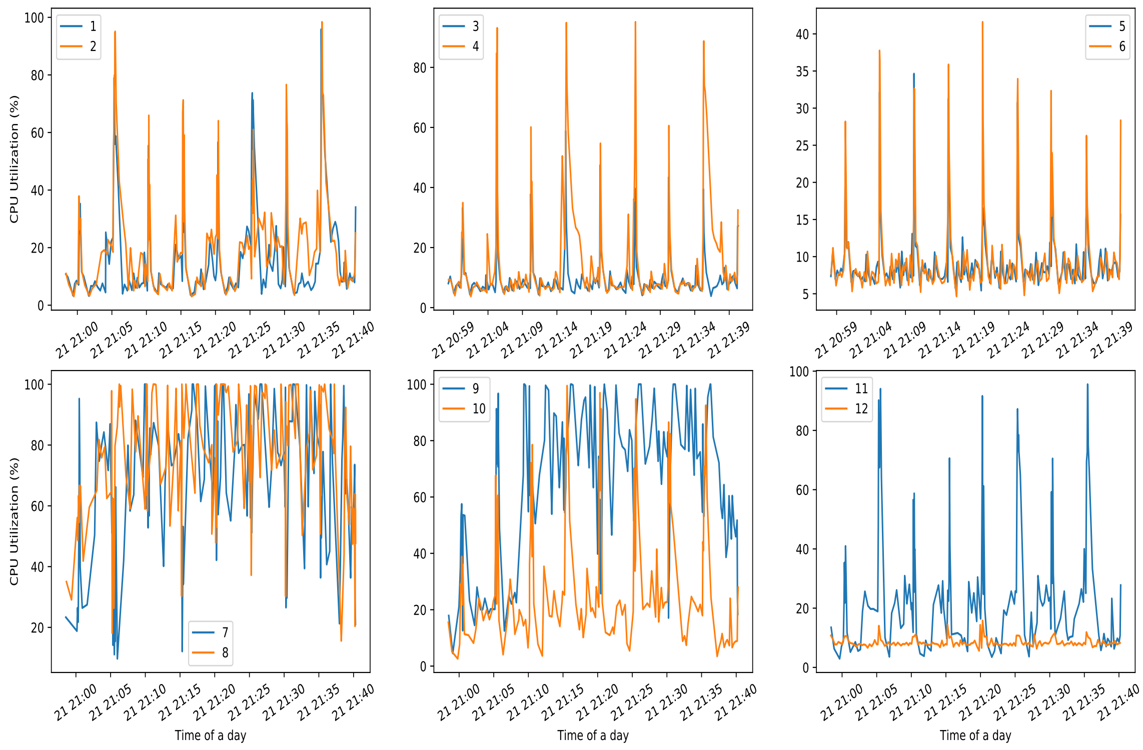


FIGURE 6.11: CPU utilization of the *W* approach in Sudoku puzzle.

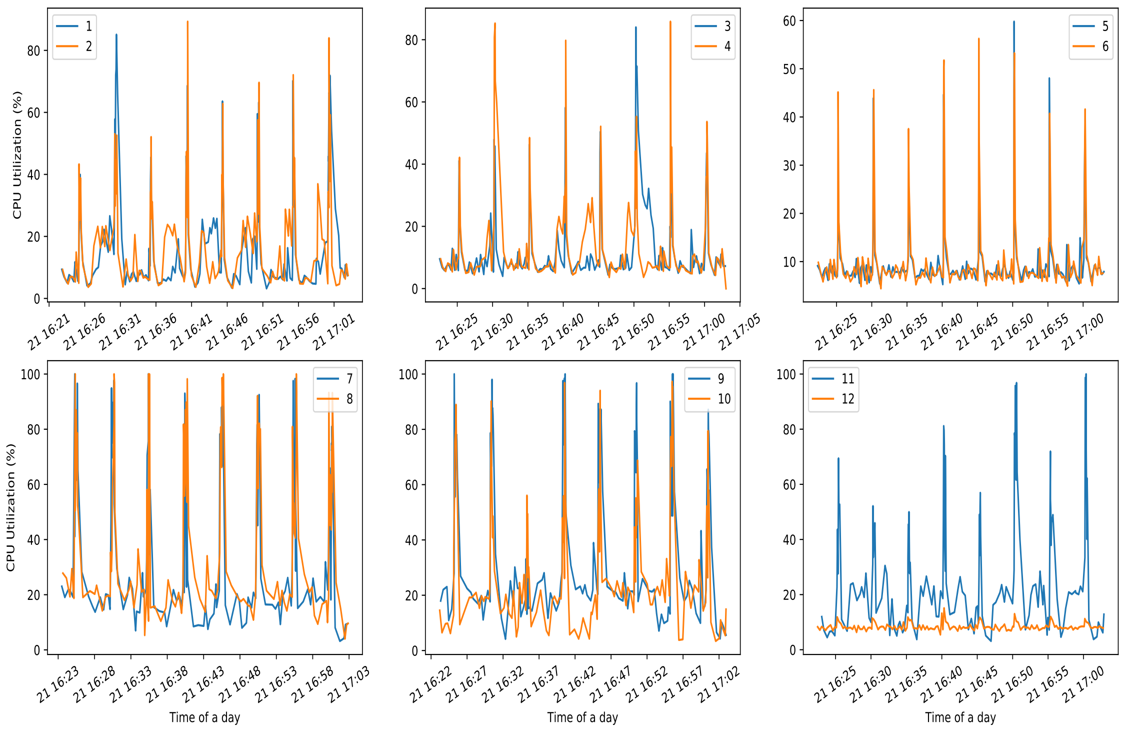


FIGURE 6.12: CPU utilization of the *WCPU* approach in Sudoku puzzle.

