VICTORIA UNIVERSITY OF
WELLINGTON
TE HERENGA WAKA

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Internet: office@ecs.vuw.ac.nz

Dynamic multi-workflow scheduling for geo-distributed cloud services: an evolutionary reinforcement learning approach

Chuan Law

Supervisors: Aaron Chen & Hui Ma

Submitted in partial fulfilment of the requirements for
Master of Artificial Intelligence.

Abstract

This study extends the Self-Attention Policy Network for Cloud Workflow Scheduling (SPN-CWS) by incorporating geo-distributed environments, addressing the complexities of multi-region resource allocation and workflow execution. A simulation-based approach is employed to model distributed virtual machines (VMs) in the cloud, capturing latency effects, communication delays, and data transfer costs inherent to geographically dispersed workflows. To enhance scheduling accuracy, we introduce a region mismatch penalty, which discourages task execution on VMs outside their designated regions, thereby reducing cross-region data transfer overhead. The proposed framework integrates reinforcement learning techniques to balance cost-efficiency, SLA performance, and region-aware scheduling, optimizing decisions across multiple cloud regions. By applying this method to realistic workflow patterns such as CyberShake, the extended simulator provides a robust platform for analyzing the impact of geo-distributed scheduling, with fitness evaluation metrics designed to assess execution efficiency, cost-effectiveness, SLA adherence, and region-aware scheduling compliance.

Acknowledgments

I would like to give a special thanks to my parents who have sacrificed so much for me to be here today. Although you may not fully understand what I do, I am forever grateful for how you have always supported me without question.

To my friends in Wellington - Alex, Rei and Olivia - thank you for your constant encouragement and for keeping me grounded throughout the years. Your support and friendship means more to me than you know.

To my colleagues in Data Platform at Trade Me, thank you for always moving sprint ceremonies to accommodate my study days. Your understanding has played a crucial role in helping me succeed in both work and academia.

Finally, a thank you to Cherie for taking a chance on me back in 2019 during the Covid-19 pandemic. Your belief in me back then has set me on the path toward artificial intelligence and I wouldn't be studying this without you.

Contents

Chapter 1

Introduction

1.1 Problem Statement

Cloud computing adoption and usage is rapidly on the rise both within the traditional technology industry and also into newer sectors. Globally, there is a projected 25% increase from 2023 to 2024 in spending of Cloud System Infrastructure as a Service (IaaS) and a projected 30% increase for 2025 which signals the tremendous growing demand from consumers [1]. One of the most widely used in the industry is an offering that provides cloud based virtual compute engines or virtual machines (VM's) for rent on demand. VM's are part of the IaaS provided by all major cloud providers for example: Elastic Compute 2 instance (EC2) from Amazon Web Services (AWS), Azure Virtual Machine from Microsoft Azure and Google Compute Engine (GCE) from Google Cloud Platform (GCP).

In New Zealand, a country with historically no dedicated server locations from any major cloud providers is predicted to double its cloud spending from 2022 to 2026 from NZ$ 2.6 billion to NZ$ 5.1 billion [2]. Organizations use comparatively cost-effective VM's to carry out multitudes of tasks in scale without the need of physical hardware infrastructure. As the complexity of tasks grow, so too will the difficulty of using cloud compute in an optimal manner. Therefore there is a need for cost effective scheduling of dynamic computation-intensive jobs which also maintain operational commitments of service up-time. Workflow scheduling in the cloud is characterized by resource heterogeneity, task dependencies, and fluctuating compute workloads and present complex challenges that are difficult to optimize even for experienced professionals.

This is known as the deadline-constrained dynamic workflow scheduling (DCDWSC) problem, with the goal to optimally schedule tasks within specified time limits [3]. There are two main optimizations to be made which are categorized as cost and operational requirements [4]. Scheduling policies need to be created to act as a form of 'broker' to handle and control how and what virtual machine (VM) instances are spun up to complete workflows. Historically, these have required skilled domain experts well versed in the pricing of cloud compute offerings along with technical expertise to select the correct VM type to use.

Accompanying this problem is the need for using geographically distributed providers amongst the dynamic nature of workflow scheduling. When companies use public cloud, they tend to choose the datacenter closest to them for latency considerations.

"Trade Me's data is now served from Sydney [...] That was "inevitable" with all the players in Sydney at that point in time and also to meet the site's latency requirements." [5].

Despite being located in New Zealand and serving almost exclusively New Zealand users, companies like Trade Me need to use cloud resources located at suboptimal regions due to real world limitations in infrastructure available. These geographically distributed systems have more complex computational demands and data dependencies, where input datasets reside in different locations. This in turn requires more specialized and efficient scheduling methods to minimize additional inter-region data transfer costs and communication delay on top of total VM cost and SLA compliance.

This is the main influence behind Dynamic Multi-Workflow Scheduling Problem for Geo-Distributed Cloud Services. Geo-distributed systems are very common practice in most organizations but the related literature is lacking in optimization research for this highly complex but valuable work.

1.2 Motivations

The increasing complexity of multi-workflow scheduling in geo-distributed environments necessitates novel approaches that go beyond traditional heuristic and rule-based methods. Existing solutions often struggle with scalability, adaptability, and cost-efficiency, especially when dealing with unpredictable workloads and resource availability across geographically distributed cloud infrastructures.

This research is motivated by the need to leverage Evolutionary Reinforcement Learning (ERL) and Self-attention Policy Network for Cloud Workflow Scheduling (SPN-CWS) to improve the efficiency of workflow execution. By integrating region-aware evolutionary optimization with reinforcement learning, ERL provides a mechanism for dynamic adaptation to the highly complex nature of geographical resource constraints, while SPN-CWS can enhance the understanding of spatial dependencies and their relationship between VMs and workflows across multiple locations [6]. Together, these methods aim to optimize scheduling policies for cost reduction, inter-region constraints and keeping service level agreements in large-scale geo-distributed systems.

1.3 Goals and objectives

This project explores the current status of multi-workflow scheduling in geo-distributed data centers, and investigates how they can be optimized for cost with consideration to SLA adherence.

Our primary goal of this research is to develop and evaluate a new ERL method based on SPN-CWS with enhanced performance for workflow scheduling and execution across geo-distributed cloud environments. The study aims to:

- **Investigate existing workflow scheduling techniques** and analyze their limitations in handling dynamic, large-scale, multi-workflow environments.

- **Develop a new ERL-based adaptive scheduling approach** that optimizes geo-distributed scheduling strategies through evolutionary learning.

- **Extend SPN-CWS neural network input** to allow the neural network to learn and optimize for inter-region costs associated with geo-distributed systems.

- **Simulate and evaluate the proposed geo-distributed framework** against the original SPN-CWS neural network scheduling approaches to measure improvements in total rental costs, VM execution and SLA Penalty charges.

- **Identify trade-offs and constraints** in ERL and SPN-CWS-based scheduling, including computational overhead, generalization ability, and real-world feasibility.

By addressing these objectives, the research seeks to advance intelligent workflow scheduling techniques that balance performance, scalability, and cost in geo-distributed multi-workflow environments.

1.4 Report organization

The remainder of this report is structured as follows:

- **Chapter 2: Literature Review** – Provides an overview of existing workflow scheduling techniques, categorizing them into *static* and *dynamic* approaches. It explores key methodologies including *mathematical optimization, heuristic-based strategies, local search algorithms, genetic programming/genetic algorithms*, and *reinforcement learning*. The chapter also examines the unique challenges of *multi-workflow scheduling in geo-distributed cloud environments*, and defines the workflow scheduling problem within this context, outlining the core constraints, objectives, and evaluation metrics considered in this study.

- **Chapter 3: Proposed Solution** – Defines the *workflow scheduling problem in a geo-distributed setting*, detailing the key constraints, objectives, and performance metrics considered in this study.

- **Chapter 4: Implementation** – Describes the design and implementation of ERL and SPN-CWS-based scheduling models, including training strategies, neural network architectures, and reinforcement learning process. Additionally, it discusses the simulation environment and datasets used for evaluation.

- **Chapter 5: Experimental Results** – Presents experimental benchmarks and sensitivity analysis of our geo-distributed ERL and SPN-CWS-based scheduling compared against a single region baseline SPN-CWS.

- **Chapter 6: Conclusion and Future Work** – Summarizes key findings and outlines potential enhancements and real-world applications of ERL and SPN-CWS for cloud workflow scheduling.

Chapter 2

Literature Review

In this section a literature review explores the existing research on workflow scheduling and covers the key concepts and methodologies used by current studies. The review starts with the categorization of workflow scheduling approaches, focusing on the distinctions between static and dynamic scheduling while also discussing their respective advantages and limitations.

2.1 Background

There are multiple approaches to solving workflow scheduling problems, with static and dynamic scheduling being two of the primary categories [4]. Static Workflow Scheduling (SWS) approaches assume that information about workflows, such as their arrival time and the total number of workflows are known beforehand. Approaches are able to carry out global searches on the solution space at the workflow and task level which is then used to generate a scheduling policy in advance for use at the beginning of a workflow execution. However, as static scheduling assumes that all task execution and workflow information is readily available beforehand, static scheduling is not practical in a cloud environment where users can submit workflows consisting of various sizes and at unpredictable intervals [7], [8].

Most static workflow scheduling solutions instead focus on minimizing the makespan of a workflow while ignoring a few other important scheduling objectives such as total execution cost and SLA constraints. The makespan of a workflow is defined as the total time to complete all tasks in selected VM instances. However, for combinatorial problems most studies focus on dynamic workflow scheduling strategies to optimize important scheduling objectives prevalent in cloud computing [4].

Dynamic Workflow Scheduling (DWS) approaches consider fluctuating arrival times of workflows and resource availability which create many unique challenges. DWS is more aligned with solving common real world scenarios that require quick decision-making under resource heterogeneity and evolving task dependencies. Cloud resource heterogeneity itself is the presence of the vast amount of different cloud compute offerings available from current public cloud services. Table 5.1 details varying degrees of compute capabilities available from Amazon Web Services (AWS). Some are more suited for specific requirements such as higher storage. VM's that are provided from public cloud provide the benefit to end users by being vastly simplified operationally. They are rented out by the hour and charged accordingly by compute capability. This research focuses on hardware diversity, varied pricing models, location variability and performance optimization for the task at hand.

The scheduling of workflows involve allocating tasks to available resources (e.g., virtual machines in a cloud environment) while optimizing various performance objectives. These often include minimizing the makespan, reducing monetary cost and ensuring adherence to Quality of Service (QoS) requirements, such as Service Level Agreements (SLA's) for 99.9% workflow uptime. The inherent complexity of the workflow scheduling problem arises from the combinatorial nature of task-resource mappings and dependency constraints imposed by execution task dependencies. This is similar to the well known classical Job Shop Scheduling Problem (JSSP) where strict rules that dictate the specific order, machine type and activity rules of a job are used. Both workflow scheduling [8] and job shop scheduling problems are considered NP Hard problems due to the large search space of possible schedules that satisfy all constraints. JSSP in particular has been extensively investigated [9] and is considered a "strong" NP Hard problem where even small Job Shop problems are not guaranteed to have an optimal solution. Thus, there is no known polynomial-time algorithm capable of solving all instances of the problem efficiently. An extension to this is the Flexible Job Shop Problem (FJSSP) which allows an operation to be processed on any machine from a set of alternatives, with constraint considerations such as processing times or machine capabilities. Similarly for DWS, a task can be executed on multiple virtual machines (VMs) in the cloud depending on factors like compute power, cost, and latency. The modern use of cloud compute for DWS problems builds on top of existing JSSP optimizations.

2.2 Problem Definition

The aim for "Dynamic multi-workflow scheduling for geo-distributed cloud services" is to optimize the DWS problem with consideration to the physical requirements introduced by utilizing a geographically distributed cloud environment. We extend an evolution strategies-based RL [10] to train and evaluate a self-attention policy network SPN-CWS [11] for geographically distributed dynamic workflows. Critical workflows need to have high availability and by using distributed setups, these workflows are able to gain the benefit of 99% up-time availability [12] by taking advantage of region fail-over strategies to ensure continuity of processes. A globally distributed system also benefits users located in different locations as they are able to receive a quicker turnaround time for workflow requests if done within their region.

In this section we formally describe the distributed dynamic multi-workflow scheduling problem (DDMWS) as illustrated in Figure 2.1. This problem is an extension of recent work [11] which defines many shared terms such as execution time and the VM rental fee of the dynamic multi workflow scheduling problem. DDMWS builds on the idea of a *broker* acting as the central component that orchestrates which virtual machine (VM) instances are spun up to complete which tasks from multiple workflows but now within a location distributed system. The broker is now responsible for accepting workflow requests from users located in different regions and spinning up region specific VM's to complete them. This also involves allocating the tasks to existing leased VM's that may not be from the original users region. The goal of DDMWS is still aligned with related work in DWS to minimize the total cost of both the monetary VM rental fees and the SLA violation penalties.

Cloud Environment Model

In the cloud environment, a workflow consists of a sequence of tasks represented as a directed acyclic graph (DAG) where each node represents a task. This follows the representation used by the vast majority of existing works [7], [13]. A task is only marked as ready when all its parent tasks are complete (see: figure 2.2). Tasks in a workflow are

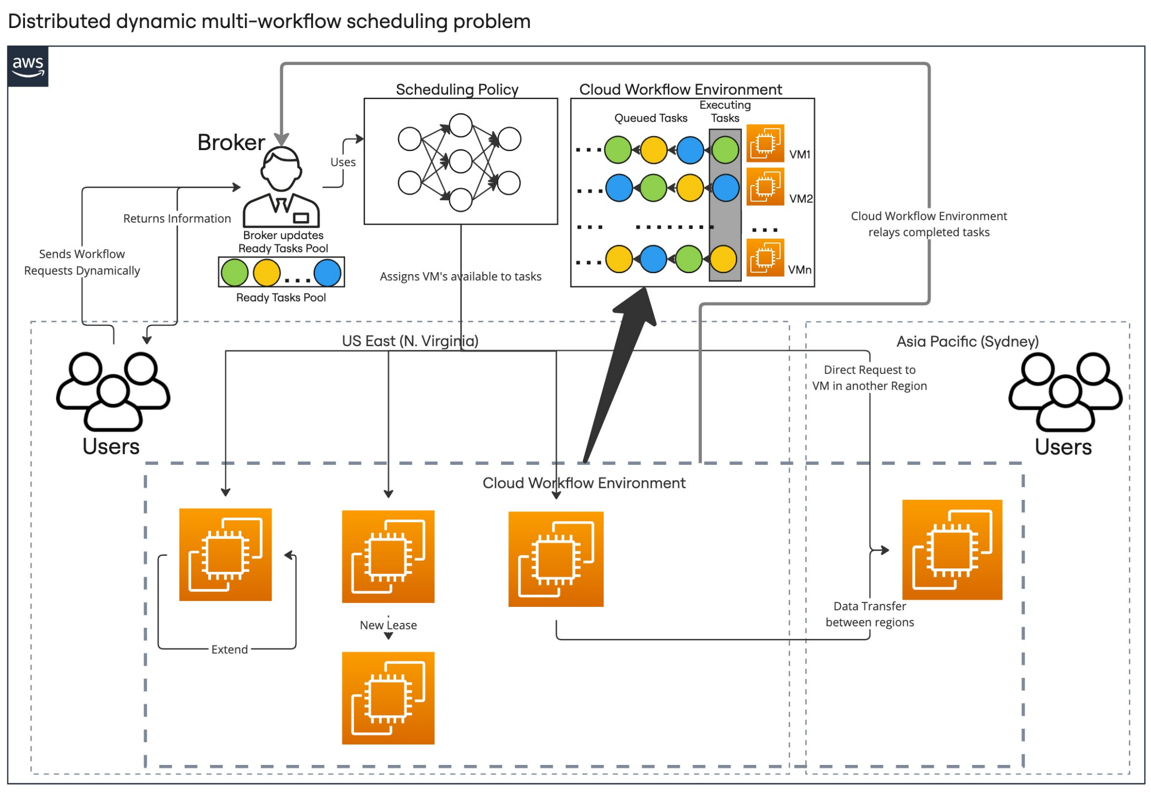
Distributed dynamic multi-workflow scheduling problem
aws
Broker
Uses
Scheduling Policy
Cloud Workflow Environment
Queued Tasks
Executing Tasks
VM1
VM2
VMn
Returns Information
Broker updates Ready Tasks Pool
Ready Tasks Pool
Sends Workflow Requests Dynamically
Assigns VM's available to tasks
Cloud Workflow Environment relays completed tasks
US East (N. Virginia)
Asia Pacific (Sydney)
Direct Request to VM in another Region
Users
Users
Cloud Workflow Environment
Extend
New Lease
Data Transfer between regions

