Chapter 1

Introduction

The rapid evolution of cloud computing has transformed the way computing resources are provisioned and utilized, offering scalable, flexible, and cost-effective solutions for diverse applications. Efficient task scheduling in cloud environments is critical to optimize resource usage, reduce execution time, and minimize operational costs while satisfying various constraints such as deadlines and budgets. This thesis explores AI-enhanced multi-objective techniques to address the complexities of task scheduling in cloud computing, leveraging advanced optimization algorithms and learning-based strategies. The introduction provides a foundational background on cloud computing, highlights the significance and challenges of scheduling, and outlines the performance metrics and tools employed for evaluation, setting the stage for the detailed investigation that follows.

1.1 Background of Cloud Computing

Cloud computing represents a paradigm shift in the provisioning and consumption of computing services, characterized by the delivery of scalable and on-demand resources over the Internet. It abstracts the complexities of infrastructure management, enabling users to focus on application development and business goals rather than underlying hardware or software details. This shift has significantly influenced how organizations manage their IT environments, improve agility, and reduce operational costs [1].

Cloud computing typically offers services across three primary models, Infrastructure as a Service, Platform as a Service, and Software as a Service, which cater to different layers of the computing stack [2]. These models provide varying degrees of control, flexibility, and management for users and developers, and together they facilitate a wide range of computing tasks from simple web hosting to complex scientific simulations and AI-driven analytics [3].

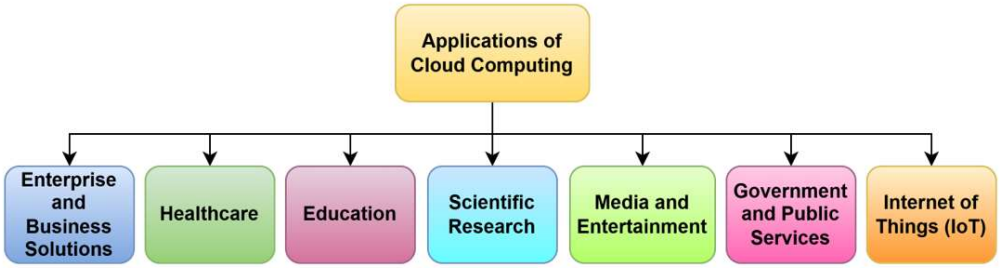


Figure 1.1: Applications of Cloud Computing

1.1.1 Applications of Cloud Computing

Cloud computing has revolutionized the way computational resources are delivered and consumed, offering on-demand access to a shared pool of configurable computing assets such as servers, storage, applications, and services [4]. Its adaptability and scalability have enabled its integration across a wide range of sectors, as depicted in Figure 1.1, transforming traditional IT infrastructures into flexible and service-oriented models. The following are key domains where cloud computing has found significant applications:

Enterprise and Business Solutions : Businesses leverage cloud platforms to host enterprise resource planning systems, customer relationship management applications, and human resource management systems. These cloud-hosted solutions enable organizations to scale their operations dynamically, reduce capital expenditures, and enhance collaboration across geographically distributed teams [5].

Healthcare : In the healthcare domain, cloud computing facilitates secure storage and sharing of electronic health records, telemedicine services, and real-time access to patient data. Cloud-enabled analytics are also used for predictive diagnostics, treatment planning, and large-scale epidemiological studies, all while ensuring compliance with privacy regulations [6].

Education : Educational institutions utilize cloud services to deploy virtual learning environments, manage digital libraries, and support e-learning platforms. Cloud computing enhances accessibility for students and educators, supports collaborative tools, and ensures the scalability of resources during peak demand periods, such as online examinations or course registration [7].

Scientific Research : Cloud infrastructures offer high-performance computing resources and big data processing capabilities essential for complex simulations, genomic sequencing, climate modeling, and physics experiments. Researchers benefit from elastic compute power without the need for significant upfront hardware investments [8].

Media and Entertainment : The media industry uses cloud platforms for content creation, editing, storage, and streaming. Services like video-on-demand, real-time rendering, and scalable content distribution networks rely heavily on cloud infrastructures to deliver seamless user experiences to global audiences [9].

Government and Public Services : Government agencies adopt cloud computing for digi-

tizing public services, enhancing data transparency, and improving citizen engagement through e-governance platforms. The cloud also aids in disaster recovery, data analytics for policy planning, and secure interdepartmental communication [10].

Internet of Things : The cloud serves as the backbone for IoT systems by providing scalable storage, real-time data processing, and analytics platforms. Cloud computing enables smart cities, industrial automation, and connected healthcare by integrating vast streams of data from sensors and actuators into actionable insights [11].

Overall, the widespread applications of cloud computing demonstrate its potential to serve as a foundational technology across various sectors. This widespread adoption underscores the critical need for efficient resource management and task scheduling to ensure optimal performance, cost-efficiency, and reliability of cloud-based systems.

1.1.2 Key Factors and Motivations :

The widespread adoption of cloud computing across diverse domains is driven by a combination of technological advancements, economic incentives, and the increasing demand for scalable, resilient computing resources. Understanding the key factors and motivations behind the proliferation of cloud computing is essential to grasp the underlying challenges and opportunities, particularly in the context of efficient task scheduling [12]. The following points highlight the major factors:

Cost Efficiency and Resource Optimization : Cloud computing eliminates the need for heavy upfront capital investments in IT infrastructure. The pay-as-you-go pricing model allows organizations to align costs directly with usage, thus improving budget predictability and reducing cost [13]. Efficient scheduling of tasks becomes vital in this context, as it directly impacts the consumption of computational resources and, consequently, the cost.