Furthermore, we carried out an analysis of the *NSTAD* approaches and compare it with FT, Docker Swarm and Kubernetes. Moreover, we applied this analysis in a real-time multiprocessor allocation case study. The *NSTAD* uses the equation from the previous section (equation 6.5) to allocate the new tasks to a node. It can be observed from the result in figure 6.13 that the approach achieved similar results as the FT approach. On the other hand, the approach outperforms Docker Swarm and Kubernetes. Although FT and *NSTAD* achieved a similar result, *NSTAD* has a higher median than FT approach.

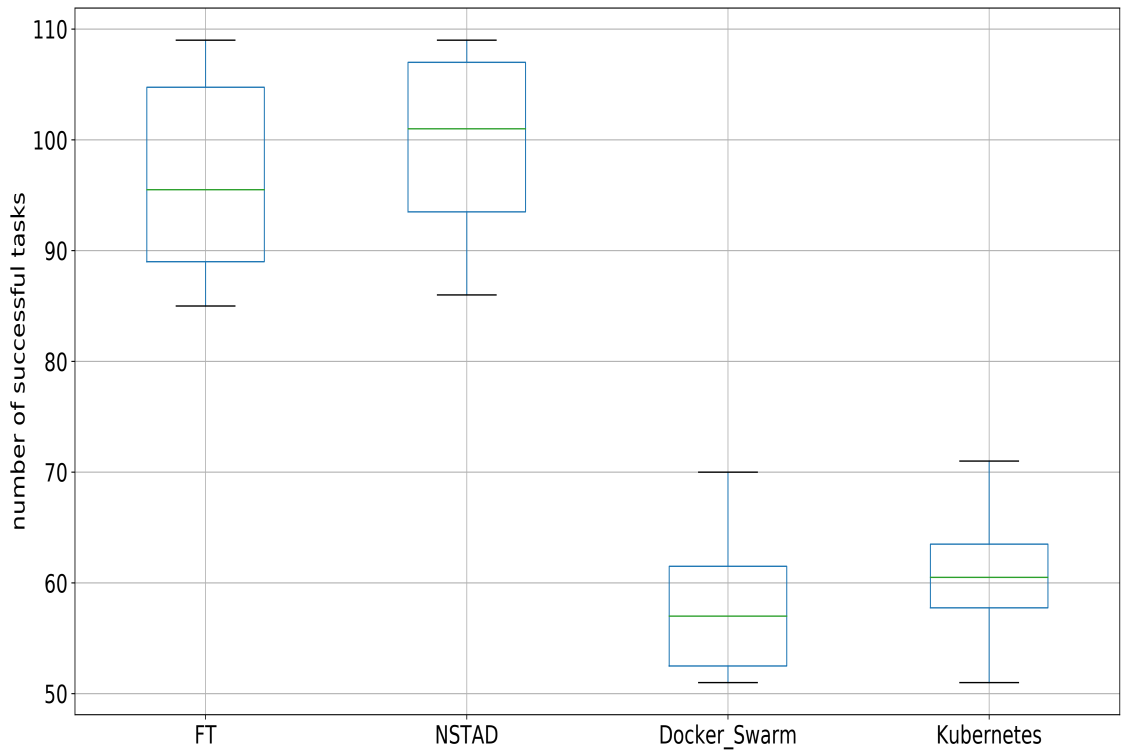


FIGURE 6.13: Result of 10 experiments of different approaches and different GA workloads (real-time multiprocessor allocation)

6.4 Summary

In summary, this work aims to manage the GA workload in a container-based environment by considering two case studies (real-time multiprocessors allocation and the Sudoku puzzle). As the previous approach (FT) had some limitations in handling node interference, this work introduces node interference based on the weight of the tasks in a node. Therefore, we have proposed fitness tracking based on the weight (W) approach to improve the number of tasks executed on time and achieve the fitness required by the user and allowing multiple tasks to be executed on time. After running 10 experiments and comparing W against the FT and state-of-the-art container-based orchestration systems approaches, the W approach, which does not control the CPU utilisation achieved a better result. In contrast, $WCPU$ was affected using CPU control over the tasks within a node that exceeded the threshold. Although $WCPU$ was affected by the CPU utilisation control, the approach was significant when we compared it against Docker Swarm and Kubernetes. The limitation of this work is adding a node interference feature considering task information with some control over the CPU utilisation when the node reached the threshold.