Figure 2.1: How a broker handles dynamically arriving workflow requests from geographically distant users using VM's distributed across two regions.

executed by a set of available VM's located in different regions. The type of VM chosen is determined by the learning policy produced by the ERL system. In this study, the types of VM available for use by the policy are restricted to only the AWS 'm5' EC2 instance family as detailed in 5.1, but the quantity available for each type is unlimited. This means that the broker can lease an arbitrary number of these VM instance types on demand without any restrictions. A VM instance defined as v can be characterized as follows:

$$v = \{VMType(v), VMCapacity(v), VMHourlyRate(v), VMLocation(v)\} \tag{2.1}$$

where $VMType(v)$ indicates the VM type, $VMCapacity(v)$ the computation capability per time unit of v, $VMHourlyRate(v)$ the the rental fee per hour to lease v and $VMLocation(v)$ being the geographical region VM v is located. According to documentation [14] and existing work [11] [8] [15], an hour is the minimum usage rate that is charged and thus even a fraction of usage within the hour will still charged as the full hour.

Workflow

In DDMWS, we assume that a finite number of workflow sequences containing a set of tasks are submitted unpredictably by users to a broker over time T. Let $W = \{w_1, w_2, \ldots, w_N\}$ represent the set of workflows, where each workflow w_i consists of a set of dependent tasks, SLA configurations and time of arrival. A workflow w can be defined as follows:

$$w = [DAG(w), AT(w), DL(w), \beta(w) | w \in W(T)] \tag{2.2}$$

where $\beta(w)$ is the user specified SLA penalty rate (see Eq. 2.14). $AT(w)$ is the arrival time of w, and $DL(w)$ is the user specified SLA deadline (see Eq. 2.12). $DAG(w)$ defines a directed acyclic graph (DAG) of tasks that represents all the tasks and their dependencies to be executed as part of workflow w (see Eq. 2.4). The following formulations describe each component considered in a workflow from the submission start to completion.

Task

Each workflow will have a varying number of tasks depending on its size. We denote t_n as the n-th task in a workflow with $Task(w)$ representing the set of all tasks in a workflow w_i:

$$Task(w_i) = \{t_1, t_2, \ldots, t_n\} \tag{2.3}$$

where each $t \in Task(w)$ is defined as a piece of work of size $Size(t)$ quantified as the total amount of computation required for a VM to complete task t.

In each workflow w_i, the dependent tasks are connected and represented by a DAG (see: figure 2.2) in which each node of the graph represents a task that needs to be executed as part of a workflow. A DAG representation is crucial to the representation as it allows any task $t \in Task(w)$ to be represented as a downstream dependency of another task t. The DAG of workflow w comprising of all tasks $Task(w)$ is represented as:

$$DAG(w) = [Task(w), Edge(w)] \tag{2.4}$$

Each direct edge $(t, t') \in Edge(w)$ connects one task $t \in Task(w)$ to another task $t' \in Task(w)$. Formally indicating that t is a predecessor task of t′ and t′ is a successor/parent task of t. Any task will become ready for execution only when all of its predecessor tasks are completed or no predecessor tasks exist.

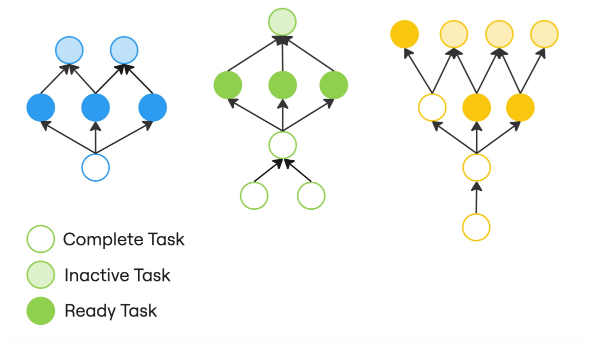

Figure 2.2: Representation of a workflow DAG.

Latency

In a distributed environment, each task t_i contains $Location(t)$ which determines the region it is executed on. Assuming same region transfer has negligible latency, we will only consider cases of inter-region transfer where additional execution time caused by latency will need to be accounted for.

The total transmission time between two regions can be determined by considering both the data transfer speed and the network latency. $CommunicationDelay$ is thus referred to as the time in seconds (s) to transmit the data from one task to another based on bandwidth ($DataSize/Bandwidth$) plus the network latency:

$$CommunicationDelay(t',t) = \left(\frac{DataSize(t',t)}{Bandwidth(v)}\right) + Latency(r_a, r_b) \tag{2.5}$$

where:

- $DataSize(t',t)$: Size of data in bits to be transferred between a predecessor t' and successor t pair of tasks (defined as part of our proposed solution in: 3.2).
- $Bandwidth(v)$: Network bandwidth capability of VM in Gigabits per seconds (Gbps).
- $Latency(r_a, r_b)$: Network Latency in seconds (s) between source region r_a and target region r_b.

Inter-Region Data Transfer Cost

Inter-region data transfer is the movement of data from a source region that is not the same as the target region. Following [16], inter-region data transfer incurs a higher cost that varies on the source region being used. The total data transfer cost $DT(t_i)$ for task t is defined as:

$$DT(t) = \sum_{t' \in Succ(t)} \left(\frac{\text{DataSize}(t,t')}{10^9}\right) \times \begin{cases} \text{WithinRegionCost}(r_a) & \text{if } r_a = r_b \\ \text{InterRegionCost}(r_a, r_b) & \text{if } r_a \neq r_b \end{cases} \tag{2.6}$$

where:

- (r_a) represents the source region a and (r_b) represents the target region b.
- $t \in \text{Succ}(t')$: The summation over the set of all successor tasks of predecessor task t'
- WithinRegionCost(r_a): Cost of data transfer within the same region per gigabyte (GB).
- InterRegionCost(r_a, r_b): Cost of data transfer between two regions per gigabyte (GB).

Execution Time Definition

In recent work [11] the execution time of of task t is defined as:

$$EXT(t, w, \pi) = \frac{\text{Size}(t)}{\text{VMCapacity}(v_{t,w,\pi})} \tag{2.7}$$

where:

- $Size(t)$: the total amount of computation required to complete task.
- (t, w, π): a task t that is part of workflow w selected using scheduling policy π
- (v_t, w, π): a VM v_t that is part of workflow w selected using scheduling policy π

In DWS, workflows arrive at the broker dynamically over time, which require policy π to make its scheduling decisions adaptively. In order to reduce the VM rental fees and the possibility of incurring SLA penalties, every ready task must be scheduled for execution as soon as possible when possible (ie. when all predecessor tasks are complete or do not exist). The following formulation captures the completion time of a ready task $t \in Task(w)$ of workflow w:

$$CT(t, w, \pi) = ST(t, w, \pi) + EXT(t, w, \pi) \tag{2.8}$$

where:

- $CT(t, w, \pi)$: completion time of a task in workflow w selected using policy π.
- $ST(t, w, \pi)$: start time of a task in workflow w selected using policy π

Hence, the completion time of workflow w under policy π is:

$$WCT(w, \pi) = \max_{t \in Task(w)} CT(t, w, \pi) \tag{2.9}$$

In DDMWS, effective execution time of a task not only depends on the completion times of the task but also the latency of data transfer between dependent tasks. Communication delay caused by data transfer and inter-task communication is incorporated into the workflow completion time $WCT(w, \pi)$ for workflow w under policy π as:

$$WCT(w, \pi) = \max_{t \in Task(w)} \left(CT(t, w, \pi) + \max_{t' \in Pred(t)} \left(CT(t', w, \pi) + CommunicationDelay(t', t) \right) \right) \tag{2.10}$$

where:

- $Pred(t)$: The set of immediate predecessor tasks of task t.
- $CommunicationDelay(t', t)$: The communication delay between a predecessor task t' to the current task t.

VM Rental Fee

DDMWS focuses on handling multiple workflow processes that arrive dynamically, each with their own collection of tasks that are scheduled to be executed across multiple VM's. The total rental cost of all VM's are calculated when all workflows are marked as completed and the difference between the end time $t_e(v, \pi, T)$ and start time $t_s(v, \pi, T)$ of the execution period associated with all workflows carried out across all VM's.

In a distributed environment the VM's being used can be located geographically far away from each other. When a VM needs to send data outbound to another VM for processing that is not located within the geographical region variable additional costs are incurred by the cloud provider depending on the source region. This is known as inter-region data transfer and it is a key cost that was previously excluded from existing studies. The total rental cost must now include costs associated with transferring data between different regions. Additionally, communication delay as described in equation (2.5) is included as part of the difference between the end time and start time of using a VM v. Another consideration from distributed environments is that the existing infrastructure available in certain regions can result in their base VM rental cost per hour being substantially different from one another. The total rental cost of all VM's needed to complete a full instance of distributed workflow scheduling problem T can be formulated as:

$$TotalVMFee(\pi, T) = \sum_{v \in Set(\pi, T)} \left(\text{VMHourlyRate}(v) \cdot \left\lceil \frac{t_e(v, \pi, T) - t_s(v, \pi, T)}{3600} \right\rceil + \sum_{T_j \in \text{Tasks}(v)} DT(T_j) \right) \tag{2.11}$$

where:

- $Set(\pi, T)$: The set of all VMs used under scheduling policy π during the time period T.
- $\text{VMHourlyRate}(v_t)$: The hourly rental cost of VM v_t.
- $DT(T_j)$: The total data transfer cost for task T_j.
- $\sum_{T_j \in \text{Tasks}(v)}$: The summation iterates over all tasks T_j executed on VM v_t.

SLA Penalty Definition

Workflows used in real systems are often associated with a user-defined concept called Service Level Agreements (SLA's). For cloud workflow scheduling, this means that tasks need to be completed before a specified workflow deadline in order to avoid any SLA violation penalties. Practically, this means ensuring all required tasks are complete before the hourly rental has ended in order to prevent needing to rent a whole new hour for the VM. The latest time to complete tasks without incurring penalties is defined as an SLA deadline, i.e $DL(w)$, is determined as:

$$DL(w) = AT(w) + \gamma \times MinMakespan(w) \tag{2.12}$$

where:

- $AT(w)$: Arrival time of workflow w.
- γ: Relaxation coefficient which controls the leniency of the workflow deadline. See experiment setup 5.1 for how this is utilized in DDMWS.
- $MinMakespan(w)$: The hypothetical shortest time duration required for processing w, using the fastest VM to complete all tasks without any delay.

With a geo-distributed workflow, we must extend the SLA Penalty definition to account for the communication delay introduced with inter-region task execution. We define the $latency_{\text{Penalty}}$ as:

$$Latency_{\text{Penalty}} = \sum_{(t,t')\in E} CommunicationDelay(t,t') \times latencyPenaltyFactor \tag{2.13}$$

where:

- $latencyPenaltyFactor$: New scaling factor introduced to penalize workflows with high total communication delay, defined in implementation step 4

Following recent studies [8], [11], the SLA penalty of a geo-distributed workflow is now calculated together with a penalty applied to tasks that incur communication delay. This is represented as:

$$SLA_{\text{Penalty}}(w,\pi) = \begin{cases} 0 & \text{if: } WCT(w,\pi) < DL(w) \\ \beta(w) \times [WCT(w,\pi) - DL(w) + Latency_{\text{Penalty}}]\} & \text{else:} \end{cases} \tag{2.14}$$

where:

- $\beta(w)$: Penalty coefficient for the workflow with higher values correlating directly with the criticality of a workflow.

Optimal Scheduling Policy

We aim to find an optimal scheduling policy π to minimize the total cost with consideration to distributed workflows included as part of the TotalVMFee, SLA_{Penalty} formulated as:

$$\arg\min_{\pi} TotalCost(\pi) = \arg\min_{\pi} \left\{ TotalVMFee(\pi,T) + \sum_{w\in W(T)} SLA_{\text{Penalty}}(w,\pi) \right\} \tag{2.15}$$

where:

- $W(T)$: The set of workflows containing tasks in T.

2.3 Related Work

In recent years, the two most important and commonly explored objectives to optimize when minimizing total cost is the monetary rental costs of VM's and their trade-off with SLA penalties [11] [8] [17]. Critical level work with strict completion policies frequently prioritize meeting deadlines over cost savings and these policies aim to avoid SLA violations at all costs, even if it means renting additional VM hours or over-provisioning resources [18]. Due to resource elasticity of cloud compute, research into how this can be used to make superior scheduling policies has been growing in popularity [4].

This section covers the extensive (but not exhaustive) methods used in previous research with a perspective of usage within static and dynamic workflow scheduling problems. While all of them can be used on either static (SWS) and dynamic (DWS) problems, using certain methods over others is better suited in some cases due to the nature of the problem. Here we discuss in varying levels of effectiveness: Mathematical, Heuristics Based, Local Search, Genetic Programming (GP), Genetic Algorithms (GA) and Reinforcement Learning (RL) methods to build scheduling policies.

2.3.1 Mathematical Methods

Static Methods

Mathematical optimization techniques, such as linear programming (LP) and integer linear programming (ILP), are widely used for static scheduling with great success. In recent work [19] linear equations are modelled using variables such as time restrictions and the grouping and ordering of tasks.

Significant work such as GRP-HEFT [20] where Faragardi et al. introduce an extension of the Heterogeneous Earliest Finish Time (HEFT) algorithm, tailored to address workflow scheduling in Infrastructure-as-a-Service (IaaS) cloud environments under budget constraints. GRP-HEFT dynamically provisions resources and schedules tasks to minimize makespan while adhering to a strict budget. It demonstrated improved makespan and cost efficiency compared to traditional HEFT and other baseline methods, however it assumes accurate task execution and resource cost estimates which limits its adaptability to highly dynamic, real-time workloads.