Scalability and Elasticity : One of the primary motivations for adopting cloud platforms is their ability to dynamically scale resources in response to fluctuating workloads. This elasticity ensures consistent performance during peak usage periods. Effective scheduling mechanisms are essential to maintain service quality and performance without over-provisioning or underutilizing resources [14].

Agility and Innovation Enablement : Cloud services accelerate development cycles by providing on-demand infrastructure for testing, deployment, and scaling. This fosters innovation by enabling rapid prototyping and continuous integration/continuous deployment practices. Efficient scheduling ensures that resource contention and latency do not become bottlenecks during high-velocity software delivery [15].

Global Accessibility and Collaboration : Cloud platforms support ubiquitous access to applications and data, empowering geographically distributed teams to collaborate seamlessly. Motivated by this global reach, organizations increasingly rely on cloud services for remote work, real-time data sharing, and cross-border projects. Scheduling algorithms must therefore account

for latency, bandwidth, and geographic placement of resources [16].

Support for Emerging Technologies : The evolution of data-intensive and compute-heavy applications, such as Artificial Intelligence (AI), Machine Learning (ML), big data analytics, and the Internet of Things, demands robust and flexible computing environments. Cloud platforms meet this demand, but the underlying task scheduling mechanisms must also evolve to efficiently handle complex, multi-objective optimization scenarios inherent to such technologies [17].

Reliability and Business Continuity : Cloud computing offers built-in mechanisms for redundancy, fault tolerance, and disaster recovery, which are critical for business continuity. Effective scheduling strategies contribute to this reliability by distributing workloads intelligently, avoiding resource contention, and minimizing downtime [18].

Environmental Considerations : Energy efficiency and sustainable computing are emerging concerns in modern IT practices. By enabling optimal resource utilization, cloud computing reduces the carbon footprint compared to traditional data centers. Task scheduling plays a crucial role in improving energy efficiency by minimizing idle resource time and consolidating workloads [19].

The motivating factors behind cloud adoption are deeply intertwined with the need for intelligent and adaptive task scheduling. As cloud environments continue to evolve in complexity and scale, multi-objective scheduling strategies are becoming indispensable for maximizing performance, minimizing costs, and ensuring sustainable operations.

1.2 Scheduling in Cloud Computing

Scheduling plays a critical role in cloud computing by efficiently allocating computational tasks to available resources in order to optimize performance metrics such as execution time, cost, energy consumption and resource utilization. Due to the dynamic and distributed nature of cloud environments, task scheduling is a complex problem that directly impacts the overall Quality of Service (QoS) and system throughput.

1.2.1 Importance of Efficient Scheduling in Cloud Environments

Efficient task scheduling is a cornerstone of performance optimization in cloud computing environments. As cloud systems are inherently dynamic and heterogeneous, scheduling determines how computational resources such as CPU cycles, memory, and bandwidth are allocated to a diverse set of tasks. An effective scheduling mechanism can significantly enhance resource utilization, minimize execution time, reduce costs, and meet Service-Level Agreements (SLAs) [20].

In a cloud ecosystem, users typically submit large numbers of tasks with varying computational demands and constraints, including deadlines, budgets, and priorities. Cloud providers, in turn, must manage these tasks using finite resources while maintaining fairness, maximizing throughput, and minimizing latency. Inefficient scheduling can lead to resource underutilization,

increased energy consumption, SLA violations, and sub-optimal application performance [21].

Moreover, cloud computing's support for multi-tenant execution environments introduces challenges such as task interference, load imbalance, and unpredictable resource availability. Efficient scheduling helps mitigate these issues by ensuring tasks are assigned to appropriate resources based on real-time system states, workload characteristics, and optimization goals [22]

With the growing reliance on cloud infrastructure for mission-critical and latency-sensitive applications, such as data analytics, online transaction processing, and AI-driven workloads, the importance of intelligent, adaptive, and multi-objective scheduling strategies continues to rise. These strategies aim to strike a balance between competing objectives like execution time and cost, energy efficiency, deadline and budget adherence, and system responsiveness [23].

The efficiency of task scheduling in cloud environments directly affects both the economic and operational aspects of cloud service delivery. As such, it remains an active and vital area of research and development within the broader field of cloud computing.

1.2.2 Challenges in Scheduling

Scheduling tasks efficiently in cloud computing environments presents numerous challenges due to the dynamic, heterogeneous, and complex nature of cloud infrastructures and applications [24]. These challenges impact the ability to achieve optimal resource utilization, meet QoS requirements, and maintain cost-effectiveness. Key challenges in cloud task scheduling include:

- **Resource Heterogeneity and Dynamism:** Cloud environments consist of diverse resources with varying capabilities, availability, and performance characteristics. The dynamic allocation and deallocation of virtualized resources add further complexity to scheduling decisions [25].
- **Multi-Objective Optimization:** Scheduling often involves optimizing multiple conflicting objectives such as minimizing execution time, cost, and energy consumption while maximizing reliability and throughput. Balancing these objectives is inherently challenging [26].
- **Task Dependencies and Workflow Complexity:** Many applications consist of workflows with intricate dependency structures, requiring careful ordering and synchronization of tasks to ensure correctness and efficiency [27].
- **Scalability:** Cloud platforms serve a vast number of users and applications simultaneously, necessitating scheduling algorithms that can scale efficiently with increasing workloads and resource pools [28].
- **Uncertainty and Variability:** Variations in resource performance, network latency, and workload patterns introduce uncertainty, making it difficult to predict task execution times and plan schedules accurately [29].