Chapter 7

Conclusions and Future Work

Cloud computing provides on-demand and pay-as-you-go services and resources. As a result, business providers such as Amazon and Google have moved to cloud computing, taking advantage of the resources and allowing many cloud users to access them. These resources are placed in a shared pooled which allow the cloud user to choose the suitable resources based on their needs. A cloud user can provide a workload to be executed that consists of numerous jobs and each job may contain several tasks some of which depend on other tasks. One specific load that this thesis handles is the GA workload. Based on two case studies: real-time multiprocessor allocation and Sudoku puzzle, the GA workload consists of numerous individual tasks. Each task has a certain deadline and fitness requirements, while the resource management must execute tasks and achieve the fitness required before the deadline.

Resource management is important for improving QoS between cloud providers and cloud users, and deciding on resource allocation and meeting QoS requirements. Resource management can manage resources using specific strategies to execute the task without compromising the QoS requirement based on the workload requested by the user. In order to improve resource management considering GA workload, this thesis considered strict deadlines and the fitness achieved as two features to increase the number of successful tasks. The number of successful tasks means that the resource management executed the task and achieved the fitness required by the user at a given deadline.

The container orchestrator is one technique used to deploy and manage the workload. For example, Docker Swarm and Kubernetes are considered to be state-of-the-art container-based orchestration systems. The limitation of these orchestrators is that they do not consider tasks' deadline or fitness required by the user when executing them. Thus, this thesis considered them as baselines to compare the proposed approaches. As explained in chapter 4, we proposed an approach that containerises the GA in such a way that the Docker Manager, which is part of the orchestrator, can instantiate containers to run the GA application and its parameters based on the user's requirements. Also, we introduce an orchestrator that can allocate the incoming tasks based on the information from the Node Observer. This information is similar to the number of tasks per node and CPU utilization.

The purpose of this thesis is to investigate two hypotheses on how to manage numerous instances of GAs running as containers in a cloud environment. These hypotheses are as follows:

- *QoS-based resource management can handle GA tasks to meet the application's hard real-time timing and user-defined fitness requirements. This resource management considers executing only one task at a time per cloud node and ensures that the tasks achieve the user-defined fitness level on time.*
- *Using a situation that is representative of a real scenario, the node interference used in resource management can lessen the effect when we execute multiple tasks per cloud node. The node interference considers the tasks' information like response time and fitness achieved to determine which cloud node can execute the incoming task from the user with negative impact on other tasks in the same cloud node.*

The first hypothesis was investigated in detail in chapter 5. In this chapter, several approaches were proposed to improve the number of successful tasks. These approaches are FT, FP and FPLR. The FT approach is used to track the fitness achieved as the tasks go through several iterations. FP and FPLR approaches use fitness prediction at a given deadline as an additional condition when executing the task to determine whether the task will achieve the fitness required by the deadline or not. After running numerous experiments in both case studies, the results show that the approaches have improved the number of successful tasks when compared to the baselines (Docker Swarm and Kubernetes). Thus, the FT approach has achieved 16% more successful tasks when compared to Docker Swarm and Kubernetes whereas FP achieved 21% more successful tasks. Further, FPLR achieved a higher percentage when compared to FT and FP, with 26% more successful tasks.

The second hypothesis was investigated in detail in chapter 6 which considers node interference. As the previous approaches assume that each cloud node will only do one task at a time, the node interference is considered to allow several tasks to use the same cloud node while having less of an impact on the currently executing tasks. As a result, the chapter presented a weighted-based node interference method that takes into account fitness achieved and response time at a particular iteration. The weight-based node interference is based on two features which is the slack time and difference in fitness achieved. These features were used in calculating the weight and represent *NST* and *NDF*. The results show that *W* achieved 20% more successful tasks when compared to Docker Swarm and Kubernetes, whereas *NST* and *NDF* achieved lower percentages in terms of the number of successful tasks with 12% in *NDF* and 9% in *NST*.

7.1 Future Work

The work presented in this thesis considered the management of container-based GA workload in a cloud environment using two case studies. Having different approaches proposed in this area is important for future work. As the approaches in both works achieved a better result than the baselines, several areas and features can be exploited. One direction for further research is considering different constraints, such as soft deadlines and allowing the task to miss the deadline with a specific margin to achieve the fitness required.

Further research direction concerning node interference. The work undertaken in this thesis is related to node interference and considers information that is from the task executed such as the response time and fitness achieved. However, the approach did not consider any information from the actual node such as CPU utilization and memory. The benefit of considering information from the node is allowing the new task to be executed in a node that is most likely to have sufficient resources to achieve the user requirement.

The previous directions including the work undertaken in this thesis did not include the scalability of the infrastructure. Thus, another direction is scalability, which can be undertaken either by vertical or horizontal scaling. Vertical scaling is when we add more CPU or memory to the existing node, whereas horizontal scaling is undertaken by adding another node to the cluster. Both ways can improve resource management and provide a chance for the task to achieve user-defined requirements.