Dynamic Methods

Mathematical methods lack adaptability, making them harder to use for handling real-time workflow changes and the dynamic arrival of tasks. One of the most successful methods "Real-Time Multiple-Workflow Scheduling (RMWS) scheme" from [17] is designed to adapt to fluctuations in arriving workloads and resource availability by using a probabilistic upward ranking strategy to calculate the sub-deadline of each task for the priority of VM's to be selected. The algorithm considers all the features in the cloud environment with a focus on the fluctuations in VM performance and can be divided into three parts: workflow pre-processing, VM allocation and feedback processing to achieve a joint optimization of makespan, rental cost and resource utilization. Experiments found this mathematical approach was able to achieve a better total rental cost and resource utilization compared with two popular baseline algorithms ROSA and NOSF.

Table 2.1: Summary of Mathematical Workflow Scheduling Methods

Paper	Method Type	Objective(s)	Geographically Distributed	Adaptivity	Cost Awareness	Key Contributions
Faragardi et al. [20]	GRP-HEFT	Makespan, Budget	No	Static	Yes	Extension of HEFT algorithm which adds budget-awareness in workflow scheduling in IaaS clouds with improved cost efficiency.
Bochenina et al. [19]	Static Scheduling	Deadlines, Heterogeneity	No	Static	No	Optimized static scheduling with task grouping and deadline constraints.
Ma et al. [17]	Heuristic-Based Scheduling	Cost, Deadlines	No	Dynamic	Yes	Proposes a Real-time Multiple-Workflow Scheduling (RMWS) scheme that dynamically schedules workflows with varying deadlines to minimize costs. achieved superior performance over two state-of-the-art algorithms for cost and deadline adherence.

2.3.2 Heuristics Based

Another significant area of research related to mathematical methods is heuristic-based algorithms. While not strictly mathematical optimization techniques, heuristics provide efficient, rule-based approaches for solving complex scheduling problems. Greedy algorithms are commonly used for simpler scheduling tasks, where they prioritize selecting the maximum number of dependent tasks that can be completed within a given time frame. These heuristic approaches trade off optimality for speed and adaptability, making them particularly useful in scenarios where exact mathematical optimization may be computationally expensive or infeasible.

Closely related to heuristic-based methods are manual scheduling approaches, which rely on predefined rules and expert knowledge rather than algorithmic decision-making.

Static Methods

Manual methods are particularly effective in controlled static environments, where domain expertise can be leveraged to create scheduling policies tailored to specific workload characteristics. For example, the well known manual approach using heterogeneity-aware policies from [21] that are designed explicitly to address performance and cost trade-offs in a data analytics based system. While results are effective, they require significant domain expertise and can be tedious to maintain. Manual methods often include difficult and often tedious methods that perform well on ideal scenarios but may drop in effectiveness as the system faces changes that the scheduling policy has not accounted for.

While manual scheduling policies are effective at solving for deterministic processes, they are usually less applicable to real-world scenarios where operational conditions are often changing and users are able to send into workflow requests at random times. This is likely the reason for the rise in popularity in using artificial intelligence approaches to optimizing scheduling policies for a dynamic workflow scenario. As described earlier, resource heterogeneity makes this problem complex.

Dynamic Methods

Manual approaches are less effective in dynamic scenarios due to their lack of adaptability. They perform poorly when workflows change unpredictably or arrive at random intervals.

Table 2.2: Summary of Heuristic-Based Workflow Scheduling Methods

Paper	Method Type	Objective(s)	Geographically Distributed	Adaptivity	Cost Awareness	Key Contributions
Chen et al. [22]	TraDE	Network Traffic, Latency	Yes	Dynamic	Yes	Network and traffic-aware adaptive scheduling for microservices under dynamic conditions.
Ma et al. [17]	Dynamic Scheduling	Makespan, Multi-Workflow	No	Dynamic	No	Real-time workflow scheduling for multiple workflows in cloud environments.
Liu et al. [23]	Online Scheduling	Flexibility, Makespan	No	Dynamic	No	An online multi-workflow scheduling approach for IaaS cloud that handles uncertain task execution times effectively.
Yu and Shi [24]	Planner-Guided Scheduling	Multi-Workflow, Performance	No	Static	No	A planner-guided scheduling strategy to optimize the execution of multiple workflow applications in parallel computing environments.
Arabnejad et al. [25]	Budget and Deadline-Aware Scheduling	Budget, Deadline	No	Dynamic	Yes	Developed a budget and deadline-aware scheduling approach for e-science workflows in cloud environments, ensuring cost efficiency while meeting deadlines.
Lee et al. [21]	Manual, Rule-Based	Performance, Cost	No	Dynamic	Yes	Heterogeneity-aware scheduling for cost-performance trade-offs.

2.3.3 Local Search

Local search is a metaheuristic optimization technique used to find good solutions for complex problems by iteratively improving a candidate solution through small modifications. Instead of searching the entire solution space, local search explores neighboring solutions and moves toward better ones based on a defined objective function. In workflow scheduling, tasks must be assigned to computing resources while optimizing objectives such as execution time, cost, and resource utilization.

Static Methods

Local search algorithms are generally quite effective for small static problems where standard greedy approaches can be used effectively. However to solve for large complex problems, non greedy methods must be considered.

Historically the Tabu Search algorithm proposed by Hurink et al. [26] has been used for complex JSSP optimizations. The key contribution of this work lies in the integration of adaptive tabu list management, specialized neighborhood structures, and a graph representation to effectively navigate the complex solution space. The tabu list proposed is based off of the concept of something being 'taboo' and prevents repeating recent bad decisions in the search space by remembering what was done. This allows for efficient exploration to new, better solutions. Experimental evaluations on benchmark job-shop scheduling problems demonstrate that the TS-based approach outperforms existing methods, effectively minimizing makespan while efficiently handling multi-purpose machine constraints in scheduling. The Tabu Search successfully optimizes task execution by leveraging adaptive search control and dynamic machine reassignment, making it well-suited for complex scheduling environments.

Dynamic Methods

Dynamic extensions of local search methods exist, but they are generally less flexible than AI-based methods.

Xing et al. [27] proposed a Local Search driven Periodic Scheduling algorithm, LSPS, that is designed to handle dynamic instances. Using workflows that have deadlines and random task runtime, LSPS selects a bag of ready tasks during periods without data-dependent constraints. The proposed LSPS incorporates a monitoring system for real-time information gathering. Specifically, it employs two monitors. The first monitor detects the arrival of new workflows, assigns sub-deadlines to all their tasks, and places them into the task pool. The second monitor identifies the start of a new scheduling period, at which point all ready tasks in the task pool are selected. A preliminary schedule is then generated using a heuristic rule for all ready tasks.the local search algorithm iteratively refines the schedule until a predefined stopping condition is met. In other words, the proposed LSPS algorithm improves workflow scheduling efficiency by dynamically adapting to task uncertainties while optimizing resource utilization and reducing costs. Experimental evaluations conducted in real cloud platforms demonstrated that LSPS effectively reduces monetary costs and improves resource efficiency compared to existing methods ROSA, CWSA and EPSM.

Jihad et al. [28] proposed an Enhanced Iterated Local Search (EILS) algorithm to improve the efficiency of scientific workflow scheduling in cloud environments. The main contribution of this work lies in the integration of multiple local search strategies, including hill climbing, simulated annealing, and tabu search, to enhance exploration and exploitation in the search space. The algorithm introduces an adaptive perturbation mechanism, which dynamically adjusts search intensification and diversification to escape local optima, improving the convergence rate. EILS also optimizes for task execution ordering and resource allocation by iteratively refining the solution based on cost and makespan objectives. The experimental results indicate that EILS outperforms traditional local search methods by achieving a more balanced trade-off between execution time and cost, making it particularly effective for scientific workflows with large-scale task dependencies and high computational demands.

Qin et al. [29] introduced the Adaptive Iterated Local Search (AILS) algorithm to tackle the budget-constrained workflow scheduling problem in cloud computing environments.

The AILS framework enhances the traditional Iterated Local Search by adding adaptive perturbation mechanism that dynamically adjusts both the resource pool and task permutations. This effectively balances exploration and exploitation within the search space. Additionally, an intensification strategy is employed to optimally assign tasks to leased resources, ensuring adherence to budget constraints while minimizing execution time. The authors also utilize a Markov chain model to analyze the convergence properties of the AILS algorithm, providing a theoretical foundation for its performance. AILS outperforms existing scheduling methods, achieving superior results in terms of cost efficiency and makespan reduction. In essence, the proposed AILS algorithm offers a robust solution for budget-aware workflow scheduling, adeptly managing resource allocation and task scheduling to meet financial constraints without compromising performance.

Table 2.3: Summary of Local Search Methods

Paper	**Method Type**	**Objective(s)**	**Geographically Distributed**	**Adaptivity**	**Cost Awareness**	**Key Contributions**
Hurink et al. [26]	Tabu Search	Makespan	No	Static	No	Tabu search optimization for static job-shop scheduling problems.
Qin et al. [29]	Local Search	Budget-Constrained Scheduling	No	Dynamic	Yes	An Adaptive Iterated Local Search (AILS) algorithm that dynamically adjusts search parameters to meet budget constraints in workflow scheduling within cloud environments.
Xing et al. [27]	Local Search	Scheduling with Random Task Runtimes	No	Dynamic	Yes	A local search-driven periodic scheduling approach designed to handle workflows with stochastic task runtimes in cloud environments.
Jihad et al. [28]	Local Search	Makespan, Cost	No	Dynamic	Yes	An Enhanced Iterated Local Search (EILS) algorithm for scheduling scientific workflows in cloud environments, aiming to reduce both execution time and cost. The approach integrates multiple local search techniques, including hill climbing, simulated annealing, and tabu search

2.3.4 Genetic Programming (GP) and Genetic Algorithms (GA)

Static Methods

GP methods are traditionally used for offline evolution of static heuristics but are less common in purely static scheduling due to limited scope for heuristic improvement. GA instead is the more preferred solution for static scheduling. It remains a critical area of research, particularly in homogeneous parallel systems and job-shop scheduling. Two studies, Al Na'mneh et al. [30] and Luchoomun et al. [31], explore genetic algorithm (GA)-based approaches for optimizing static scheduling problems. While both papers apply GA for precomputed scheduling their applications differ significantly. Al Na'mneh & Darabkh (2013) is better suited for parallel computing environments, where uniform resource availability allows for simpler scheduling decisions. In contrast, Luchoomun et al. handle greater complexity, making their hybrid GA more versatile for real-world scheduling challenges. These findings suggest that while GA-based scheduling is highly effective, problem-specific enhancements are crucial for achieving optimal results in different static scheduling scenarios.

Dynamic Methods

Hyper-heuristics are a well researched and effective approach with various recent works exploring its effectiveness at improving manually designed heuristics [32] [33]. The dynamic nature of the workflow arrival times also makes GP a good candidate as it is has been shown to effectively optimize combinatorial problems.

In [32] the authors acknowledge the existence of many effective manually designed heuristics but with most lacking consideration for deadline penalties in the SLA of dynamic workflow scheduling problems. Their novel approach Dynamic Workflow Scheduling Genetic Programming (DWSGP) aims to minimizing both rental fees and SLA penalties to better suit heterogeneous workflows commonly seen the the cloud. Challenges unique to GP such as what to use as terminal and function nodes and creating an accurate fitness evaluation problem are specifically challenging when applied in the context of workflow scheduling. Unique to cloud workflows, another challenge is deciding whether to rent a new VM or extend existing VM's. DWSGP evolves a scheduling heuristic that allocates each ready workflow task to one VM instance in real-time. The scheduling rule is also used to calculate the priority values of all VM's for the current task. Like other GP methods for workflow scheduling DWSGP represents individuals as a GP tree and in this context a GP tree represents a scheduling policy. The evolved GP trees are evaluated by a simulated cloud workflow environment that evaluates the scheduling policy. The GP tree is evolved using standard methods crossover, mutation and reproduction operators and makes use of elitism to take the best 3 individuals to be re-trained to the next generation. Notably DWSGP adds three new parameters into the terminal set that were not previously considered by existing work: Completion ratio of the i-th workflow (WCR_i), Utilization ratio of the k-th VM instance (VMU^k) and Remaining Available Time of the k-th VM instance ($VMRT^k$). And a new function set parameter *If-then-else* also not considered previously. Results from this study showed that the three new terminal nodes proved to be important when building scheduling policies that outperformed other popular baselines such as HEFT, FCFS and CHP.

In [34] GP is used in a similar setup by defining terminal and function nodes based off of workflow scheduling parameters. A simulated cloud environment is also used to evaluate performance. In this research the authors propose the Dynamic Workflow Scheduling Genetic Programming (DSGP) algorithm to dynamically design scheduling heuristics which minimize the average total makespan. Distinctly tasks with a higher number of children in the corresponding workflows are prioritized with the fact that once they are completed, a larger number of child tasks will become ready for processing by the Workflow Scheduler, which allows for more tasks to begin execution throughout the workflows. By evolving heuristics, the DSGP algorithm can adapt to changing conditions of unpredictable workflow arrival and resource availability in the cloud. Results show DSGP was able to create heuristics that outperform manually designed and widely used algorithms such as FCFS and HEFT. Tasks are scheduled according to the value calculated by the heuristic with the smaller the value is resulting in the faster it will be executed. This is the study's version of a priority concept and shown to be effective. Analysis and frequency counts of DSGP created heuristics and their generated features show smaller Expected Completion Time of a task and a higher VM compute speed can lead to a smaller value calculated. Analyzing the frequency of features in well performing heuristics was found that that rules generated by DSGP created a positive changes towards a scheduling heuristic. The Genetic Programming Based Hyper Heuristic Approach concludes that heuristics generated by DSGP are capable of creating feature combinations that optimize the makespan better than some well known manually designed scheduling policies.

In the recent paper Yang et al. [33], expands on dynamic workflow scheduling policy solutions by extending the existing Dual-Tree Genetic Programming (DTGP) algorithm that combines virtual machine selection rules (VMSR) and task selection rules (TSR) with a novel adaptive mutation algorithm. The research introduced 3 new probability vectors that enable the enhanced algorithm to evolve scheduling heuristics based on the impact of each tree produced. Key GP mechanisms such as the 1/5 rule is used to update these new vectors when evolving the selection rules. This adaptive mutation mechanism adjusts mutation rates based on the performance of individuals in the population which promotes diversity in new solutions and also minimizes premature convergence. Experimental results demonstrate that the proposed approach outperforms existing methods in terms of meeting deadlines and optimizing resource utilization.

Table 2.4: Summary of Genetic Programming-Based Workflow Scheduling Methods

Paper	Method Type	Objective(s)	Geographically Distributed	Adaptivity	Cost Awareness	Key Contributions
Al-Mouhamed et al. (2014) [30]	Genetic Algorithm	Makespan, Resource Utilization	Yes	Static	No	Proposed a genetic-based scheduling algorithm for static tasks in heterogeneous distributed systems, introducing customized genetic operators to optimize execution time and resource allocation.
Luchoomun et al. (2014) [31]	Genetic Algorithm	Makespan Optimization	No	Static	No	Enhanced genetic algorithm for static job-shop scheduling, introducing adaptive genetic operators and hybrid local search to improve solution quality and convergence speed.
Yang et al. [33]	GPHH	Makespan, Utilization	No	Dynamic	Yes	A novel adaptive mutation mechanism that improves the effective dual-tree genetic programming for dynamic scheduling.