- **Cost and Budget Constraints:** Users often impose strict budget limits on resource usage, requiring scheduling algorithms to optimize within financial constraints without compromising performance [30].
- **Fault Tolerance and Reliability:** Cloud resources may fail or become unavailable unexpectedly. Scheduling must incorporate mechanisms to detect failures and recover tasks to maintain system reliability[31].
- **Security and Privacy Concerns:** Sensitive data and tasks may require execution on secure or compliant resources, adding constraints that complicate scheduling decisions [32].
- **Energy Efficiency:** The growing environmental impact and operational costs of cloud data centers drive the need for energy-aware scheduling strategies that reduce power consumption without sacrificing performance [33].

Addressing these challenges demands sophisticated scheduling algorithms that leverage advanced optimization techniques, ML, and adaptive heuristics. Furthermore, simulation and modeling tools play a crucial role in evaluating scheduling strategies under realistic cloud conditions.

1.2.3 Independent Task Scheduling

Independent tasks scheduling involves allocating tasks that have no interdependencies and can be executed concurrently or in any order. This type of scheduling is simpler compared to workflow scheduling but still requires strategies to optimize resource usage, minimize completion time, and balance load across virtual machines [34]. Common approaches include heuristic and metaheuristic algorithms designed to handle the allocation of independent, parallelizable tasks efficiently. In cloud environments, independent tasks are often encountered in scientific computing, Monte Carlo simulations, large-scale data processing, and job-based applications like image rendering or video encoding. These tasks are typically submitted in bulk and require efficient allocation to ensure scalability and responsiveness of the system [35].

1.2.4 Workflow Scheduling in Cloud

Workflow scheduling in cloud computing refers to the process of mapping and executing a sequence of interdependent tasks, known as a workflow, on distributed cloud resources. Figure 1.2 depicts the framework for workflow scheduling in cloud. Each task within a workflow may have dependencies that decide the order of execution. It is often represented as a Directed Acyclic Graph (DAG), where nodes correspond to tasks and edges indicate data or control dependencies.

In cloud environments, workflow scheduling plays a crucial role in enabling the efficient execution of complex scientific, engineering, and business applications. Examples include bioinformatics pipelines, climate modeling, financial simulations, and multimedia processing. These applications often require high computational power, data storage, and stringent adherence to QoS

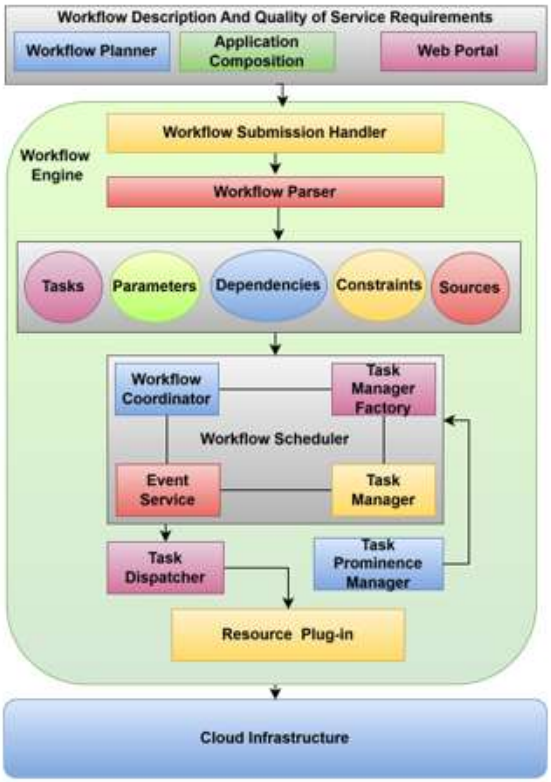


Figure 1.2: Workflow Scheduling Framework

requirements such as deadlines and budgets. The key challenge in workflow scheduling is to allocate tasks to heterogeneous and virtualized resources while preserving the dependency structure and optimizing multiple objectives [36].

Workflow scheduling in the cloud differs significantly from independent task scheduling due to the need to respect task dependencies. A delay in the execution of one task can propagate and affect the entire workflow, making intelligent task prioritization and resource provisioning essential [37]. Moreover, workflow management systems like Pegasus, Taverna, and Kepler, along with simulation tools such as WorkflowSim, are frequently used to model, simulate, and evaluate workflow scheduling strategies. Workflow scheduling is a complex but essential aspect of cloud computing, demanding sophisticated techniques to meet application requirements while ensuring efficient use of the underlying cloud infrastructure [38].

1.2.5 Scheduling Constraints

Scheduling in cloud computing is subject to a variety of constraints that significantly influence the design and effectiveness of scheduling algorithms. These constraints originate from the inherent characteristics of cloud resources, user requirements, and application-specific needs. Understanding and addressing these constraints is crucial for achieving efficient and reliable task scheduling. Key scheduling constraints include:

- **Resource Availability:** Cloud resources such as virtual machines (VMs), storage, and network bandwidth may have limited availability and can fluctuate dynamically due to multi-tenancy, maintenance, or failures.
- **Task Dependencies:** In workflow scheduling, tasks often have precedence relationships that decide the order of execution. Scheduling must respect these dependencies to ensure correctness.
- **Deadline Constraints:** Many applications impose strict deadlines within which tasks or workflows must be completed to meet QoS requirements.
- **Budget Constraints:** Users may specify monetary limits for executing their tasks on cloud resources, requiring scheduling algorithms to optimize within cost boundaries.
- **Resource Heterogeneity:** Cloud environments consist of diverse resources with varying computational capabilities, memory sizes, and costs, which affect task-to-resource mapping decisions.
- **Communication Overhead:** Data transfer between tasks or between cloud nodes introduces latency and costs, especially in distributed or multi-cloud setups, influencing scheduling decisions.
- **Security and Privacy Requirements:** Certain tasks may require execution on trusted or geographically restricted resources to comply with security policies or data privacy regulations.
- **Energy Consumption Limits:** Some cloud providers or applications impose limits on energy usage, necessitating energy-aware scheduling strategies.
- **Scalability and Elasticity:** Scheduling must accommodate scaling resources up or down based on workload fluctuations, while minimizing disruption and maintaining performance.