During the process of predicting the fitness required, the decision of whether to update the coefficients of the linear or quadratic terms can be based on the prediction error. If the error remains small, the model currently in use might be sufficient. However, if the error starts to increase, indicating a departure from the current model's accuracy, the system can initiate a transition to the alternative model.

Integrating linear and quadratic models in fitness forecasting combines their strengths to create a more robust and adaptable prediction strategy. The dynamic transition between models based on problem-solving characteristics enhances accuracy and ensures a better fit to the changing fitness landscape. The success of this integration lies in the careful definition of transition thresholds to optimize the switching process.

The concepts and insights presented by Anderson et al. in [100] can have valuable implications for the cloud computing research community, even though the paper itself focuses on real-time systems. Some of these insights are that the dynamic allocation of resources to jobs and workloads is a feature of cloud computing. New resource allocation techniques in cloud environments may be influenced by their proposed method, which achieves an ideal balance between predictability and effectiveness. Cloud providers can improve the trade-off between completing tasks on the deadline and optimizing resource utilization by adapting their proposed approach.

Furthermore, cloud services often operate under Service Level Agreements (SLAs), similar to real-time systems meeting task deadlines. The optimization-based algorithm's ability to minimize task deadline misses aligns with cloud providers' goal of ensuring SLA compliance. Adapting this principle could help cloud services consistently meet user expectations and contractual agreements.

7.1.1 Generalizations to different workloads

While the current thesis focuses on different approaches that try to achieve the fitness required by a given deadline, future research could explore the applicability of these approaches to a wider range of computational workloads. One workload might be a complex workload with dependencies. The approach then needs to deal with a job that may contain several tasks and needs to consider two deadlines one of which is a task deadline and the other the job deadline. Investigating the effectiveness of the proposed methods across different problem domains and complexities would provide valuable insights into the generalizability of the approach.

Certain workloads might have varying resource utilization patterns over time. For instance, in cloud computing environments, resource allocation can change dynamically based on demand fluctuations. Generalizing the fitness prediction models involves considering how these resource utilization patterns influence the model's ability to forecast fitness accurately. Adaptive coefficient updating strategies might be necessary to accommodate such dynamics.

In this thesis, it was noted that the linear and quadratic fitness prediction models could potentially be brought into a unified procedure to enhance forecasting accuracy. Combining both models in a cohesive manner might involve adaptive switching between linear and quadratic models based on the characteristics of the computation process. Exploring this integration could lead to a more adaptable fitness prediction approach that adapts to different problem-solving scenarios.

References

[1] Reinhard Wilhelm, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, and Reinhold Heckmann. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):1–53, 2008.

[2] Min Du and Feifei Li. ATOM: Efficient Tracking, Monitoring, and Orchestration of Cloud Resources. *IEEE Transactions on Parallel and Distributed Systems*, 28(8):2172–2189, 2017.

[3] Giovanni Mariani, Andreea Anghel, Rik Jongerius, and Gero Dittmann. Predicting Cloud Performance for HPC Applications: A User-Oriented Approach. *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, pages 524–533, 2017.

[4] Muhammad Bilal Qureshi, Maryam Mehri Dehnavi, Nasro Min-Allah, Muhammad Shuaib Qureshi, Hameed Hussain, Ilias Rentifis, Nikos Tziritas, Thanasis Loukopoulos, Samee U. Khan, Cheng Zhong Xu, and Albert Y. Zomaya. Survey on Grid Resource Allocation Mechanisms. *Journal of Grid Computing*, 12(2):399–441, 2014.

[5] Amit Kumar Singh, Piotr Dziurzanski, Hashan Roshantha Mendis, and Leandro Soares Indrusiak. A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems. *ACM Computing Surveys*, 50(2):1–40, 2017.

[6] In Kee Kim, Jacob Steele, Yanjun Qi, and Marty Humphrey. Comprehensive elastic resource management to ensure predictable performance for scientific applications on public IaaS clouds. *Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014*, pages 355–362, 2014.

[7] A.T. Saraswathi, Y.R.A. Kalaashri, and S. Padmavathi. Dynamic Resource Allocation Scheme in Cloud Computing. *Procedia Computer Science*, 47:30–36, 2015.

[8] Abdullah Yousafzai, Abdullah Gani, Rafidah Md Noor, Mehdi Sookhak, Hamid Talebian, Muhammad Shiraz, and Muhammad Khurram Khan. Cloud resource allocation schemes: review, taxonomy, and opportunities. *Knowledge and Information Systems*, 50(2):347–381, 2017.

[9] Bin Sun, Brian Hall, Hu Wang, Da Wei Zhang, and Kai Ding. Benchmarking private cloud performance with user-centric metrics. *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014*, pages 311–318, 2014.

[10] Vahid Arabnejad, Kris Bubendorfer, and Bryan Ng. Budget and Deadline Aware e-Science Workflow Scheduling in Clouds. *IEEE Transactions on Parallel and Distributed Systems*, 30(1):29–44, 2019.