Continued on next page

Paper	Method Type	Objective(s)	Geographically Distributed	Adaptivity	Cost Awareness	Key Contributions
Xu et al. [35]	Genetic Programming	Flexibility, Throughput	No	Dynamic	No	Ensemble-based evolution for flexible job shop scheduling.
Escott et al. [34]	GPHH	Makespan, Heterogeneity	No	Dynamic	Yes	Hyper-heuristic approach for dynamic workflow scheduling with improved heterogeneity handling.
Marko Đurasević et al. [36]	Ensemble Collaboration Method	Dynamic Scheduling	No	Dynamic	No	A novel ensemble collaboration method where dispatching rules are applied independently at each decision point to create simulations of schedules which enhances performance in dynamic scheduling scenarios.
Yang et al. [32]	Budget-Aware Scheduling	SLA, Budget, Heterogeneity	No	Dynamic	Yes	Dynamic workflow scheduling for heterogeneous resources under SLA and budget constraints.
Su Nguyen et al. [37]	Genetic-based Constraint Programming	Resource-Constrained Job Scheduling	No	Dynamic	Yes	A genetic programming algorithm to discover efficient search strategies for constraint programming in resource-constrained job scheduling.

2.3.5 Reinforcement Learning (RL)

Reinforcement learning has emerged as a popular method for solving the vast domain of scheduling problems. Classical problems such as Job Shop Scheduling (JSSP) and its extension Flexible Job Shop Scheduling (FJSSP) are both popular and effective

Traditional scheduling methods, such as heuristics or mathematical programming, often struggle when faced with uncertainty and dynamic constraints.

Static Scheduling

RL is rarely used in static scheduling since pre-trained policies cannot adapt to dynamic operational changes. Static RL policies focus on pre-execution optimization. However there have been some studies that make use of static scheduling results for benchmarking. In [38] Bader et al. propose a reinforcement learning (RL)-based approach to optimize CPU and memory resource allocation for scientific workflows in cloud environments. The study explores Gradient Bandits and Q-Learning as RL methods to improve efficiency in Nextflow, a widely used workflow management system. The primary goal of the research is to minimize resource wastage while maintaining high computational performance. The proposed methods are evaluated on predefined scientific workflow scenarios, making this study one of the rare instances where RL is applied to a static scheduling problem. Experimental results show that both RL techniques outperform traditional fixed-resource allocation strategies, and significantly reduce CPU and memory over-provisioning. Compared to a state-of-the-art feedback-loop-based baseline, the RL models achieve superior resource efficiency without compromising execution time which highlights one of the many advantages of using static scheduling for RL.

Similarity for (JSSP) Liu et al. [39] introduce an Actor-Critic Deep Reinforcement Learning approach and Unlike traditional heuristics, their method learns optimal scheduling decisions through policy-gradient optimization, allowing it to generalize across different job configurations. The framework is evaluated using static JSSP benchmarks, achieving notable performance gains over conventional heuristic-based methods. The key contribution lies in its ability to improve makespan efficiency while maintaining scalability for complex scheduling tasks.

Dynamic Scheduling

Reinforcement Learning has been increasingly applied to a variety of scheduling problems beyond workflow scheduling as it offers dynamic and adaptive solutions to complex optimization challenges [40]. Scheduling problems, which involve allocating resources over time to perform a collection of tasks, are prevalent in other fields such as manufacturing with job shop scheduling, project management. In a comprehensive RL literature review [41] the authors highlight the vast amounts of existing works specifically for job shop scheduling problems.

Many real world factors such as electricity considerations and hardware maintenance are removed by moving away from on premise infrastructure. Thus all WS research considers the pricing paid to cloud providers first and foremost in determining the effectiveness of the scheduling solution.

This is shown in recent existing works [11] [8] which focus on the concept of a "cost-aware" system using reinforcement learning techniques to learn scheduling policies to optimize the VM usage in a way that minimizes rental fees. It is suggested in [11] to invoke SLA violation penalties to be more cost effective as these penalties usually end up being cheaper than renting out large, faster VM's for a fraction of the time. In this work a Self-Attention Policy Network for Cloud Workflow Scheduling (SPN-CWS) is used to process global information across all candidate VMs simultaneously, enabling more informed task scheduling decisions. To train SPN-CWS effectively, an Evolution Strategy-based Reinforcement Learning (ERL) system was developed is improve the self-attention mechanisms with evolutionary reinforcement learning.

The trade-off between rental fees in a distributed environment while also ensuring operational commitments are met make the DWS problem extremely difficult to solve. Having multiple objections makes reinforcement learning a popular and effective approach for DWS problems. Latency and data transfer considerations have limited research specifically on the utilization of VM's directly on the DWS problem. Existing works [15] that consider a geo-distributed setup focus on strategies for containerized applications instead of the granular VM instances directly. In this study AWS Fargate's scalability model is explored and used to makes it possible to scale applications dynamically based on demand without any need for human intervention. This presents a novel algorithm to scale containers efficiently while optimizing performance metrics for costs similar to the DWS problem for VM's, it is uniquely different due to the difference in infrastructure used. VMs are coarser-grained and require full operating system (OS) provisioning, making them suitable for workloads where resource isolation is critical. Notably, the Containers used in the study are finer-grained and share the host kernel, making them better for agile scaling and transient workloads. While effective for its scalability and responsiveness of microservice use-cases while considering parameters such as latency and throughput, existing geo-distributed research is not applicable for solving the DWS problem. This is due to its dynamic computation-intensive workload setup that is better suited for a custom VM-based solution that considers SLA guarantees.

Jayanetti et al. [42] propose a deep reinforcement learning (DRL)-based approach for cost-aware workflow scheduling in cloud computing environments. The paper focuses on optimizing task execution costs while ensuring deadline adherence for cloud-based workflows. Unlike traditional heuristic or rule-based scheduling, the proposed method leverages DRL to dynamically learn and adapt to changing cloud environments.

The study introduces an end-to-end deep learning model trained to balance cost and performance trade-offs. The DRL agent makes scheduling decisions by analyzing historical execution data and real-time system states, selecting optimal virtual machines (VMs) to minimize both execution time and cost. The model incorporates multi-objective optimization, ensuring that Service-Level Agreements (SLAs) are met while reducing expenses associated with cloud resource usage.

Experimental results demonstrate that the DRL-based scheduler outperforms traditional heuristic and static scheduling approaches, achieving significant cost savings while maintaining computational efficiency. The findings suggest that deep learning-based scheduling strategies can effectively adapt to real-world cloud workload variations, making them a viable alternative to conventional scheduling algorithms.

Table 2.5: Summary of Reinforcement Learning-Based Workflow Scheduling Methods

Paper	**Method Type**	**Objective(s)**	**Geographically Distributed**	**Adaptivity**	**Cost Awareness**	**Key Contributions**
Shen et al. [11]	SPN-CWS	SLA, Cost	No	Dynamic	Yes	Cost-aware workflow scheduling using self-attention and evolutionary reinforcement learning.
Shi et al. (2023) [15]	Auto-Scaling	Latency, Throughput	Yes	Dynamic	Yes	Proposes an RL-based auto-scaling approach for containerized applications in geo-distributed clouds. Optimizes resource allocation dynamically to minimize latency and maximize throughput, leveraging a scalable, distributed reinforcement learning model.
Jayanetti et al. [42]	Deep RL	Cost, Performance	No	Dynamic	Yes	Cost-optimized dynamic workflow scheduling using deep reinforcement learning.
Huang et al. [8]	Evolutionary RL	Multi-Workflow, SLA, Cost	No	Dynamic	Yes	Multi-workflow scheduling with cost awareness and SLA optimization.

Continued on next page

Paper	Method Type	Objective(s)	Geographically Distributed	Adaptivity	Cost Awareness	Key Contributions
Liu et al. (2020) [39]	Reinforcement Learning (Actor-Critic)	Minimize Makespan	No	Static	No	Proposes an Actor-Critic RL framework for solving static Job Shop Scheduling Problems (JSSP), demonstrating superior scheduling efficiency over heuristic-based methods.
Salimans et al. [10]	Evolution Strategies	Scalable policy optimization	No	Dynamic	No	Demonstrates that evolution strategies (ES) can be a competitive alternative to RL, offering better scalability and parallelization capabilities.
Bader et al. [38]	Gradient Bandits and Q-Learning	Minimize Resource Wastage	No	Static	No	RL-based resource allocation framework for scientific workflows, implemented within Nextflow and evaluated in a static setting.
Kintsakis et al. [43]	Reinforcement Learning (RL)	Execution Time, Resource Utilization	No	Dynamic	No	Introduces an RL-based scheduler within a Workflow Management System that autonomously learns optimal scheduling policies, enhancing adaptability and efficiency over traditional heuristic methods.

Summary of Static Workflow Methods

Static workflow scheduling methods assume complete knowledge of tasks, dependencies, and resource availability before execution, allowing for precomputed, optimized schedules. The five primary categories of static scheduling algorithms include Local Search, Mathematical Methods, Heuristics based, Genetic Programming/Genetic Algorithms and Reinforcement Learning. These approaches provide robust and efficient scheduling solutions in controlled environments where workflow arrivals and system conditions remain constant.

Local Search Methods refine task assignments iteratively to improve makespan or resource utilization, but their performance depends heavily on initial conditions and problem-specific tuning. Mathematical Methods provide precise solutions under well-defined constraints, but they struggle with scalability in large cloud environments. Heuristics-Based Approaches use fixed-rule policies designed by experts to optimize scheduling objectives such as cost, energy efficiency, or deadline adherence. While computationally efficient, these methods lack adaptability when workflow characteristics or resource availability deviate from predefined assumptions. GPHH evolves scheduling heuristics through Genetic Programming, improving flexibility over fixed heuristics but remaining inherently precomputed and non-reactive to real-time variations. While static scheduling methods excel in predictable environments, they are not well suited for real-world cloud computing scenarios, where task arrivals, execution times, and resource availability fluctuate dynamically. Precomputed schedules cannot respond to sudden workload spikes, resource failures, or unexpected changes in system performance, leading to inefficiencies and potential SLA violations.

Given these limitations, the study of dynamic workflow scheduling has gained prominence. Unlike static approaches, dynamic scheduling adapts in real-time by continuously re-evaluating scheduling decisions based on current system conditions. This adaptability is crucial for modern cloud infrastructures, where workflows arrive unpredictably, and resources must be provisioned on demand to balance cost, performance, and SLA constraints.

To overcome these limitations, dynamic workflow scheduling has emerged as a crucial area of research. Unlike static methods, dynamic scheduling continuously re-evaluates scheduling decisions in real-time, adapting to changing cloud conditions. The next section explores dynamic workflow scheduling, detailing methods that address the shortcomings of static scheduling by leveraging real-time decision-making techniques such as heuristics, Reinforcement Learning (RL), and adaptive evolutionary methods.

Summary of Dynamic Workflow Methods

For dynamic approaches in cloud there are 4 main categories of algorithms. Local Search, heuristics-based, GP/GA and RL. Mathematical methods in particular are not well suited due to the lack of adaptability in proposed policies to handle the varied workflow tasks that arrive.

The reviewed works collectively highlight various effective methods for efficient scheduling, resource management, and SLA adherence for workflow scheduling optimization problems. Due to the nature of combination problems in workflow scheduling, certain approaches are better suited to use over others. Workflow scheduling problems are NP-hard due to increases in size causing exponential increases in complexity. This has greatly increased popularity to use AI approaches to evolve solutions for. Traditional problem optimization has also found huge amounts of success using heuristic algorithms. Our research question which focuses on evolutionary reinforcement learning for dynamic VM-based scheduling, aligns closely with DWS objectives, addressing the challenges of task inter-dependencies, latency, and cost efficiency in geo-distributed systems. It is therefore realistic and thus beneficial to consider geographic limitations with existing research on DWS cost optimization in order for the approach to be adopted. The optimization of execution time, cost, availability, and quality of service in the form of SLA's will be used as the fitness evaluation of the framework.

Furthermore, this research can be extended to enhance enterprise cloud scenarios which include cloud-based AI services, and latency-critical applications like IoT and media streaming. By addressing cost, latency, and SLA adherence challenges in geo-distributed cloud environments, this work provides a foundation for scalable, efficient, and real-world cloud scheduling solutions in an industry that has demonstrated a rapid growth in this area over the last decade. However one of the biggest limitations with existing works is emulating real life reliability of cloud compute and acquiring a complete set of constraints that can be modeled effectively for reinforcement learning techniques.

While RL and GP have shown strong performance in dynamic workflow scheduling, several challenges remain:

- High Training Costs – Both RL and GP require extensive computational resources to train scheduling policies, especially when considering large-scale, geo-distributed workflows.

- Reliability and Generalization – RL models can overfit to specific workload patterns, making it difficult to generalize across diverse cloud environments.
- Hybrid Approaches – A promising direction is combining RL with GP-based hyper-heuristics, leveraging GP's ability to generate heuristics and RL's adaptability to real-time conditions.

Dynamic workflow scheduling remains a complex and open problem in cloud computing research. Existing heuristic and GP-based methods offer effective offline solutions but struggle with real-time adaptation. Meanwhile, RL approaches excel at real-time learning but face challenges in training efficiency. Future work should explore hybrid GP-RL models to leverage the strengths of both techniques while mitigating their individual weaknesses.

This research aligns closely with existing efforts in cost-aware and multi-objective dynamic scheduling while introducing novel solutions for handling geo-distributed cloud infrastructures. The next section will build upon these insights to explore practical implementations of evolutionary reinforcement learning for workflow scheduling in geo-distributed environments.

Table 2.6: Comparison of Static and Dynamic use cases in Workflow Scheduling

Approach	**Static Scheduling**	**Dynamic Scheduling**
Heuristic	Commonly used - relies on predefined rules or heuristics (e.g., HEFT, Min-Min). Well-suited for predictable workloads.	Possible but less effective when adaptability is required. Rule-based methods struggle with unforeseen changes.
Mathematical Methods	Highly applicable - commonly used for static problems (e.g., LP, ILP, CP).	Limited applicability - can be adapted for real-time re-optimization but lacks flexibility for highly dynamic changes.
Local Search	Applicable - works well for static problems (e.g., Greedy algorithms, Tabu Search, Simulated Annealing).	Applicable - extended for dynamic settings (e.g., Adaptive Simulated Annealing).
Genetic Programming (GP)	Can evolve static heuristics offline.	Effective - adapts scheduling policies dynamically.
Reinforcement Learning (RL)	Rare - pre-trained policies may be applied.	Very effective - learns and adapts to dynamic environments.

Chapter 3

Proposed Solution

This chapter outlines the proposed solution for optimizing workflow scheduling in a geo-distributed cloud environment. Due to the prohibitive cost of renting multiple real-world virtual machines (VMs) for continuous workflow evaluation, a simulation-based approach is employed. The proposed solution introduces several enhancements to an existing scheduling framework to account for regional variations in cost, latency, and data transfer overhead.

To address the challenges of distributed dynamic multi-workflow scheduling (DDMWS), this solution incorporates new functionality, including region-specific VM pricing, inter-region data transfer costs, communication delay penalties and incorrect region execution penalties. These factors directly impact scheduling decisions, necessitating modifications to the underlying reinforcement learning (RL) algorithm used for scheduling optimization. By integrating four new region aware state parameters, the enhanced RL model can make more informed decisions, balancing cost efficiency with performance constraints.