These constraints often interact in complex ways, requiring scheduling algorithms to balance competing demands. Effective scheduling strategies incorporate constraint-aware mechanisms such as constraint satisfaction techniques, penalty functions, or adaptive heuristics to ensure feasible and optimized task execution. Scheduling constraints define the boundaries within which cloud task scheduling operates, and addressing them is essential for delivering reliable, efficient, and cost-effective cloud services.

1.2.6 Scheduling Objectives

Scheduling in cloud computing aims to optimize the allocation of tasks to computational resources while satisfying various performance and quality criteria. The objectives of scheduling are often multi-faceted and sometimes conflicting, requiring careful trade-offs to achieve an optimal or near-optimal solution. The primary scheduling objectives in cloud environments include:

- **Minimization of Makespan:** Reducing the total completion time of all scheduled tasks is crucial to improving system throughput and ensuring timely execution, especially for latency-sensitive applications [39].
- **Cost Efficiency:** Cloud resources are typically billed based on usage metrics such as CPU time, memory, and data transfer. Scheduling algorithms strive to minimize the monetary cost incurred while meeting user requirements [40].
- **Deadline and Budget Adherence:** Many applications require strict deadlines and budgets for task or workflow completion. Scheduling must ensure that these temporal constraints are respected to meet QoS agreements [41].
- **Energy Efficiency:** Reducing the energy consumption of cloud data centers has become increasingly important for environmental sustainability and operational cost reduction. Scheduling strategies often incorporate energy-aware objectives [42].
- **Load Balancing:** Distributing tasks evenly across available resources prevents bottlenecks and maximizes resource utilization, thereby enhancing overall system stability and performance [43].
- **Reliability and Fault Tolerance:** Scheduling must account for the possibility of resource failures and incorporate mechanisms to maintain task execution integrity, such as replication and checkpointing [44].
- **Scalability and Adaptability:** The scheduling approach should efficiently handle varying workloads and dynamically changing resource availability without significant degradation in performance [45].

Balancing these objectives poses significant challenges, as optimizing one may negatively impact another—for example, minimizing cost might increase makespan or reduce reliability. Therefore, multi-objective optimization techniques, including Pareto optimality concepts and weighted objective functions, are frequently employed to identify acceptable trade-offs [43]. Hence, the scheduling objectives define the criteria by which scheduling algorithms are evaluated and guide the design of effective task allocation strategies within cloud computing environments.

1.3 System Model

This section presents the system model underlying the proposed task scheduling framework in the cloud computing environment. The model encompasses both the task and resource layers, detailing the characteristics of computational tasks and the configuration of the cloud infrastructure. It assumes a static, non-preemptive scheduling scenario where all task information is known in advance, allowing for global optimization. The model also incorporates essential scheduling metrics such as makespan, cost, energy consumption, resource utilization and scheduling overhead, providing a foundation for developing and evaluating multi-objective optimization algorithms.

1.3.1 Task Model

In a cloud computing environment, the task scheduling problem involves a collection of N independent computational tasks, denoted as $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_N\}$. Each task τ_i is associated with a specific computational workload and is defined by three key parameters: the task length ℓ_{τ_i}, which represents the number of computational instructions (expressed in Millions of Instructions (MI)); the input data size ι_{τ_i}, required prior to execution; and the output data size o_{τ_i}, generated upon task completion. Accordingly, the set of tasks τ_i can be formally represented as in Equation 1.1:

$$\tau_i = \{\ell_{\tau_i}, \iota_{\tau_i}, o_{\tau_i}\}. \tag{1.1}$$

1.3.2 Cloud Resource Model

In a cloud computing environment, tasks are executed using virtualized infrastructure provisioned by cloud service providers. This infrastructure comprises physical servers—referred to as hosts—that support the instantiation of VMs. These VMs are the fundamental computational units on which tasks are scheduled and executed. Each VM is characterized by a specific type, processing capacity, memory size, and associated cost. While the number of VM instances is considered virtually infinite due to the elastic provisioning model of the cloud, the number of distinct VM types is finite, constrained by hardware configurations offered by the provider.

Let the cloud datacenter be denoted by D, and let $\mathcal{V}m = \{v_1, v_2, \ldots, v_V\}$ represent the set of V active VMs currently provisioned in the datacenter. Each VM $v_\nu \in \mathcal{V}m$ is hosted on a physical machine and is allocated a certain amount of computing power in terms of processing elements (PEs) and bandwidth for data transmission.