[11] Yali Zhao, Rodrigo N. Calheiros, Graeme Gange, Kotagiri Ramamohanarao, and Rajkumar Buyya. SLA-Based Resource Scheduling for Big Data Analytics as a Service in Cloud Computing Environments. *Proceedings of the International Conference on Parallel Processing*, 2015-December(1):510–519, 2015.

[12] Danilo Ardagna, Giuliano Casale, Michele Ciavotta, J.F. Pérez, and Weikun Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 5(1):1–17, 2014.

[13] Mujahid Tabassum and Kuruvilla Mathew. a Genetic Algorithm Analysis Towards Optimization Solutions. *International Journal of Digital Information and Wireless Communications*, 4(1):124–142, 2014.

[14] Piotr Dziurzanski, Jerry Swan, and Leandro Soares Indrusiak. Value-based manufacturing optimisation in serverless clouds for industry 4.0. *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18*, pages 1222–1229, 2018.

[15] Leandro Soares Indrusiak. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of Systems Architecture*, 60(7):553–561, 2014.

[16] Saad Mustafa, Babar Nazir, Amir Hayat, Atta ur Rehman Khan, and Sajjad A Madani. Resource management in cloud computing: Taxonomy, prospects, and challenges. *Computers & Electrical Engineering*, 47:186–203, 2015.

[17] Brendan Jennings and Rolf Stadler. Resource Management in Clouds: Survey and Research Challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

[18] S S Manvi and G Krishna Shyam. Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. *Journal of Network and Computer Applications*, 41(1):424–440, 2014.

[19] G. Da Cunha Rodrigues, R.N. Calheiros, V.T. Guimaraes, G.L. Dos Santos, M.B. De Carvalho, L.Z. Granville, L.M.R. Tarouco, and R. Buyya. Monitoring of cloud computing environments: Concepts, solutions, trends, and future directions. *Proceedings of the ACM Symposium on Applied Computing*, 04-08-Apri:378–383, 2016.

[20] Tarek Mahdhi and Haithem Mezni. A prediction-Based VM consolidation approach in IaaS Cloud Data Centers. *Journal of Systems and Software*, 146:263–285, 2018.

[21] Abdul Hameed, Alireza Khoshkbarforoushha, Rajiv Ranjan, Prem Prakash Jayaraman, Joanna Kolodziej, Pavan Balaji, Sherali Zeadally, Qutaibah Marwan Malluhi, Nikos Tziritas, Abhinav Vishnu, Samee U. Khan, and Albert Zomaya. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 98(7):751–774, 2016.

[22] Fábio D. Rossi, Miguel G. Xavier, César A.F. De Rose, Rodrigo N. Calheiros, and Rajkumar Buyya. E-eco: Performance-aware energy-efficient cloud data center orchestration. *Journal of Network and Computer Applications*, 78(October 2016):83–96, 2017.

[23] Amit Kumar Singh, Piotr Dziurzanski, and Leandro Soares Indrusiak. Value and energy optimizing dynamic resource allocation in many-core HPC systems. *Proceedings - IEEE 7th International Conference on Cloud Computing Technology and Science, CloudCom 2015*, pages 180–185, 2016.

[24] Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescape. Cloud monitoring: Definitions, issues and future directions. *2012 1st*

IEEE International Conference on Cloud Networking, CLOUDNET 2012 - Proceedings, pages 63–67, 2012.

[25] Sena Seneviratne, David C Levy, and Rajkumar Buyya. A Taxonomy of Performance Prediction Systems in the Parallel and Distributed Computing Grids. *CoRR*, abs/1307.2, 2013.

[26] Maryam Amiri and Leyli Mohammad-Khanli. Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications*, 82(October 2016):93–113, 2017.

[27] Isam Mashhour Al Jawarneh, Paolo Bellavista, Filippo Bosi, Luca Foschini, Giuseppe Martuscelli, Rebecca Montanari, and Amedeo Palopoli. Container Orchestration Engines: A Thorough Functional and Performance Comparison. *IEEE International Conference on Communications*, 2019-May:1–6, 2019.

[28] Maria A. Rodriguez and Rajkumar Buyya. Container-based Cluster Orchestration Systems: A Taxonomy and Future Directions. (April):1–19, 2018.

[29] Jay Shah and Dushyant Dubaria. Building modern clouds: Using docker, kubernetes google cloud platform. *2019 IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC 2019*, pages 184–189, 2019.

[30] Akshay Dhumal and Dharanipragada Janakiram. C-Balancer: A System for Container Profiling and Scheduling. pages 1–10, 2020.

[31] Rene Peinl, Florian Holzschuher, and Florian Pfitzer. Docker Cluster Management for the Cloud - Survey Results and Own Solution. *Journal of Grid Computing*, 14(2):265–282, 2016.

[32] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[33] Abraham Jaison, N. Kavitha, and P. S. Janardhanan. Docker for optimization of Cassandra NoSQL deployments on node limited clusters. *Proceedings of IEEE International Conference on Emerging Technological Trends in Computing, Communications and Electrical Engineering, ICETT 2016*, 2017.