Key design components include a cost-aware scheduling mechanism that dynamically adjusts task placement based on regional constraints, an updated reward function that penalizes excessive communication delays.

By introducing these enhancements, the proposed solution aims to improve workflow execution efficiency while minimizing cost and communication overhead for the more complicated geo-distribiuted workflows. The following sections provide a detailed breakdown of the proposed technical implementation and its expected impact on the policy scheduling performance.

3.1 Training System

Following on the implementation in [11], an ERL system is used to evolve scheduling policies to minimize the total cost of VM rental fees. The scheduling policy π is vital in determining the VM used to execute task t based on the state information of the current DWS environment.

Policy Network

In order to account for the new cost calculations, the Self-Attention Policy Network for Cloud Workflow Scheduling (SPN-CWS) will need to be retrained to factor in new parameters as part of the state information used as input. Thus for DDMWS, we extend the policy network to incorporate region-aware state information to enhance its decision making capabilities in geo-distributed cloud systems.

The scheduling policy π is trained using Evolution Strategies (ES), an evolutionary reinforcement learning algorithm inspired by OpenAI's ES framework. The training process aims to minimize total scheduling costs by optimizing the assignment of tasks to VMs while considering the newly added inter-region execution constraints in the workflow simulator.

Unlike traditional RL approaches, where an agent directly interacts with an environment to update its parameters, ES-based training optimizes the policy using a population-based approach. Each generation consists of multiple policy perturbations, and their performance is evaluated based on a fitness function that we modify to incorporate region-aware penalties. The pseudo-code of the algorithm used for training is presented in Algorithm 1. The additions to our new ERL is as follows:

Geo-Distributed Optimization in ERL

To account for geo-distributed constraints, we introduce the following extensions to the standard ES framework:

1. Region-Aware Task Scheduling, the policy learns to:
   - Prioritize intra-region execution to minimize latency penalties and data transfer costs.
   - Allow cross-region scheduling when beneficial, trading off higher latency for resource availability or lower VM costs.
2. Updated Fitness Function with Region Mismatch Penalty (eg: 3.4)

   The fitness function $F(\theta_i)$ for each policy perturbation is defined as:

$$F(\theta_i) = -\Big(TotalVMFee(\pi, T) + \sum_{w \in W(T)} SLA_{\text{Penalty}}(w, \pi) + \sum_{(t,t') \in E} \text{RegionMismatch}_{\text{Penalty}}(t, t', \pi) \Big) \tag{3.1}$$

   where:

   - $W(T)$: The set of workflows containing tasks in T.
   - $(t, t') \in E$: A pair of tasks where t is a predecessor of t'.
3. Policy Gradient Update: The optimization step updates the policy parameters based on the estimated gradient:

$$\nabla_{\hat{\theta}} \mathbb{E}_{\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)} \Big[F(\theta_i) \Big] \tag{3.2}$$

   leading to an update:

$$\hat{\theta} \leftarrow \hat{\theta} + \alpha \nabla_{\hat{\theta}} \mathbb{E}_{\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)} \Big[F(\theta_i) \Big] \tag{3.3}$$

   where α is the learning rate, and σ^2 is the Gaussian noise variance.

These extensions extend the scheduling policy π effectively to adapt to the geo-distributed cloud infrastructure, reducing overall costs when dealing with region distant resource scheduling while maintaining flexibility in resource allocation.

Algorithm 1 ERL for training geo-distributed SPN-CWS (π_{sc})

```
Require: Population size: $N$, max number of generations: $Gen$, initial parameters of $\pi_{sc}$: $\hat{\theta}$,
    initial learning rate: $\alpha$, and the Gaussian standard noise deviation: $\sigma$
Ensure: Scheduling policy: $\pi_{sc}$ (the trained SPN-CWS)
 1: while the current number of generations $\leq Gen$ do
 2:     Randomly generate a DDMWS training problem from the simulator: $Pro$
 3:     for each individual ($i = 1, 2, ...$) in $N$ do
 4:         Sample $\epsilon_i \sim \mathcal{N}(0, I)$
 5:         Compute parameters of $\pi_i$: $\theta_i = \hat{\theta} + \sigma\epsilon_i$
 6:         Evaluate fitness value $F(\theta_i)$ using Eq. (3.1) based on $Pro$
 7:     end for
 8:     Estimate the policy gradient: $\nabla_{\hat{\theta}}\mathbb{E}_{\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)}F(\theta_i)$ using Eq. (3.2)
 9:     Update parameters of $\pi_{sc}$:
            $\hat{\theta} \leftarrow \hat{\theta} + \alpha\nabla_{\hat{\theta}}\mathbb{E}_{\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)}F(\theta_i)$
10: end while
```

State Representation

The state representation for the region-aware policy network is composed of two main components Task-Info and VM-Info which Captures task-specific information relevant to scheduling decisions and represents the set of available VMs and their characteristics respectively.

Four new state parameters which include the three Task-Info related states 'Task Region ID', 'Latitude', 'Longitude' and one VM-Info related state 'VM Region ID' are proposed for the algorithm to consider. 'Latitude' and 'Longitude' represent the geographical coordinates of the task's region of origin and both region ID's are categorical values that map to the name of the region where the tasks originate. The state representation that is used by the new SPN-CWS policy is summarized in table 3.1. These modifications will enable the policy to consider spatial characteristics, such as inter-region communication delay and data transfer costs, alongside traditional scheduling factors like VM availability and execution time. Providing spatial data is recommended over just the categorical data found in region ids, as it provides more raw data for the neural network to find patterns in. We anticipate having these additional region-based features known at a global level will help optimize the new reward function 3.2 and also allow the policy network to learn spatial correlations between task and VM placement, reducing latency penalties and optimizing inter-region task execution. Notably these metrics are not considered in the vast majority of existing studies.

Policy Network Architecture

The region-aware policy network builds on a self-attention mechanism, similar to the approach proposed in SPN-CWS, with modifications for handling geo-distributed scheduling. The network takes as input:

- Task embedding: Encodes task features using a feedforward layer. Now includes spatial features for latitude & longitude and the task region ID.
- VM embedding: Encodes VM features similarly. Now includes the VM region ID.

Largely inspired by SPN-CWS, the new policy network employs a transformer-based encoding method that uses the self-attention mechanism to model dependencies between tasks and VMs. Specifically:

- Multi-Head Self-Attention (MHSA): which captures relationships between tasks & VMs, identifying best-fit scheduling options.

- Position-wise Feedforward Network (FFN): to processes attention outputs to refine feature representations.

The final output is a softmax layer that selects the best VM for scheduling based on Cost minimization, SLA compliance and Latency awareness.

Table 3.1: All Task information (*task-info*) and VM information (*VM-info*) used as inputs in SPN-CWS. New states proposed in DDMWS are marked in bold.

Category	**Attribute**	**Description**
task-info	Number of Successors	The number of tasks that depend on rt in w_{rt}, i.e., $\|Suc(w_{rt}, rt)\|$
	Completion ratio	The workflow completion ratio of w_{rt} through $Com_{tasks}/Total_{tasks}$, where Com_{tasks} is the number of completed tasks in w_{rt} and $Total_{tasks}$ is the total number of tasks in w_{rt}.
	Arrival rate	Estimated orkflow arrival rate for future workflows according to the current execution situation of w_{rt} [8].
	Task Region ID	Represents the ID of the region where the task originates from or is to be executed.
	Latitude	The geographical latitude coordinate of the region in the real world.
	Longitude	The geographical longitude coordinate of the region in the real world.
VM-info	Task deadline	Indicates whether using this VM can meet the task deadline of rt. Depending on the task size, the workflow deadline is assigned to each task. Large size tasks will be assigned a larger task deadline.
	Incurred cost	Indicates the corresponding VM rental fee and Task deadline violation penalty to be incurred upon using the VM to execute rt.
	Remaining time	Indicates the remaining VM rental time after executing rt on this VM.
	Fittest VM	Indicates whether the VM being considered for executing rt has the lowest *Incurred cost* among all candidate VMs while meeting the *Task deadline*.
	VM Region ID	Represents the ID of the region in which this virtual machine (VM) was provisioned in.

3.2 Simulator System

Due to the high cost implications of renting multiple real world VM's for continuous workflow evaluation, most methods make use of simulation based studies using real world datasets.

In DDMWS we introduce new possible regions that can be used when provisioning a new VM. The introduction of a geo-distributed system into the multi-workflow scheduling problem necessitates several enhancements to the existing architecture. The primary objective is to incorporate regional variations and constraints into the scheduling algorithm while maintaining efficient resource utilization, minimizing costs, and ensuring acceptable performance levels. This document outlines the technical solution design for implementing the required changes.

New Functionality

To accommodate the distributed nature of the system, the following additions and parameters have been implemented:

Regional VM Fees

Each region has unique costs associated with provisioning virtual machines (VMs) this includes the base cost to rent a VM in a particular region for an hour (see table: 5.1).

Data Size

Each task now has a defined physical file size that represents how much space it occupies in a computers storage. $DataSize(t)$ is a new parameter that estimates a tasks real world data size in bits. It is based on a tasks processing time Size(t) scaled by some order of magnitude we define in implementation step 4. This assumes that tasks that have a larger processing time will generally be larger in size of bits.

Data Transfer Costs

Data transfer fees are incurred when tasks in workflows span multiple regions. These additional costs are fixed depend on the volume of data transferred out of a given region. In order to calculate the inter-region data transfer costs, the $DataSize(t)$ of the task and the region specific costs outlined in table: 5.3 will need to be considered.

Communication Delay

The latency between regions can impact the overall workflow execution time which has been formally defined in equation 2.5. Scheduling decisions must account for these delays to meet deadlines. Table 5.3 details the new values of latency and inter region data transfer costs that will be added to the simulation evaluation.

The new $DataSize(t)$ parameter is also required to calculate communication delay as a tasks size will affect how long it takes to move between regions with varying levels of latency. Additionally, the new $latencyPenaltyFactor$ parameter is used to scale the penalty applied for communication delays. If too high, the model avoids inter-region communication even when it's necessary. If too low, the model ignores latency, leading to high communication overhead.

Extended Functionality

In distributed systems, especially those spanning multiple regions, tasks may experience significant delays due to factors like network latency and data transfer times. In order to represent these in our workflow we propose to extend our existing simulator The following sections details what was updated in our training algorithm and SPN-CWS design.

Region Mismatch Penalty

We introduce another penalty to be applied when a VM selected by the scheduling policy is tasked with executing a task outside of its designated region [44]. This penalty is designed to explicitly penalize policies that select high latency execution operations between a VM and their queued task and reward those that optimize the task scheduling across a geo-distributed workflow.

$$RegionMismatch_{Penalty} = \sum_{t \in T} (\text{Latency}(r_a, r_b)) \times \text{RegionMismatchPenaltyFactor} \tag{3.4}$$

where:

- RegionMismatchPenaltyFactor: Scaling coefficient that determines how strongly using an incorrect VM region should impact the overall penalty, defined in the implementation step 4

Updated SLA Penalty Calculation

With consideration to communication delay between the execution of tasks located in different regions, we now incorporate a latency penalty factor to scale the size of total communication delay each predecessor task incurs for each of their successors. The total latency penalty is calculated from all tasks in a workflow which is then added to the existing SLA penalty definition which was previously only comprised of the deadline penalty. See equation: 2.13

Thus we define a new constant the *latencyPenaltyFactor* that influences the reward function by penalizing high communication delays. Its adjustment can significantly impact scheduling behavior

Increasing the Latency Penalty Factor should encourages the model to prioritize minimizing communication delays, which would be better suited to situations where SLA adherence or response time is critical for applications.

- Reduced SLA penalties.
- More tasks assigned to regions with lower latency.
- Potential increase in VM costs due to stricter latency optimization.

Decreasing the Latency Penalty Factor should reduces the priority of minimizing communication delays, which would suit cost-sensitive workflows where slight SLA violations are acceptable.

- Lower overall VM costs as the model may prefer cheaper regions over those with lower latency.
- Higher SLA penalties and increased execution time.

Updated Reward Function

The reward function should penalize high latency task executions and data transfer costs if possible. If a task misses its deadline due to high communication delay 2.5, then latency that is introduced by a geo-distributed system contributes indirectly to the SLA penalty. However, not all latency leads to SLA violations as some tasks may have long communication delay added to their execution time but still finish within the deadline. Thus we penalize latency separately but include it in the overall SLA penalty calculation. This should guide the trained scheduling policy to implicitly avoid solutions with higher latency and therefore increased VM costs. Additionally, we introduce an explicit way to avoiding inter-region communication by adding another penalty *Region_Mismatch_Penalty*. Thus our new reward function is defined as:

$$R = -\left(VM\_Cost + SLA_{\text{Penalty}} + RegionMismatch_{\text{Penalty}}\right) \tag{3.5}$$

Updated Optimal Scheduling Policy

In DDMWS, we update the optimal scheduling policy π to minimize the total cost with consideration to distributed workflows included as part of the TotalVMFee, SLA_{Penalty} and now the *Region_Mismatch_Penalty* definitions formulated as:

$$\arg\min_{\pi} TotalCost(\pi) = \arg\min_{\pi} \left\{ TotalVMFee(\pi, T) + \sum_{w \in W(T)} SLA_{\text{Penalty}}(w, \pi) + \sum_{(t,t') \in E} RegionMismatch_{\text{Penalty}}(t, t', \pi) \right\} \tag{3.6}$$

where:

- $W(T)$: The set of workflows containing tasks in T.
- $(t, t') \in E$: A pair of tasks where t is a predecessor of t'.

Modelling Geo-Distributed Functionality

Before implementing the new additions to the simulator, we need to model the new cost calculations for each inter-region pair of executed tasks. In this instance, we use a small single CyberShake example with each task being assigned a *region_id* randomly to denote where it needs to be executed. This in turn determines what region new VM's are created in and what existing VM's can execute a task when idle. Figure 3.1) is a visual representation of the newly distributed CyberShake model. It consists of 30 nodes that represent each task and 52 edges that connect each task to any successors. Each node is coloured by region and communication delay and data transfer costs are only considered when a predecessor node has a successor that is located in a different region than its own. For this instance with task two having five successor tasks, communication delay (equation: 2.5) will be applied to all five successor tasks as they are all located outside task id two's region. Additionally, data transfer costs (equation: 2.6) will be added to the current VM being leased five times for the same reason. And so on and so forth until all tasks are marked as complete.