VM Instance Capacity ($\mathcal{C}_{v_\nu}$): The computational capacity of an individual VM v_ν is defined by the number of instructions it can process per unit time, typically measured in Millions of Instructions Per Second (MIPS). This capacity depends on two parameters: the number of PEs assigned to the VM, denoted as $\mathcal{PE}_v^n$, and the processing speed of each element, denoted as $\mathcal{PE}_v^s$. Accordingly, the total processing capacity of VM v_ν is given by the Equation 1.2:

$$\mathcal{C}_{v_\nu} = \mathcal{PE}_v^n \times \mathcal{PE}_v^s \tag{1.2}$$

Total Capacity of Active VMs ($\mathcal{C}_{\mathcal{V}m}$): To evaluate the computational strength available for task execution, the total capacity of all active VMs in the datacenter D is computed as the sum of the capacities of individual VMs. This aggregated capacity, denoted as $\mathcal{C}_{\mathcal{V}m}$, is defined as in Equation 1.3:

$$\mathcal{C}_{\mathcal{V}m} = \sum_{\nu=1}^{V} \mathcal{C}_{v_\nu} \tag{1.3}$$

This value represents the total instruction-processing potential that can be leveraged during scheduling.

Datacenter Capacity Constraint ($\mathcal{C}_\mathcal{D}$): Although VM provisioning in the cloud is scalable,

the physical datacenter still has a finite upper limit on the total computing power it can offer at any given time. Therefore, the combined processing capacity of all provisioned VMs must not exceed the maximum datacenter capacity. This constraint is formalized as in Equation 1.4:

$$\mathcal{C}_{\mathcal{V}m} \leq \mathcal{C}_D \tag{1.4}$$

This constraint ensures that the scheduling and execution of tasks remain feasible with respect to the available cloud infrastructure and prevents over-subscription of computational resources. The cloud resource model thus provides a foundational structure for modeling VM characteristics and capacity constraints, which are essential inputs to any task scheduling strategy in cloud systems.

1.3.3 Task Scheduling Model

Task scheduling in cloud computing involves allocating a set of computational tasks to available VMs with the objective of optimizing various performance metrics. This study adopts a *static scheduling* approach, enabling the utilization of global information to achieve optimal resource allocation decisions.

Execution Time ($\mathcal{E}_{\tau_i}^{v_j}$): The execution time quantifies the duration required by a virtual machine v_j to process a given task τ_i. This duration depends on the computational workload of the task and the processing capacity of the assigned VM. The execution time is mathematically expressed as given in the Equation 1.5:

$$\mathcal{E}_{\tau_i}^{v_j} = \frac{\ell_{\tau_i}}{\mathcal{C}_{v_j}}, \tag{1.5}$$

where ℓ_{τ_i} represents the task length (in MI), and $\mathcal{C}_{v_j}$ denotes the processing speed of VM v_j (in MIPS). Accordingly, tasks assigned to higher-capacity VMs experience shorter execution times.

Waiting Time ($\mathcal{W}_{\tau_i}$): Waiting time refers to the period a task remains in the queue before it begins execution, typically due to the unavailability of computing resources. If a VM is immediately available, the waiting time is zero. Otherwise, it is determined by the earliest completion time among tasks currently occupying the VM. The waiting time is given by the Equation 1.6:

$$\mathcal{W}_{\tau_i} = \begin{cases} 0, & \text{if } \mathcal{A}_{\mathcal{V}} \neq \emptyset, \\ \min(\mathcal{C}_{\mathcal{P}_{\tau_i}}), & \text{if } \mathcal{A}_{\mathcal{V}} = \emptyset, \end{cases} \tag{1.6}$$

where $\mathcal{A}_{\mathcal{V}}$ is the set of currently available VMs, and $\mathcal{C}_{\mathcal{P}_{\tau_i}}$ denotes the completion times of tasks preceding τ_i on the assigned VM.

Completion Time ($\mathcal{C}_{\tau_i}$): Completion time indicates the exact moment at which task τ_i finishes execution on its assigned VM. It comprises both the waiting time and the execution time as

formulated in the Equation 1.7:

$$\mathcal{C}_{\tau_i} = \begin{cases} \mathcal{E}_{\tau_i}^{v_j}, & \text{if } \mathcal{A}_{\mathcal{V}} \neq \emptyset, \\ \mathcal{W}_{\tau_i} + \mathcal{E}_{\tau_i}^{v_j}, & \text{if } \mathcal{A}_{\mathcal{V}} = \emptyset. \end{cases} \tag{1.7}$$

Minimizing the completion time of individual tasks contributes to improved task throughput and overall system responsiveness.

Makespan ($\mathcal{M}$): Makespan is a critical metric in task scheduling, defined as the maximum completion time among all tasks in the workflow. It serves as an indicator of the total time required to complete the entire workload. It can be formulated as the Equation 1.8:

$$\mathcal{M} = \max(\mathcal{C}_{\tau_i}), \quad \forall i \in [1, N]. \tag{1.8}$$

Minimizing makespan is essential for enhancing system performance and achieving timely execution of tasks.

Scheduling Time ($\mathcal{S}$): Scheduling time refers to the computational time incurred by the scheduling algorithm in determining optimal task-to-VM mappings before actual task execution begins. It is computed as in Equation 1.9:

$$\mathcal{S} = \mathcal{T}_{total} - \mathcal{M}, \tag{1.9}$$

where $\mathcal{T}_{total}$ denotes the total time elapsed from the submission of the first task to the completion of the last. A lower scheduling time indicates a more efficient scheduling mechanism.

1.3.4 Workflow Model

In cloud computing environments, a workflow is typically modeled as a DAG, denoted as $G(\mathcal{T}, \mathcal{E})$, where $\mathcal{T} = \{t_1, t_2, \ldots, t_T\}$ is the set of computational tasks, and $\mathcal{E} \subset \mathcal{T} \times \mathcal{T}$ represents the set of directed edges that define precedence constraints among tasks. An edge $e_{h,\iota} \in \mathcal{E}$ indicates that task t_ι depends on task t_h, implying that t_ι can only begin execution after task t_h has completed and the necessary data has been transferred. In this relationship, t_h is called the predecessor, and t_ι is the successor task [46].