[34] Min Xia, Teng Li, Yunfei Zhang, and Clarence W. De Silva. Closed-loop design evolution of engineering system using condition monitoring through internet of things and cloud computing. *Computer Networks*, 101:5–18, 2016.

[35] Mehdi Bahrami and Mukesh Singhal. A dynamic cloud computing platform for eHealth systems. *2015 17th International Conference on E-Health Networking, Application and Services, HealthCom 2015*, pages 435–438, 2015.

[36] L. Minh Dang, Md Jalil Piran, Dongil Han, Kyungbok Min, and Hyeonjoon Moon. A survey on internet of things and cloud computing for healthcare. *Electronics (Switzerland)*, 8(7):1–49, 2019.

[37] Shreshth Tuli, Rakesh Tuli, Shikhar Tuli, and Sukhpal Singh Gill. Predicting the growth and trend of COVID-19 pandemic using machine learning and cloud computing. *Internet of Things*, 11, 2020.

[38] Mohd Javaid, Abid Haleem, Raju Vaishya, Shashi Bahl, and Rajiv Suman. Diabetes & Metabolic Syndrome : Clinical Research & Reviews Industry 4 . 0 technologies and their applications in fighting COVID-19 pandemic. *Diabetes & Metabolic Syndrome: Clinical Research & Reviews*, 14(4):419–422, 2020.

[39] Md. Imran Alam, Manjusha Pandey, and Siddharth S Rautaray. A Comprehensive Survey on Cloud Computing. *International Journal of Information Technology and Computer Science*, 7(2):68–79, 2015.

[40] Aaqib Rashid and Amit Chaturvedi. Cloud Computing Characteristics and Services A Brief Review. *International Journal of Computer Sciences and Engineering*, 7(2):421–426, 2019.

[41] Manisha T. Tapale, Mahantesh N. Birje, Praveen S. Challagidad, and R.H. Goudar. Cloud computing review: concepts, technology, challenges and security. *International Journal of Cloud Computing*, 6(1):32, 2017.

[42] Misbah Liaqat, Victor Chang, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Toseef, Umar Shoaib, and Rana Liaqat Ali. Federated cloud resource management: Review and discussion, 2017.

[43] Parminder Singh, Avinash Kaur, Pooja Gupta, Sukhpal Singh Gill, and Kiran Jyoti. RHAS: robust hybrid auto-scaling for web applications in cloud computing. *Cluster Computing*, 24(2):717–737, 2021.

[44] Frederic Nzanywayingoma and Yang Yang. Efficient resource management techniques in cloud computing environment: a review and discussion. *International Journal of Computers and Applications*, 41(3):165–182, 2019.

[45] Muhammad Ibrahim, Said Nabi, Abdullah Baz, Hosam Alhakami, Muhammad Summair Raza, Altaf Hussain, Khaled Salah, and Karim Djemame. An In-Depth Empirical Investigation of State-of-the-Art Scheduling Approaches for Cloud Computing. *IEEE Access*, 8:128282–128294, 2020.

[46] Absa Stephen, Shajulin Benedict, and R. P.Anto Kumar. Monitoring IaaS using various cloud monitors. *Cluster Computing*, 22(s5):12459–12471, 2019.

[47] Ahmed Barnawi, Sherif Sakr, Wenjing Xiao, and Abdullah Al-Barakati. The views, measurements and challenges of elasticity in the cloud: A review. *Computer Communications*, 154(December 2019):111–117, 2020.

[48] Piotr Dziurzanski and Leandro Soares Indrusiak. Value-Based Allocation of Docker Containers. 2018.

[49] Pieter Jan Maenhaut, Bruno Volckaert, Veerle Ongenae, and Filip De Turck. Resource Management in a Containerized Cloud: Status and Challenges. *Journal of Network and Systems Management*, 28(2):197–246, 2020.

[50] S. R. Shishira, A. Kandasamy, and K. Chandrasekaran. Survey on meta heuristic optimization techniques in cloud computing. *2016 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2016*, pages 1434–1440, 2016.

[51] Kashif Hussain, Mohd Najib Mohd Salleh, Shi Cheng, and Yuhui Shi. Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, 52(4):2191–2233, 2019.

[52] Sašo Karakatič and Vili Podgorelec. A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing Journal*, 27:519–532, 2015.

[53] Simranjit Kaur, Pallavi Bagga, Rahul Hans, and Harjot Kaur. Quality of Service (QoS) Aware Workflow Scheduling (WFS) in Cloud Computing: A Systematic Review. *Arabian Journal for Science and Engineering*, 44(4):2867–2897, 2019.

[54] Daniel A. Menascé. Workload Characterization. *Internet Computing, IEEE*, 7(6):89 – 92, 2016.

[55] Deborah Magalhães, Rodrigo N. Calheiros, Rajkumar Buyya, and Danielo G. Gomes. Workload modeling for resource usage analysis and simulation in cloud computing. *Computers and Electrical Engineering*, 47:69–81, 2015.

[56] Dror G Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. 2014.