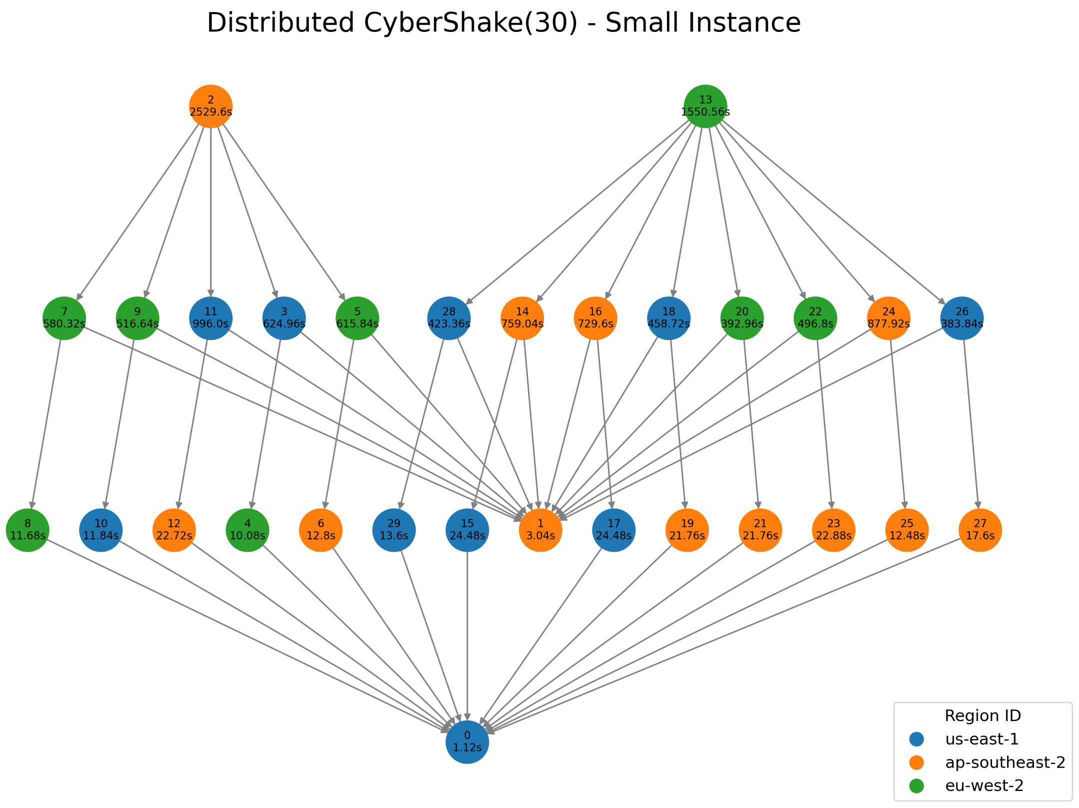

Figure 3.1: The new Directed Acrylic Graph (DAG) of one of the simulated geo-distributed workflows *CyberShake*. Each node is represented with a *task_id* that also contains the *Size*(t) 2.7 required to complete each task. In DDMWS each node is also assigned a *region_id* which denotes its origin.

Chapter 4

Implementation

In this section we detail the implementation of the new policy network, training system and cloud workflow simulator used.

Simulator

The newly developed simulator includes substantial additions in order to support the processing of geographically distributed workflows. The simulator utilized in previous work [11] contained a dataset consisting of only one set of VM pricing information (us-east-1) to train the policy network. A key addition from this paper is the curation of a new easily extendable dataset class for multiple VM region types. The following attributes were created to handle new constants required to process communication delay 2.5 and data transfer costs 2.6 in a geo-distributed system. Refer to appendix: A.1 for all the new additions added to the simulator code.

Regional VM Base Fees

VM's are priced by their speed and storage limitations with a general rule of price for rental per hour being doubled alongside an equivalent increase in vCPU and Memory. Table 5.1 shows prices for the AWS 'm5' family of virtual machines across different regions which follow this cost pattern, for example a 'large' VM with 2 vCPU's costs half of that of an 'xlarge' instance with 4 vCPU's.

To account for the cost of distributed variations of VM's, we define a set of dictionaries *region_map* and *vm_basefee*. The former links an integer *region_id* with a standardized AWS region name and the latter links *region_id* with the base cost fee of a VM based on the region it belongs to. We define the base cost of a VM as its price per hour based on one vCPU of computation. However as the minimum provided by AWS is two vCPUs we derive this by dividing their cheapest two vCPU option by two.

Bandwidth

Bandwidth is based directly from the type of vCPU provisioned for VM's as taken from AWS: 5.1. Bandwidth capacity for each vCPU VM Type is measured in Gigbits per second (Gbps) and stored as a dictionary *bandwidth_map*. For added diversity in bandwidth values, we represent VM's with 2, 4 and 8 vCPU's as having a standardized bandwidth of 8 Gbps to align with the documentation reference of being "Up to 10 Gbps".

Latency

Latency is a key component used when calculating inter-region communication delays. A nested dictionary $latency\_map$ is used to store the latency in milliseconds (ms) between each source $region\_id$ and all possible target $region\_id$'s.

Data Transfer

Inter-region data transfer costs are additional costs charged out per GB of outgoing data transferred to a VM that is not located in the same region as the source VM. This is simply defined as a dictionary $data\_transfer\_cost\_map$ with each key referencing a $region\_id$.

Scaling Factors

We introduce three new scaling constants that are required for calculating the values described in our problem definition and proposed solution which our new geo-distributed simulator can now handle:

- $dataScalingFactor$
- $latencyPenaltyFactor$
- $regionMismatchPenaltyFactor$

Communication Delay and Data Transfer are two calculations that use the new *dataScalingFactor* to handle the scaling of *dataSize* value. The *ProcessTime* of a task is simply multiplied by the *dataScalingFactor* which is value between 0 and 1.0. When a task is queued, each of their successor tasks are now evaluated for their region location and *datasize*. For each successor task region located outside of the predecessor region, a data transfer cost and communication delay is added to total VM cost and task enqueue time respectively (see equation 2.5 and 2.6).

The updated SLA Penalty 2.14 uses the $latencyPenaltyFactor$ to scale workflows with a higher total incurred *Communication_Delay* 2.5. This scaling factor is also set between a value of 0 and 1.0.

Lastly, the new Region Mismatch Penalty 3.4 uses the $regionMismatchPenaltyFactor$ to scale the pre-determined latency between two regions as determined by the $latency\_map$ defined earlier. Again this is also set between a value of 0 and 1.0.

Assigning new region and data size to tasks

The datasets of workflows used for training and evaluation: 'CyberShake', 'Inspiral', 'Montage' and 'Sipht' currently do not contain any information regarding location. Due to this, the simulator used in recent work [11] assumes every task to be carried out by VM's located in the the us-east-1 region.

Our newly developed simulator assigns a new attribute $regionId$ to each node in the DAG representation randomly. Each node is also assigned a *dataSize* parameter that uses the method described previously in the *Scaling Factors* section 4 to estimate a tasks physical size based on its $processTime$.

We have decided not to modify the original dataset file directly but extend it after it is initially loaded into memory to allow for ease of use when experimenting with variables such as the $dataScalingFactor$.

Guiding the VM Selection Rule for geo-distributed tasks

One key design we had to consider when creating our geo-distributed simulator is that because regions are pre-assigned to each task in a workflow we had to decide on how this would interact with the policy selected VM. We chose to limit only newly created VMs to be provisioned in the same region as the next task in the queue. Thus, the scheduling policy would need to pick the best existing VM or create a new one in that region. This simplifies the scheduling as the VM is now guaranteed on the correct region initially however it can prevent the model from learning an optimal placement strategy. The optimal placement strategy should involve the consideration of selecting an existing cheaper VM in terms of rental costs and low enough communication delay and data transfer costs that it would be cheaper overall. While this does add complexity compared to previous work we believe a hybrid approach involving the pre-determined task regions and dynamic region selection in VMs could improve this. Driven by the new $RegionMismatchPenalty$ 3.4, the total reward will be higher the more time the VM chosen to execute a task belongs outside of the tasks region of origin.

In DDMWS the SLAPenalty via the latencyPenaltyFactor will be higher if more successor tasks need to be computed via inter-region data transfer from predecessors. This introduces a new trade-off optimization between using a VM region that does not match a predecessor task region but does keep it local for successors vs just taking the region-mismatch penalty and just having more communication delay.

This is the basis of our hybrid implementation for VM selection options as we allow the policy to also choose a VM in a different region if there is some benefit from the trade-off optimization (eg. choosing an expensive VM in a different region and incurring all the costs is faster and also cheaper than using an existing slower one in the same region (see table: 5.1). We want the new policy scheduler to consider if running the task in another region is cheaper (e.g., lower VM costs) and allow it to do that.

A typical scenario in the new DDMWS where a hybrid approach occurs is as follows:

- Simulator gets first workflow and uses the trained policy to make an action when presented with all the state information available. As it is the task from the first workflow, a new VM will be created. From the new additions in DDMWS, when a new VM is created it will be assigned a region (and therefore use that regions cost model) that matches the region of next queued task.
- When the Next task is ready, use the policy to make a decision again
- If action is to use an existing VM, the VM region is checked if it matches the region of the next queued task, if it is not, a region mismatch penalty is applied. The total of this for all actions are added to the reward policy.
- For each predecessor/successor pair, calculate communication delay and data transfer costs as usual and generate the total communication delay for all successor tasks where valid. Then add this to the SLAPenalty. Add the calculated data transfer costs to the total rental costs.

Impact of Geo-Distributed Optimization

By incorporating the region-aware scheduling constraints into ES, our policy:

- **Learns to balance execution efficiency and inter-region costs**, choosing optimal VM placements dynamically.

- **Adapts region selection based on observed latencies**, penalizing inefficient placements.

- **Minimizes inter-region penalties without restricting scheduling flexibility**, enabling better geo-distributed cost-aware scheduling placement. Tasks are assigned to regions based on cost optimization strategies, prioritizing regions with lower VM fees and minimal data transfer requirements.

Chapter 5

Experiment Results

5.1 Simulation Configuration

In this section, we present the DDMWS simulation environment for training and evaluating SPN-CWS using ERL for geo-distributed job scheduling. The simulation framework models a cloud computing environment with geographically distributed VM's, incorporating real-world network latency and data transfer constraints.

Regional VM Configuration and Workload Patterns:

Our environment consists of multiple geo-distributed VM's available for provisioning with heterogeneous computational resources. These VM's vary in processing power, memory, and network bandwidth, reflecting real-world cloud infrastructure. In line with existing studies [8], [11] we use six types of VM's with their respective settings summarized in Table 5.1.

Table 5.1: EC2 VM Cost per hour and Bandwidth limits across regions [14] [45]

VM Name	vCPU	Memory (GB)	Network Bandwidth (Gbps)	US East (N. Virginia)	Asia Pacific (Sydney)	Europe (London)
m5.large	2	8	Up to 10	$0.096	$0.12	$0.111
m5.xlarge	4	16	Up to 10	$0.192	$0.24	$0.222
m5.2xlarge	8	32	Up to 10	$0.384	$0.48	$0.444
m5.4xlarge	16	64	Up to 10	$0.768	$0.96	$0.888
m5.8xlarge	32	128	10	$1.536	$1.92	$1.776
m5.12xlarge	48	192	12	$2.304	$2.88	$2.664

Each data center provides a range of virtual machine (VM) types with different configurations, including CPU, GPU, and memory-optimized instances. Following standard cloud workload benchmarks, our experiments utilize diverse job workflows representing real-world computation-intensive tasks. These workflows are categorized into three complexity levels based on the number of dependent tasks: *Small*, *Medium*, and *Large*. The complexity influences scheduling decisions, where larger workflows introduce greater challenges due to data locality constraints and inter-task communication overhead. Summarized in Table 5.2, the workflows used to experiment consist of four well known real world datasets [46]

(CyberShake, Montage, Inspiral and SIPHT) commonly used in recent studies [8], [11], [20], [34].

Table 5.2: The three workflow sets used in this study.

Workflow set	**Name of Workflow (number of tasks)**			
Small	CyberShake(30)	Montage(25)	Inspiral(30)	SIPHT(30)
Medium	CyberShake(50)	Montage(50)	Inspiral(50)	SIPHT(60)
Large	CyberShake(100)	Montage(100)	Inspiral(100)	SIPHT(100)

Network Constraints and Scheduling Challenges

Unlike traditional cloud scheduling, our framework accounts for inter-data-center network latencies and data transfer costs. Tasks scheduled across multiple regions experience varying communication delays, making region-aware scheduling a critical factor. To model realistic scenarios, network latencies are derived from real-world cloud provider data summarized in Table 5.3, ensuring that the ERL system for the neural network training is able to learn effective scheduling policies under practical geographical constraints.

Table 5.3: EC2 Data Transfer Costs and Latency [47] [16]

Source Region	**Within Region ($ per GB)**	**Inter-Region ($ per GB)**	**Latency to Asia Pacific (ms)**	**Latency to Europe (ms)**
US East (N. Virginia)	0.01	0.02	197	74
Asia Pacific (Sydney)	0.01	0.098	–	264
Europe (London)	0.01	0.02	264	–

DDMWS problem instance & scenarios

The problem instance of DDMWS follows the study in [11], with each problem instance containing a total of 30 workflows generated randomly from one using a Poisson distribution with $\lambda = 0.1$ to simulate works being submitted dynamically by users over a given time. We set the SLA dealine coefficient in equation 2.14 to $\gamma \in (1.00, 1.50, 2.00)$ in order to test scenarios with different deadline requirements. The deadlines become more relaxed as γ increases, which give more of an incentive for the broker to rent cheaper and slower VM's to execute incoming workflows.

Each problem scenario is categorized by size: small DDMWS scenarios are generated from small workflows, medium scenarios from medium workflows and large scenarios from large workflows respectively. As we are evaluating the SPN-CWS performance, we also use 30 small scenarios, 30 medium-scenarios, and 30 large-scenarios of DDMWS to evaluate performance based on the average of the 30 DDMWS problem instances per size. Due to time constraints the medium and large scenario DDMWS problems instances are not used during training and only used to assess the generalization capability of the new region aware SPN-CWS.

Baseline Comparisons

To evaluate our approach, we compare against existing SPN-CWS algorithm trained as it was in its original paper against SPN-CWS trained with our newly developed ERL for geo-distributed systems. Specifically, the baseline algorithm will be trained with its original network architecture but evaluated against a distributed problem scenario. Once the two models are recorded, we then perform hyper-parameter adjustments to the $latencyPenaltyFactor$ (eq: 2.13) and $regionMismatchFactor$ (eq: 3.4) for retraining to see how the SPN-CWS model handles the trade-off between total VM rental cost and the SLA penalty with consideration to the added communication delay and inter-region data transfer costs. Performance metrics include the total cost (reward) (eq:2.15), total VM rental costs, total VM execution time and total SLA penalties.

Experimentation of SPN-CWS involves evaluating its performance on its original training configuration with only one region considered and then retraining it with the four additional parameters described in table 3.1 and using the problem instances but with distributed tasks enabled. We evaluate each model on each size (S, M, L) and present comparative analysis.

Parameter Configuration

The parameter configurations for all competing SPN-CWS algorithms adhere to their respective original paper. For DDMWS, the two new penalty scaling constants 2.13 3.4 are both set to 0.5.

For SPN-CWS, the feedforward hidden layer sizes for *Global Information Learning*, *Task Feature Enhancement*, and *Priority Mapping* are configured as 64, 32, and 32, respectively, with *ReLU* as the activation function.

Each model is trained for only 500 generations due to time and resource constraints. This is notably less than the required convergence metrics available at 3000 trained generations presented in the original study [11]. However from the original work, 500 generations does provide sufficient information on the underlying performance of the model as it is shown to be a top performer respective to earlier trained generations and the convergence values.