Each task $t_\iota \in \mathcal{T}$ is associated with three main attributes. The task length, denoted by $\mathcal{TL}_\iota$, represents the number of computational instructions required to execute the task, typically measured in MI. The input data required by the task is given by the input data file list $\mathcal{ID}_\iota$, and the data produced upon completion is represented by the output data file list $\mathcal{OD}_\iota$. Formally, the workflow model can be represented as the Equation 1.10:

$$\mathcal{W} = \left\{ G(\mathcal{T}, \mathcal{E}), \{\mathcal{TL}_\iota, \mathcal{ID}_\iota, \mathcal{OD}_\iota\}_{\forall t_\iota \in \mathcal{T}} \right\} \tag{1.10}$$

1.3.5 Workflow Scheduling Model

Workflow scheduling in cloud environments involves the process of mapping a set of interdependent computational tasks, represented as a workflow, to a set of available VMs such that scheduling objectives are achieved. This work adopts a static scheduling approach due to its global optimization potential and deterministic performance under known workload conditions. The workflow scheduling model is governed by a set of timing relationships and communication delays, as described below.

Execution Time ($\mathcal{ET}_{t_\iota}^{v_\nu}$): The execution time of a task t_ι of a workflow $\mathcal{W}$ on a virtual machine v_ν is determined by dividing the task's length $\mathcal{TL}_\iota$ by the processing capacity $\mathcal{C}_{v_\nu}$ of the VM [47]. This relationship is defined as in Equation 1.11:

$$\mathcal{ET}_{t_\iota}^{v_\nu} = \frac{\mathcal{TL}_\iota}{\mathcal{C}_{v_\nu}} \tag{1.11}$$

Transfer Time ($\mathcal{TT}_{v_\nu}^{v_\kappa}$): If two dependent tasks are assigned to different VMs, there is a communication delay due to the transfer of output data from the parent task's VM to the child task's VM. This transfer time is calculated by the Equation 1.12:

$$\mathcal{TT}_{v_\nu}^{v_\kappa} = \frac{\mathcal{DS}_{v_\nu}^{v_\kappa}}{\overline{\mathcal{B}}} \tag{1.12}$$

Here, $\mathcal{DS}_{v_\nu}^{v_\kappa}$ is the size of the data being transferred, and $\overline{\mathcal{B}}$ is the average bandwidth of the cloud network.

Data Communication Time ($\mathcal{DCT}_{t_\iota}^{\mathcal{P}_{t_\iota}}$): When a task t_ι has multiple predecessor tasks $\mathcal{P}_{t_\iota}$, and these are executed on different VMs, the time to collect all the required data is dominated by the longest data transfer. This is calculated by Equation 1.13:

$$\mathcal{DCT}_{t_\iota}^{\mathcal{P}_{t_\iota}} = \max(\mathcal{TT}_{v_\mu}^{v_\kappa}) \tag{1.13}$$

where $v_\mu \in \mathcal{V}m$ are the VMs executing the predecessors of t_ι, and v_κ is the VM assigned to t_ι.

Waiting Time ($\mathcal{WT}_{t_\iota}$): Waiting time for a task t_ι is defined as the time it must wait before it can start execution. This includes the time until all its predecessors complete, and if scheduled on a different VM, also includes the data communication delay. Formally, it is given by the Equation 1.14:

$$\mathcal{WT}_{t_\iota} = \begin{cases} 0 & \text{if } \mathcal{P}_{t_\iota} = \varnothing \\ \max(\mathcal{CT}_{\mathcal{P}_{t_\iota}}) & \text{if } \mathcal{P}_{t_\iota} \neq \varnothing \text{ and } v^{t_\iota} = v^{\mathcal{P}_{t_\iota}} \\ \max(\mathcal{CT}_{\mathcal{P}_{t_\iota}}) + \mathcal{DCT}_{t_\iota}^{\mathcal{P}_{t_\iota}} & \text{if } \mathcal{P}_{t_\iota} \neq \varnothing \text{ and } v^{t_\iota} \neq v^{\mathcal{P}_{t_\iota}} \end{cases} \tag{1.14}$$

Completion Time ($\mathcal{CT}_{t_\iota}$): The completion time of task t_ι is the sum of its execution time

and the time it must wait before beginning execution. This is expressed as in Equation 1.15:

$$\mathcal{CT}_{t_\iota} = \begin{cases} \mathcal{ET}_{t_\iota}^{v_\nu} & \text{if } \mathcal{P}_{t_\iota} = \varnothing \\ \mathcal{WT}_{t_\iota} + \mathcal{ET}_{t_\iota}^{v_\nu} & \text{if } \mathcal{P}_{t_\iota} \neq \varnothing \end{cases} \tag{1.15}$$

Makespan ***($\mathcal{M}_\mathcal{W}$)***: The makespan of a workflow, which serves as a key performance metric, is defined as the maximum completion time among all tasks in the workflow. It reflects the total time required to execute the entire workflow from start to finish as depicted in Equation 1.16:

$$\mathcal{M}_\mathcal{W} = \max(\mathcal{CT}_{t_\iota}), \quad \text{for } 1 \leq \iota \leq T \tag{1.16}$$

1.3.6 Cost Model

In a cloud computing environment, task execution and data transfer both incur monetary costs. These costs vary depending on the computational resources used and the amount of data communicated between VMs. The cost model used in this study quantifies the economic expense associated with executing a workflow $\mathcal{W}$, comprising both execution and data transfer components.