[57] A. Burkimsher, I. Bate, and L.S. Indrusiak. A characterisation of the workload on an engineering design grid. *Simulation Series*, 46(5), 2014.

[58] Cláudio Maia, Marko Bertogna, Luís Nogueira, and Luis Miguel Pinho. Response-Time Analysis of Synchronous Parallel Tasks in Multiprocessor Systems. *Proceedings of the 22nd International Conference on Real-Time Networks and Systems - RTNS '14*, pages 3–12, 2014.

[59] Mustafizur Rahman, Rafiul Hassan, Rajiv Ranjan, and Rajkumar Buyya. Adaptive workflow scheduling for dynamic grid and cloud computing environment. *Concurrency Computation Practice and Experience*, 22(6):685–701, 2010.

[60] a Burkimsher. Dependency Patterns and Timing for Grid Workloads. *Proceedings of the Fourth York Doctoral Symposium on Computer Science*, (October):25–33, 2011.

[61] Mariela Curiel and Ana Pont. Workload Generators for Web-Based Systems: Characteristics, Current Status and Challenges. *IEEE Communications Surveys and Tutorials*, 20(2):1526–1546, 2018.

[62] Hugo E.S. Galindo, Wagner M. Santos, Paulo R.M. Maciel, Bruno Silva, Sérgio M.L. Galdino, and José Paulo Pires. Synthetic workload generation for capacity planning of virtual server environments. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pages 2837–2842, 2009.

[63] Christophe Cérin, Tarek Menouer, Walid Saad, and Wiem Ben Abdallah. A New Docker Swarm Scheduling Strategy. *Proceedings - 2017 IEEE 7th International Symposium on Cloud and Service Computing, SC2 2017*, 2018-Janua:112–117, 2018.

[64] Zhang Wei-guo, Ma Xi-lin, and Zhang Jin-zhong. Research on kubernetes' resource scheduling scheme. *ACM International Conference Proceeding Series*, pages 144–148, 2018.

[65] Naylor G Bachiega, Paulo S L Souza, Sarita M Bruschi, and Simone R S De Souza. Container-based Performance Evaluation : A Survey and Challenges. *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018.

[66] Sogand Shirinbab, Lars Lundberg, and Casalicchio Emiliano. Performance Evaluation of Container and Virtual Machine Running Cassandra Workload. *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*, 2017.

[67] Hadi Goudarzi and Massoud Pedram. Maximizing profit in cloud computing system via resource allocation. *Proceedings - International Conference on Distributed Computing Systems*, pages 1–6, 2011.

[68] Nima Kaviani, Eric Wohlstadter, and Rodger Lea. Profiling-as-a-service: Adaptive scalable resource profiling for the cloud in the cloud. *International Conference on Service-Oriented Computing*, 7084 LNCS:157–171, 2011.

[69] Rafael Weingärtner, Gabriel Beims Bräscher, and Carlos Becker Westphall. Cloud resource management: A survey on forecasting and profiling models. *Journal of Network and Computer Applications*, 47:99–106, 2015.

[70] Jie Yang, Jie Qiu, and Ying Li. A profile-based approach to just-in-time scalability for cloud applications. *CLOUD 2009 - 2009 IEEE International Conference on Cloud Computing*, pages 9–16, 2009.

[71] Naghmeh Dezhabad, Sudhakar Ganti, and Gholamali Shoja. Cloud Workload Characterization and Profiling for Resource Allocation. *Proceeding of the 2019 IEEE 8th International Conference on Cloud Networking, CloudNet 2019*, pages 2019–2022, 2019.

[72] Kaniz Fatema, Vincent C. Emeakaroha, Philip D. Healy, John P. Morrison, and Theo Lynn. A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives. *Journal of Parallel and Distributed Computing*, 74(10):2918–2933, 2014.

[73] Swarm mode key concepts. `https://docs.docker.com/engine/swarm/key-concepts/`. Accessed: 2019-05-23.

[74] Jonatan Enes, Guillaume Fieni, Roberto R. Exposito, Romain Rouvoy, and Juan Tourino. Power Budgeting of Big Data Applications in Container-based Clusters. *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, 2020-September:281–287, 2020.

[75] Qiang Duan. Cloud service performance evaluation: status, challenges, and opportunities – a survey from the system modeling perspective. *Digital Communications and Networks*, 3(2):101–111, 2017.

[76] Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Transactions on Cloud Computing*, 3(4):449–458, 2015.

[77] Farrukh Nadeem, Daniyal Alghazzawi, Abdulfattah Mashat, Khalid Fakeeh, Abdullah Almalaise, and Hani Hagras. Modeling and predicting execution time of scientific workflows in the Grid using radial basis function neural network. *Cluster Computing*, 20(3):2805–2819, 2017.

[78] Bashir Mohammed, Irfan Awan, Hassan Ugail, and Muhammad Younas. Failure prediction using machine learning in a virtualised HPC system and application. *Cluster Computing*, 2019.

[79] Steve Sorrell and Jamie Speirs. Hubbert's legacy: A review of curve-fitting methods to estimate ultimately recoverable resources. *Natural Resources Research*, 19(3):209–230, 2010.