5.2 Main Result Analysis

It is observed that across all workflow sizes, the model trained with our new DDMWS simulator outperforms the baseline model in key comparison metrics: Total cost (reward), total VM cost, execution time, SLA Penalty and Region Mismatch Penalty. As mentioned in the DDMWS problem scenario 5.1 our models are all trained using extremely relaxed deadlines $\gamma = 18$ in order to give the policy the leniency to select the correct region without tight deadlines. This is important as it allows for more flexibility in deadlines for workflows with inter-region dependencies that now will take longer than in the original study. Notably, we also trained a set of models on $\gamma = 5$ to replicate the original experiment and to test the models ability to generalize in tighter deadline scenarios. Interestingly, models trained on distributed problem scenarios with a higher deadline coefficient performed worse on average to their lower coefficient counterparts. This suggests that the added latency penalty (which directly affects if a deadline can happen via the SLA penalty) is effective enough to prevent the scheduling policy from just selecting the cheapest, slowest VM. It shows DDMWS being able to generalize well and optimize the best for relaxed workflows.

From the results summarized in table 5.4 we confirm that the baseline SPN-CWS algorithm is very effective at optimizing VM rental costs for a single region as noted in

Table 5.4: Average total cost of each algorithm trained to 500 generations over 30 independent geo-distributed runs.

Scenarios	**Baseline SPN-CWS [11]**	**DDMWS SPN-CWS $\gamma = 18$**	**DDMWS SPN-CWS $\gamma = 5$**
$\langle 1.00, S\rangle$	371.73	375.19	385.79
$\langle 1.00, M\rangle$	727.17	714.73	687.44
$\langle 1.00, L\rangle$	1217.34	1169.01	1090.56
$\langle 1.50, S\rangle$	428.04	363.21	366.83
$\langle 1.50, M\rangle$	740.48	618.53	653.38
$\langle 1.50, L\rangle$	1254.89	1169.65	1089.93
$\langle 2.00, S\rangle$	386.64	362.47	337.19
$\langle 2.00, M\rangle$	689.89	622.51	672.91
$\langle 2.00, L\rangle$	1221.60	1136.74	1066.92

its original paper [11]. However its effectiveness and ability to learn and generalize in a geo-distributed problem scenario is very lackluster, shown by its fluctuating total reward value across generations and diminished performance summarized in figure: 5.1.

Another observation is that the geo-distributed Model improves more steadily across generations. Although it starts off worse it gradually improves, showing a consistent downward trend past the 200 generation mark (better rewards). After roughly 200 episodes, the DDMWS trained model is consistently better. Conversely, the baseline model degrades after episode 200, which reinforces that providing geo-distributed state information leads to better long-term scheduling performance.

5.3 Sensitivity Analysis

The introduction of region mismatch and latency penalties (in the 0.5 and 0.55 models) significantly improved the efficiency of scheduling compared to the baseline model, which lacked distributed state information. The 0.55 model (trained with γ = 18) demonstrated a lower SLA penalty and reduced VM execution costs, suggesting that the model learned to optimize VM selection and task allocation better than both the baseline and 0.5 model. Figure 5.2 summarizes these findings.

The following graphs compare trained models with different penalties (both latency and region mismatch penalty set to 0.50 and 0.55) against a baseline model. Here we highlight several key findings from our work:

VM Total Cost Sensitivity

- The trained models (both penalties) generally reduce VM costs compared to the baseline model.

- The Penalty = 0.55 model appears to be more cost-efficient, suggesting it encourages lower VM spending. However, fluctuations in cost may indicate trade-offs with other metrics.

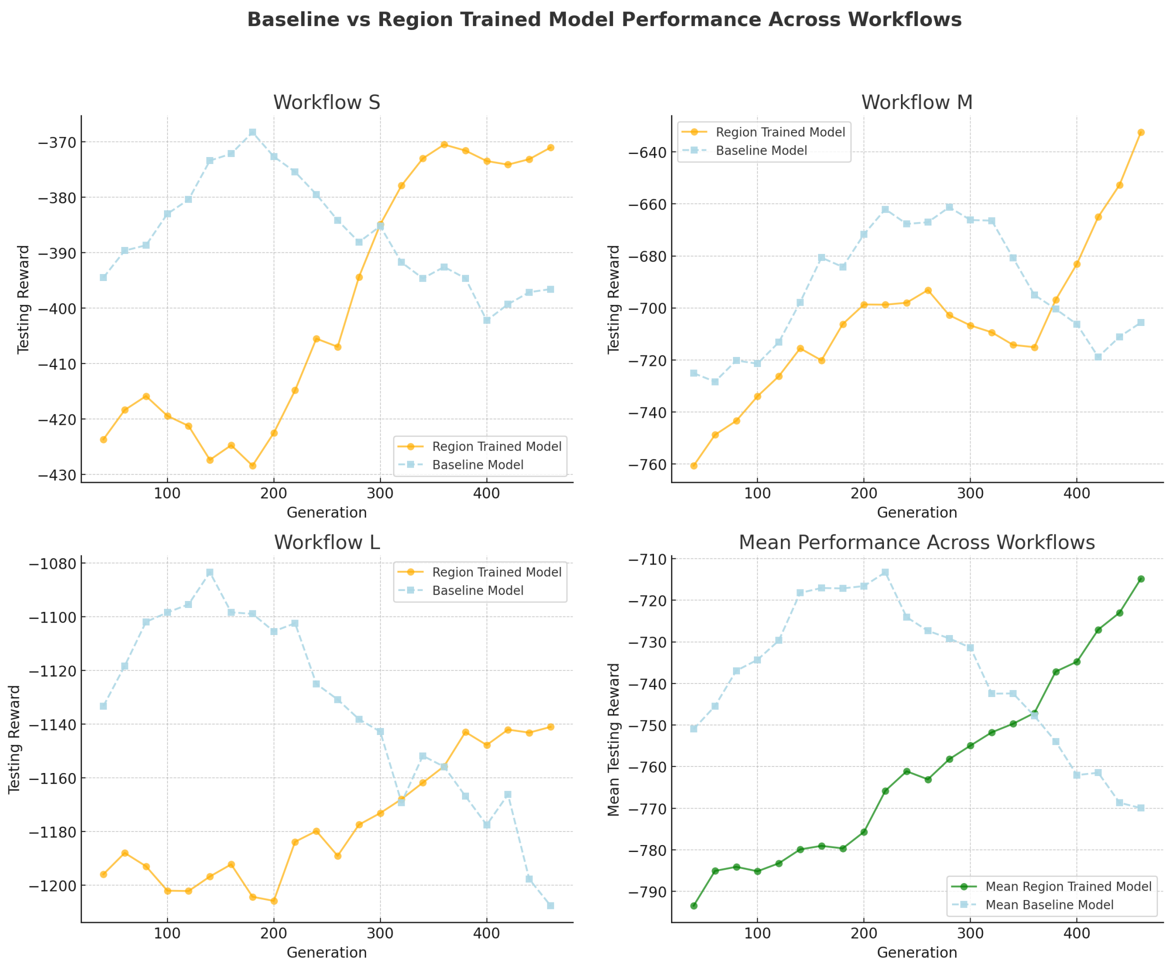

Figure 5.1: The baseline and region trained model performance evaluated against all workflow instances with $\gamma = 2$. The mean reward results are also aggregated for all instances.

SLA Penalty Sensitivity

- The trained models consistently lower SLA penalties compared to the baseline.
- The Penalty = 0.50 model seems to strike a balance between cost and SLA penalties, reducing violations while maintaining reasonable costs.
- The Penalty = 0.55 model shows some fluctuations, possibly due to stricter constraints leading to occasional higher penalties.

Region Mismatch Penalty Sensitivity

- The baseline model has higher region mismatch penalties at the earlier generations, meaning it frequently executes tasks in the wrong region. However unexpectedly it improves and excels compared to the other two region-aware trained models which suggests that it is optimizing for other objectives instead.
- The trained models significantly improve region-aware scheduling, with Penalty = 0.55 showing slightly better control. This indicates that training helps optimize task placement, reducing inefficiencies.

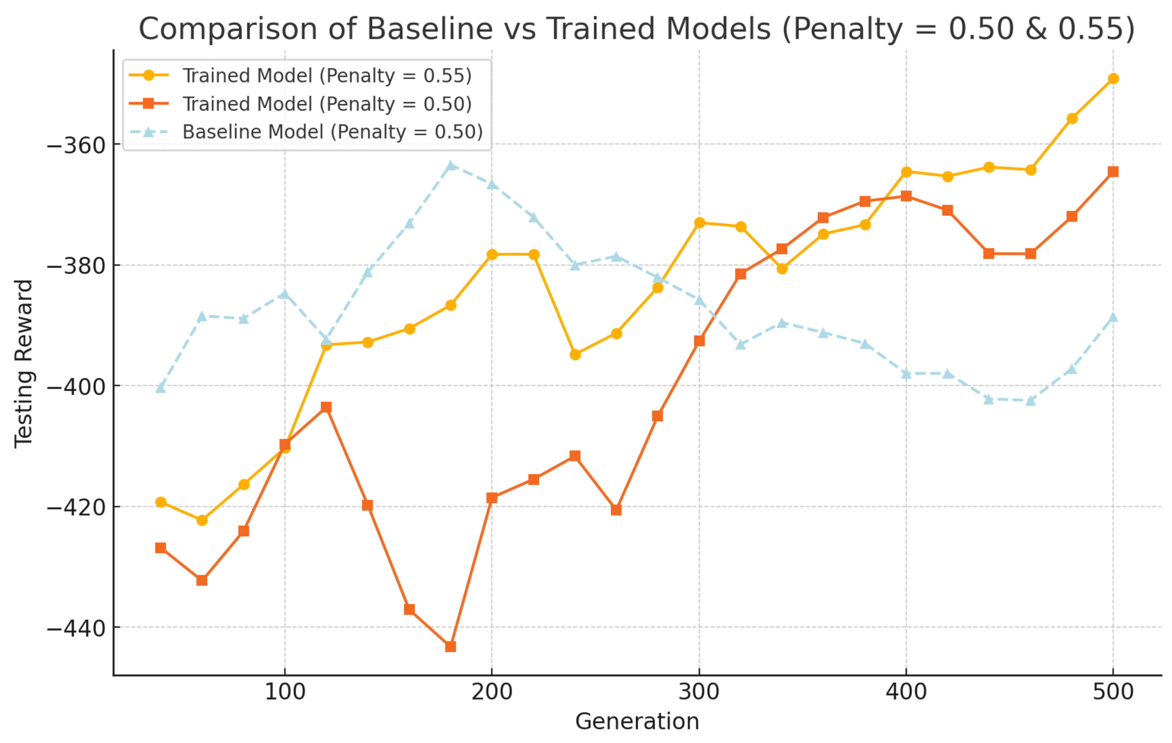

Figure 5.2: The three models trained with $\gamma = 18$ and with different latency and region mismatch penalty factors evaluated against a size “S“ workflow

VM Execution Time Sensitivity

- The trained models improve execution efficiency over the baseline, reducing overall VM execution time.
- The Penalty = 0.50 model shows more stability, while the Penalty = 0.55 model sometimes trades execution time for cost savings.
- A model with overly strict penalties could impact execution time negatively if it overly restricts scheduling flexibility.

Key Takeaways

- Penalty values impact cost vs. efficiency trade-offs: Lower penalties may allow more flexibility, while higher penalties enforce stricter scheduling but can sometimes increase costs or execution time.
- Trained models outperform the baseline across all key metrics, showing the importance of optimization.
- Finding the optimal penalty value requires balancing cost, SLA penalties, execution time, and region mismatch penalties. Currently it seems like the region mismatch penalty is too low due to the region-aware models tending to neglect its value on the total cost into the later generations.

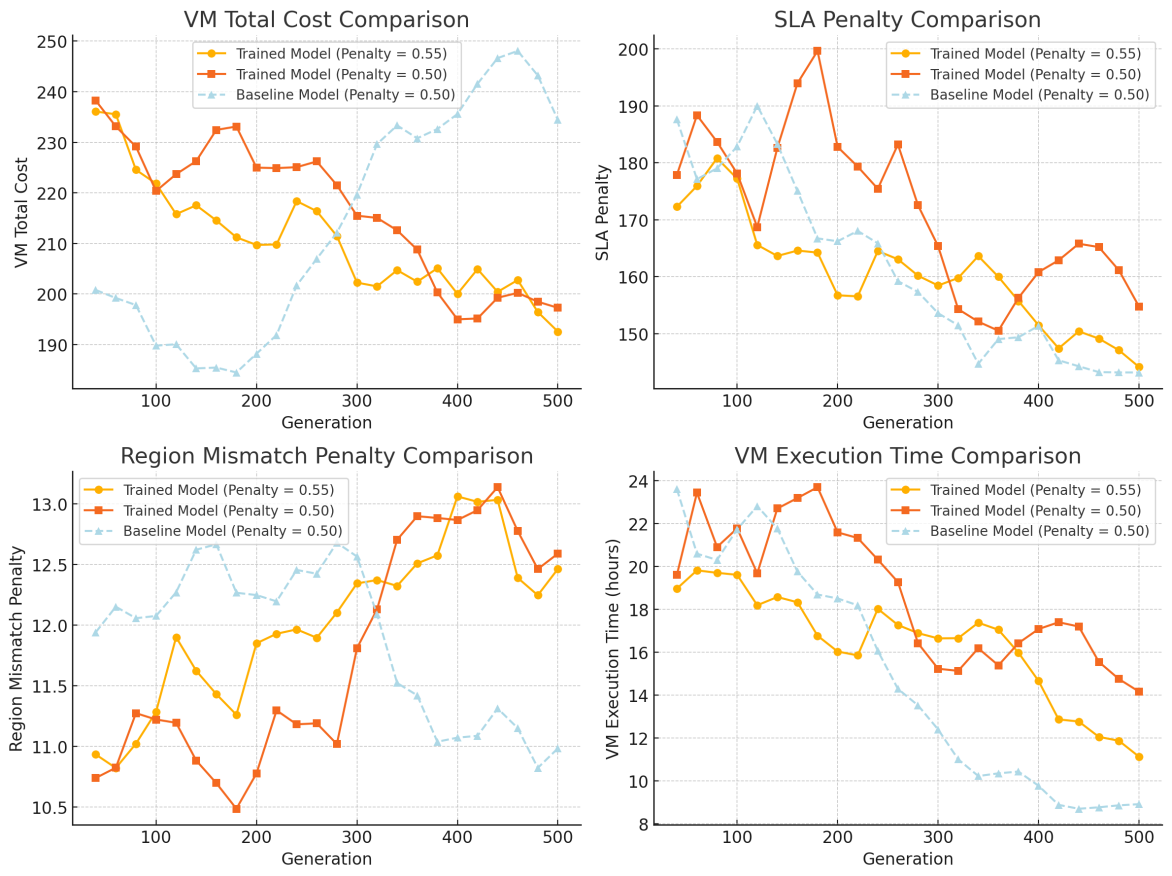

Figure 5.3: Comparison of key performance metrics between the trained models (Penalty = 0.50 & 0.55) and the baseline model on DDMWS. The graphs illustrate trends in VM Total Cost, SLA Penalty, Region Mismatch Penalty, and VM Execution Time over generations, highlighting the impact of training and penalty adjustments on scheduling performance.