Execution Cost ***($\mathcal{EC}_\mathcal{W}$)***: The execution cost for a single task t_ι on a virtual machine v_ν is determined by the product of the execution time $\mathcal{ET}_{t_\iota}^{v}$ and the cost per unit time of the VM $\mathcal{C}_{v_\nu}$. It is formally expressed as in Equation 1.17

$$\mathcal{EC}_{t_\iota}^{v_\nu} = \mathcal{ET}_{t_\iota}^{v_\nu} \times \mathcal{C}_v \tag{1.17}$$

The total execution cost for the entire workflow $\mathcal{W}$, comprising T tasks, is the sum of the execution costs of all individual tasks as depicted by Equation 1.18:

$$\mathcal{EC}_\mathcal{W} = \sum_{\iota=1}^{T} \mathcal{EC}_{t_\iota}^{v_\nu} \tag{1.18}$$

Transfer Cost ***($\mathcal{TC}_\mathcal{W}$)***: When dependent tasks are scheduled on different VMs, data transfer between them incurs additional costs. The transfer cost for task t_ι receiving data on VM v_κ from a parent task on VM v_ν is defined as in Equation 1.19:

$$\mathcal{TC}_{t_\iota}^{v_\kappa} = \frac{\mathcal{DS}_{v_\nu}^{v_\kappa} \times \mathcal{C}_B}{\overline{\mathcal{B}}} \tag{1.19}$$

Here, $\mathcal{DS}_{v_\nu}^{v_\kappa}$ denotes the size of the data transferred between the VMs, $\mathcal{C}_B$ is the cost per unit bandwidth, and $\overline{\mathcal{B}}$ is the average available bandwidth.

The total transfer cost for the workflow is computed as the sum of the transfer costs incurred

across all data dependencies, represented by Equation 1.20:

$$\mathcal{TC}_{\mathcal{W}} = \sum_{\iota=1}^{T} \mathcal{TC}_{t_\iota}^{v_\kappa} \tag{1.20}$$

Total Cost *($Cost_{\mathcal{W}}$)*: The total cost for executing a task includes both its execution and data transfer costs. The cost for an individual task t_ι is calculated as in Equation 1.21:

$$\mathcal{C}ost_{t_\iota} = \mathcal{EC}_{t_\iota}^{v_\nu} + \mathcal{TC}_{t_\iota}^{v_\nu} \tag{1.21}$$

Accordingly, the total cost for executing the complete workflow $\mathcal{W}$ is given by the Equation 1.22:

$$\mathcal{C}ost_{\mathcal{W}} = \mathcal{EC}_{\mathcal{W}} + \mathcal{TC}_{\mathcal{W}} \tag{1.22}$$

1.3.7 Energy Consumption Model

In cloud computing, energy efficiency is a critical design consideration due to its direct impact on operational costs and environmental sustainability. The energy model used in this study quantifies the power consumption and energy usage of VMs during the execution of a workflow. The energy consumed is primarily a function of the VM's operating frequency and voltage, which vary based on the workload and scheduling decisions.

Running Voltage *($\mathcal{V}_\nu^r$)*: The dynamic voltage of a virtual machine v_ν under a specific operating condition is determined using linear interpolation between its minimum and maximum voltage levels, based on the corresponding frequency range. The running voltage is computed as in Equation 1.23:

$$\mathcal{V}_\nu^r = \mathcal{V}_\nu^m + (\mathcal{V}_\nu^M - \mathcal{V}_\nu^m) \times \frac{\mathcal{F}_\nu^r - \mathcal{F}_\nu^m}{\mathcal{F}_\nu^M - \mathcal{F}_\nu^m} \tag{1.23}$$

Here, $\mathcal{V}_\nu^m$ and $\mathcal{V}_\nu^M$ denote the minimum and maximum voltage levels, respectively. $\mathcal{F}_\nu^r$ is the current (or running) frequency of the VM, while $\mathcal{F}_\nu^m$ and $\mathcal{F}_\nu^M$ denote the minimum and maximum operating frequencies, respectively.

Power Consumption *($\mathcal{P}_\nu$)*: The power consumed by a VM v_ν is modeled using a simplified dynamic power formula based on its running frequency and voltage. The power consumption is given by the Equation 1.24:

$$\mathcal{P}_\nu = \mathcal{F}_\nu^r \times (\mathcal{V}_\nu^r)^2 \tag{1.24}$$

This expression captures the direct dependence of power usage on both clock frequency and square of voltage, reflecting the behavior of PEs.

Energy Consumption *($\mathcal{E}_{\mathcal{W}}$)*: The total energy consumption of the workflow $\mathcal{W}$ is calculated as the cumulative energy consumed by all active VMs during their execution lifetimes. If $\mathcal{AT}_\nu$ denotes the active time of VM v_ν, then the energy consumption across all VMs can be depicted

by the Equation 1.25:

$$\mathcal{E}_{\mathcal{W}} = \sum_{\nu=1}^{V} \mathcal{P}_{\nu} \times \mathcal{AT}_{\nu} \tag{1.25}$$

1.3.8 Resource Utilization Model

Resource utilization is a key performance indicator in cloud computing environments, especially when evaluating the efficiency of scheduling algorithms. High utilization of VMs indicates that the computational resources are being effectively employed to process workflow tasks, minimizing idle times and operational waste.