[80] P. Srikanth, D. Rajeswara Rao, and P. Vidyullatha. Comparative analysis of ANFIS, ARIMA and polynomial curve fitting for weather forecasting. *Indian Journal of Science and Technology*, 9(15), 2016.

[81] Michael Li and Lily D. Li. A Novel Method of Curve Fitting Based on Optimized Extreme Learning Machine. *Applied Artificial Intelligence*, 34(12):849–865, 2020.

[82] Sheng Di, Derrick Kondo, and Walfredo Cirne. Google hostload prediction based on Bayesian model with optimized feature combination. *Journal of Parallel and Distributed Computing*, 74(1):1820–1832, 2014.

[83] Yexi Jiang, Chang Shing Perng, Tao Li, and Rong N. Chang. Cloud analytics for capacity planning and instant VM provisioning. *IEEE Transactions on Network and Service Management*, 10(3):312–325, 2013.

[84] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. *ACM Computing Surveys*, 47(4):1–33, 2015.

[85] Yun Huang and N. Venkatasubramanian. QoS-based resource discovery in intermittently available environments. *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*, 2002-January:50–59, 2002.

[86] Kallia Chronaki, Alejandro Rico, Marc Casas, Miquel Moretó, Rosa M. Badia, Eduard Ayguadé, Jesus Labarta, and Mateo Valero. Task scheduling techniques for asymmetric multi-core systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):2074–2087, 2017.

[87] Krzysztof Kalinowski, Damian Krenczyk, and Cezary Grabowik. Predictive - reactive strategy for real time scheduling of manufacturing systems. *Applied Mechanics and Materials*, 307:470–473, 2013.

[88] Piotr Dziurzanski and Amit Kumar Singh. Feedback-based admission control for firm real-time task allocation with dynamic voltage and frequency scaling. *Computers*, 7(2), 2018.

[89] Shuai Zhao, Piotr Dziurzanski, Michal Przewozniczek, Marcin Komarnicki, and Leandro Soares Indrusiak. Cloud-based Dynamic Distributed Optimisation of Integrated Process Planning and Scheduling in Smart Factories. *GECCO 2019 - Proceedings of the 2019 Genetic and Evolutionary Computation Conference*, pages 1381–1389, 2019.

[90] Pasquale Salza and Filomena Ferrucci. An Approach for Parallel Genetic Algorithms in the Cloud using Software Containers. pages 1–7, 2016.

[91] Prasad Devarasetty and Satyananda Reddy. Genetic algorithm for quality of service based resource allocation in cloud computing. *Evolutionary Intelligence*, 14(2):381–387, 2021.

[92] Syed Nasir Mehmood Shah, Ahmad Kamil Bin Mahmood, and Alan Oxley. Hybrid resource allocation method for grid computing. *2nd International Conference on Computer Research and Development, ICCRD 2010*, pages 426–431, 2010.

[93] Amit Kumar Singh, Piotr Dziurzanski, and Leandro Soares Indrusiak.

Market-inspired dynamic resource allocation in many-core high performance computing systems. *2015 International Conference on High Performance Computing & Simulation (HPCS)*, pages 413–420, 2015.

[94] Ming Hua Lin, Jung Fa Tsai, and Chian Son Yu. A review of deterministic optimization methods in engineering and management. *Mathematical Problems in Engineering*, 2012, 2012.

[95] Jun Tang, Gang Liu, and Qingtao Pan. A Review on Representative Swarm Intelligence Algorithms for Solving Optimization Problems: Applications and Trends. *IEEE/CAA Journal of Automatica Sinica*, 8(10):1627–1643, 2021.

[96] Anima Naik, Suresh Chandra Satapathy, and Ajith Abraham. Modified Social Group Optimization—a meta-heuristic algorithm to solve short-term hydrothermal scheduling. *Applied Soft Computing Journal*, 95:106524, 2020.

[97] Zhao Chenhong, Zhang Shanshan, Liu Qingfeng, Xie Jian, and Hu Jicheng. Independent tasks scheduling based on genetic algorithm in cloud computing. *Proceedings - 5th International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2009*, pages 9–12, 2009.

[98] Maciej Malawski, Kamil Figiela, and Jarek Nabrzyski. Cost minimization for computational applications on hybrid cloud infrastructures. *Future Generation Computer Systems*, 29(7):1786–1794, 2013.

[99] R. S. Shariffdeen, D. T.S.P. Munasinghe, H. S. Bhathiya, U. K.J.U. Bandara, and H. M.N.Dilum Bandara. Adaptive workload prediction for proactive auto scaling in PaaS systems. *Proceedings of 2016 International Conference on Cloud Computing Technologies and Applications, CloudTech 2016*, pages 22–29, 2017.

[100] James H. Anderson, Jeremy P. Erickson, Uma Maheswari C. Devi, and Benjamin N. Casses. Optimal Semi-Partitioned Scheduling in Soft Real-Time Systems. *Journal of Signal Processing Systems*, 84(1):3–23, 2016.