Chapter 6

Conclusions

6.1 Research summary and contributions

This study investigates the impact of incorporating distributed state information into workflow scheduling for cloud environments. The primary objective is to enhance scheduling efficiency by reducing execution costs and SLA penalties while accounting for regional constraints in a distributed setting. The study evaluates the performance of different models trained with various penalty factors and SLA deadline coefficients γ to understand their influence on scheduling efficiency.

To address the complexities introduced by distributed cloud environments, we incorporated a region mismatch penalty, which penalizes tasks executed in regions different from their designated ones. The model was evaluated across multiple workflow sizes (S, M and L) and trained under different penalty weights and γ values to determine the optimal balance between cost efficiency and SLA compliance.

Our findings reveal that incorporating distributed state information significantly improves scheduling performance, particularly in complex workflow scenarios. While initial training phases exhibited instability, later generations demonstrated consistent improvements in reward values, VM cost reduction, and SLA adherence. Additionally, increasing the region mismatch penalty led to a more precise allocation of tasks, effectively reducing execution mismatches while maintaining cost efficiency.

Our main contributions can be summarized as follows:

1. **Simulator Environment:**
   - Created a simulation framework that includes geo-distributed datasets, regional constraints, and latency models.
   - Updated the workflow model to include regional VM fees, data transfer costs, and communication delay metrics.
2. **Algorithm Development:**
   - Extended the existing scheduling algorithm to factor in new parameters using a cost-delay optimization approach.
3. **Testing and Validation:**
   - Conducted extensive testing using synthetic and real-world workflow scenarios to validate cost efficiency and performance improvements compared to the baseline SPN-CWS algorithm.

6.2 Future Direction

As of writing in December 2024, Microsoft has launched New Zealand's very first dedicated data center region "New Zealand North". [48] As the need for data sovereignty and quicker response times increases, large public cloud companies have more of an incentive to build regions that extend their market share. Incorporating distributed optimization leanings is therefore beneficial for future scenarios when cloud regions open up to more of the world and further cost optimizations can be explored.

There has also been recent research [42] into the possibility of optimizing a special type of VM instance called 'spot instances' [49] [50] which involves bidding on unused computing capacity to be used sporadically but at incredibly cheap prices. Using Spot instances will always yield the cheapest possible total cost and current research is focused on minimizing its only drawback which is its dynamic interruptions.

Additionally, existing work usually only considers a very limited amount of instance types that are 'general purpose' variants. In most existing work, the default is the 'm5' family [11] [8] [33]. AWS provides a large variety of 'compute optimized' instances [45] for example the 'c5n' family which has more than twice the amount of network bandwidth capability. These machine parameters may be more suited for higher latency workloads within a multi-region distributed workflow as described earlier. This highlights the complexity in the nature of the problem as there are vast amounts of VM variations that perform a variety of tasks better than others. Incorporating this dynamically and providing all available VM information as a whole would likely yield improved overall robustness of the DWS problem.

There is also a lot of future potential work on the DDMWS simulator. VM utilization per region can be monitored to implement a region based load-balancing to aid in the scheduling policy. Due to the nature of the newly introduced penalties, the relaxation coefficient γ could be adjusted dynamically based on the magnitude of geographical distance and task dependencies. Specifically, greater relaxation coefficients for workflows containing tasks that span multiple distant regions and those with complex inter-region communication requirements. Geographical distance could also be incorporated into the state information as input for the SPN-CWS neural network which may work better than the current raw coordinate information being used in this study.

Appendix A

Appendix

This appendix contains the new static constant variables added to the codebase[1]for this project.

A.1 New Dataset variables for Geo-distributed Workflows

Listing A.1: New dictionaries implemented to support distributed calculations

```
self.region_map = {
    0: "us-east-1",        # East, USA, N.Virginia
    1: "ap-southeast-2",   # Southeast, Australia, Sydney
    2: "eu-west-2"         # West, Europe, London
}

self.vm_basefee = {
    0: 0.048,
    1: 0.06,
    2: 0.0555
}

self.bandwidth_map = {
    2: 8,
    4: 8,
    8: 8,
    16: 8,
    32: 10,
    48: 12
}

self.latency_map = {
    0: {  # From N. Virginia
        1: 197,  # Latency to Sydney
        2: 75    # Latency to London
    },
    1: {  # From Sydney
        0: 197,  # Latency to N. Virginia
        2: 264   # Latency to London
    },
    2: {  # From London
        0: 75,   # Latency to N. Virginia
        1: 264   # Latency to Sydney
    }
}

self.region_coords = {
    0: [39.0438, -77.4874],    # N. Virginia
    1: [-33.8688, 151.2093],   # Sydney
    2: [51.5074, -0.1278]      # London
}

self.data_transfer_cost_map = {
    0: 0.02,
    1: 0.098,
    2: 0.02
```

[1] https://github.com/cllaw/AIML589-RL-DDMWS

Bibliography

[1] Gartner. “Gartner forecasts worldwide public cloud end-user spending to surpass $675 billion in 2024”. Accessed: 2024-12-12, Intelligent CIO. (2024), [Online]. Available: `https://www.intelligentcio.com/apac/2024/05/30/gartner-forecasts-worldwide-public-cloud-end-user-spending-to-surpass-us675-billion-in-2024`.

[2] ITBrief. “Public cloud adoption set to surge across australia, nz”. Accessed: 2024-12-12, Intelligent CIO. (2023), [Online]. Available: `https://itbrief.co.nz/story/public-cloud-adoption-set-to-surge-across-australia-nz`.

[3] Y. Yang, G. Chen, H. Ma, and M. Zhang, “Dual-tree genetic programming for deadline-constrained dynamic workflow scheduling in cloud”, in *International Conference on Service-Oriented Computing*, Springer, 2022, pp. 433–448.

[4] M. Adhikari, T. Amgoth, and S. N. Srirama, “A survey on scheduling strategies for workflows in cloud environment and emerging trends”, *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–36, 2019.

[5] R. News. “A close call: Why trade me chose google’s cloud over aws”. Accessed: 2024-12-12, Reseller News. (2024), [Online]. Available: `https://www.reseller.co.nz/article/1296197/a-close-call-why-trade-me-chose-googles-cloud-over-aws.html`.

[6] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need”, *Advances in neural information processing systems*, vol. 30, 2017.

[7] F. Wu, Q. Wu, and Y. Tan, “Workflow scheduling in cloud: A survey”, *The Journal of Supercomputing*, vol. 71, pp. 3373–3418, 2015.

[8] V. Huang, C. Wang, H. Ma, G. Chen, and K. Christopher, “Cost-aware dynamic multi-workflow scheduling in cloud data center using evolutionary reinforcement learning”, in *International Conference on Service-Oriented Computing*, Springer, 2022, pp. 449–464.

[9] I. A. Chaudhry and A. A. Khan, “A research survey: Review of flexible job shop scheduling techniques”, *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.

[10] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution strategies as a scalable alternative to reinforcement learning”, *arXiv preprint arXiv:1703.03864*, 2017.

[11] Y. Shen, G. Chen, H. Ma, and M. Zhang, “Cost-aware dynamic cloud workflow scheduling using self-attention and evolutionary reinforcement learning”, *arXiv preprint arXiv:2409.18444*, 2024.

[12] A. W. Services, *Aws service availability*, Accessed: 2025-01-18, 2025. [Online]. Available: `https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/availability.html`.

[13] F. Fakhfakh, H. H. Kacem, and A. H. Kacem, "Workflow scheduling in cloud computing: A survey", in *2014 IEEE 18th international enterprise distributed object computing conference workshops and demonstrations*, IEEE, 2014, pp. 372–378.

[14] A. W. Services, *Aws ec2 on-demand instance pricing*, Accessed: 2024-12-10, 2024. [Online]. Available: `https://aws.amazon.com/ec2/pricing/on-demand`.

[15] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Auto-scaling containerized applications in geo-distributed clouds", *IEEE Transactions on Services Computing*, 2023.

[16] A. W. Services, *Aws ec2 on-demand instance data transfer pricing*, Accessed: 2024-12-10, 2024. [Online]. Available: `https://aws.amazon.com/ec2/pricing/on-demand/#Data_Transfer`.

[17] X. Ma, H. Xu, H. Gao, and M. Bian, "Real-time multiple-workflow scheduling in cloud environments", *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4002–4018, 2021.

[18] New Relic, *Managing sla breaches: Best practices to avoid violations*, Accessed: 2024-02-11, 2024. [Online]. Available: `https://newrelic.com/blog/best-practices/managing-sla-breaches`.

[19] K. Bochenina, N. Butakov, and A. Boukhanovsky, "Static scheduling of multiple workflows with soft deadlines in non-dedicated heterogeneous environments", *Future Generation Computer Systems*, vol. 55, pp. 51–61, 2016.

[20] H. R. Faragardi, M. R. S. Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, "Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds", *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1239–1254, 2019.

[21] G. Lee and R. H. Katz, "{Heterogeneity-aware} resource allocation and scheduling in the cloud", in *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 11)*, 2011.

[22] M. Chen, M. T. Islam, M. R. Read, and R. Buyya, "Trade: Network and traffic-aware adaptive scheduling for microservices under dynamics", *arXiv preprint arXiv:2411.05323*, 2024.

[23] J. Liu, J. Ren, W. Dai, *et al.*, "Online multi-workflow scheduling under uncertain task execution time in iaas clouds", *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1180–1194, 2019.

[24] Z. Yu and W. Shi, "A planner-guided scheduling strategy for multiple workflow applications", in *2008 International Conference on Parallel Processing-Workshops*, IEEE, 2008, pp. 1–8.

[25] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget and deadline aware e-science workflow scheduling in clouds", *IEEE Transactions on Parallel and Distributed systems*, vol. 30, no. 1, pp. 29–44, 2018.

[26] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines", *Operations-Research-Spektrum*, vol. 15, pp. 205–215, 1994.

[27] L. Xing, M. Zhang, H. Li, M. Gong, J. Yang, and K. Wang, "Local search driven periodic scheduling for workflows with random task runtime in clouds", *Computers & Industrial Engineering*, vol. 168, p. 108 033, 2022.

[28] A. A. Jihad, S. T. F. Al-Janabi, and E. T. Yassen, "Enhanced iterated local search for scheduling of scientific workflows", in *2021 14th International Conference on Developments in eSystems Engineering (DeSE)*, IEEE, 2021, pp. 335–339.

[29] S. Qin, D. Pi, and Z. Shao, "Ails: A budget-constrained adaptive iterated local search for workflow scheduling in cloud environment", *Expert Systems with Applications*, vol. 198, p. 116 824, 2022.

[30] R. A. Al Na'mneh and K. A. Darabkh, "A new genetic-based algorithm for scheduling static tasks in homogeneous parallel systems", in *2013 International Conference on Robotics, Biomimetics, Intelligent Computational Systems*, IEEE, 2013, pp. 46–50.

[31] K. Luchoomun, P. Auckloo, and B. Sonah, "Enhancing performance of genetic algorithm for static job-shop scheduling problems", *International Refereed Journal of Engineering and Science (IRJES)*, vol. 3, no. 7, pp. 39–49, 2014.

[32] Y. Yang, G. Chen, H. Ma, M. Zhang, and V. Huang, "Budget and sla aware dynamic workflow scheduling in cloud computing with heterogeneous resources", in *2021 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2021, pp. 2141–2148.

[33] Y. Yang, G. Chen, H. Ma, S. Hartmann, and M. Zhang, "Dual-tree genetic programming with adaptive mutation for dynamic workflow scheduling in cloud computing", *IEEE Transactions on Evolutionary Computation*, 2024.

[34] K.-R. Escott, H. Ma, and G. Chen, "Genetic programming based hyper heuristic approach for dynamic workflow scheduling in the cloud", in *Database and Expert Systems Applications: 31st International Conference, DEXA 2020, Bratislava, Slovakia, September 14–17, 2020, Proceedings, Part II 31*, Springer, 2020, pp. 76–90.

[35] M. Xu, Y. Mei, F. Zhang, and M. Zhang, "Genetic programming for dynamic flexible job shop scheduling: Evolution with single individuals and ensembles", *IEEE Transactions on Evolutionary Computation*, 2023.

[36] M. Đurasević, L. Planinić, F. J. G. Gala, and D. Jakobović, "Novel ensemble collaboration method for dynamic scheduling problems", in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 893–901.

[37] S. Nguyen, D. Thiruvady, Y. Sun, and M. Zhang, "Genetic-based constraint programming for resource constrained job scheduling", in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2024, pp. 942–951.

[38] J. Bader, N. Zunker, S. Becker, and O. Kao, "Leveraging reinforcement learning for task resource allocation in scientific workflows", in *2022 IEEE International Conference on Big Data (Big Data)*, IEEE, 2022, pp. 3714–3719.

[39] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems", *Ieee Access*, vol. 8, pp. 71 752–71 762, 2020.

[40] Amazon Web Services, *What is reinforcement learning?*, Accessed: 2024-02-11, 2024. [Online]. Available: `https://aws.amazon.com/what-is/reinforcement-learning/`.

[41] B. M. Kayhan and G. Yildiz, "Reinforcement learning applications to machine scheduling problems: A comprehensive literature review", *Journal of Intelligent Manufacturing*, vol. 34, no. 3, pp. 905–929, 2023.

[42] A. Jayanetti, S. Halgamuge, and R. Buyya, "A deep reinforcement learning approach for cost optimized workflow scheduling in cloud computing environments", in *Proceedings of the 2024 Asia Pacific Conference on Computing Technologies, Communications and Networking*, 2024, pp. 74–82.

[43] A. M. Kintsakis, F. E. Psomopoulos, and P. A. Mitkas, “Reinforcement learning based scheduling in a workflow management system”, *Engineering Applications of Artificial Intelligence*, vol. 81, pp. 94–106, 2019.

[44] Amazon Web Services, *What to consider when selecting a region for your workloads*, Accessed: 2025-02-23, 2024. [Online]. Available: `https://aws.amazon.com/blogs/architecture/what-to-consider-when-selecting-a-region-for-your-workloads/`.

[45] A. W. Services, *Amazon ec2 instance types*, `https://aws.amazon.com/ec2/instance-types/`, Accessed: 2024-12-17.

[46] P. W. Gallery. “Cybershake workflow”. Accessed: 2025-02-23. (2024), [Online]. Available: `https://pegasus.isi.edu/workflow_gallery/gallery/cybershake/index.php`.

[47] A. W. Services, *Aws network infrastructure performance*, Accessed: 2024-12-10, 2024. [Online]. Available: `https://ap-southeast-2.console.aws.amazon.com/nip/home`.

[48] Microsoft. “New zealand’s first hyperscale cloud is open for business”. Accessed: 2024-12-12, Reseller News. (2024), [Online]. Available: `https://news.microsoft.com/en-nz/2024/12/12/new-zealands-first-hyperscale-cloud-is-open-for-business`.

[49] A. W. Services, *Aws ec2 spot instance*, Accessed: 2024-12-02, 2024. [Online]. Available: `https://aws.amazon.com/ec2/spot`.

[50] A. W. Services, *Aws ec2 spot instance advisor*, Accessed: 2024-12-02, 2024. [Online]. Available: `https://aws.amazon.com/ec2/spot/instance-advisor`.