Utilization ***($\mathcal{U}_{\mathcal{W}(\%)}$)***: Let $\mathcal{W}$ denote a workflow scheduled for execution on a datacenter $\mathcal{D}$, which hosts V active VMs$\{v_1, v_2, \ldots, v_V\}$. The variable $\mathcal{AT}_{\nu}$ represents the active time of VM v_{ν}, i.e., the total duration during which the VM is engaged in executing one or more workflow tasks. The makespan of the workflow, denoted $\mathcal{M}_{\mathcal{W}}$, corresponds to the overall time from the start of the first task to the completion of the last task.

The utilization of the workflow $\mathcal{W}$, expressed as a percentage, is calculated by computing the average active time ratio across all VMs and then scaling it by 100. This can be depicted by the Equation 1.26:

$$\mathcal{U}_{\mathcal{W}(\%)} = \frac{\sum_{\nu=1}^{V} \frac{\mathcal{AT}_{\nu}}{\mathcal{M}_{\mathcal{W}}}}{V} \times 100 \tag{1.26}$$

The utilization model adopted in this study quantifies the average percentage of time the active VMs are engaged in task execution relative to the total time taken to complete the workflow.

1.4 Framework and Tools

This section introduces the primary simulation and development tools used for implementing and evaluating task scheduling strategies in cloud computing environments.

1.4.1 CloudSim

CloudSim is a widely used simulation framework for modeling and simulating cloud computing environments and services. It provides a flexible and extensible platform to evaluate resource provisioning and scheduling algorithms without the need for actual cloud infrastructure.

CloudSim supports modeling of data centers, virtual machines, applications, and user workloads, allowing researchers to analyze the performance, scalability, and cost-effectiveness of various cloud scheduling strategies under different scenarios.

Its modular architecture enables customization and integration of new policies, making it a popular choice for academic research in cloud computing task scheduling.

1.4.2 WorkflowSim

WorkflowSim is an extension of CloudSim designed specifically to simulate and evaluate workflow scheduling in cloud environments. It provides enhanced support for modeling complex workflows, dependencies between tasks, and resource management.

By enabling accurate simulation of scientific workflows, WorkflowSim helps researchers analyze the performance of scheduling algorithms that handle task dependencies and workflow execution constraints. It supports various workflow models and allows experimentation with different scheduling policies and resource provisioning strategies.

WorkflowSim's integration with CloudSim ensures flexibility and extensibility, making it an effective tool for workflow scheduling in cloud computing.

1.4.3 Google Colaboratory

Google Colaboratory, commonly known as Google Colab, is a cloud-based interactive computing environment that allows users to write and execute Python code through a web browser. It provides free access to computing resources, including GPUs and TPUs, making it an excellent platform for developing and testing ML and optimization algorithms.

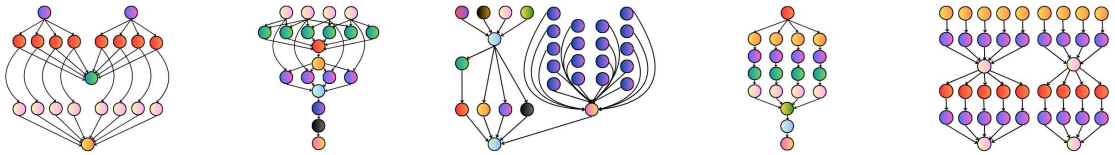
In the context of cloud computing task scheduling, Google Colab serves as a convenient tool for implementing, experimenting, and validating scheduling algorithms, especially those involving deep learning and Reinforcement Learning (RL) techniques.

Its collaborative features and integration with popular libraries and frameworks accelerate research and development workflows without requiring local hardware resources.

1.5 Benchmark Workflows

To rigorously evaluate the performance of the proposed scheduling algorithm, a diverse set of real-world scientific workflows is employed as benchmarks. These workflows span multiple scientific domains and exhibit varying structural complexities, task dependencies, and computational characteristics, making them suitable for comprehensive performance analysis.

The selected benchmarks include Montage (astronomy), CyberShake (earthquake hazard), LIGO Inspiral Analysis (gravitational wave physics), Epigenomics (bioinformatics) and SIPHT (bioinformatics) [48] [49]. The structure of these workflows has been presented in Figures 1.3a - 1.3e. Each workflow represents a distinct application domain and encompasses a mix of CPU-intensive, I/O-intensive, and data-parallel tasks [50]. These workflows have been widely used in the literature for evaluating resource management and scheduling strategies in cloud and grid environments, thus providing a reliable foundation for comparative analysis.



(a) CyberShake (b) Montage (c) SIPHT (d) Epigenomics (e) Inspiral

Figure 1.3: Structure of Scientific Workflows

1.5.1 Montage

The Montage workflow constructs astronomical image mosaics, with its size determined by the number of input FITS images. Each image is initially reprojected using `mProjectPP`, producing a reprojected image and an area map. Overlapping image pairs are then processed by `mDiffFit`, which computes difference images. These differences are aggregated using a least squares method by `mConcatFit`, followed by `mBgModel`, which calculates a global background correction. This correction is then applied to individual images using `mBackground`. Metadata from all images is collected by `mImgTbl`, and the final mosaic is generated by `mAdd`, which combines the reprojected images into a single FITS file and corresponding area image. Among all tasks, `mAdd` is the most computationally intensive and I/O-heavy. The final mosaic is downsampled by `mShrink` and converted to JPEG format by `mJPEG`. For larger workflows, multiple levels of `mImgTbl` and `mAdd` may be used [51].

Profiling data for a workflow generating a mosaic of an 8-degree square sky region shows that most of the execution time is spent in `mAdd`, which handles substantial I/O (1 GB read and 775 GB written per job) with low CPU utilization. Other tasks, such as `mBackground`, `mImgTbl`, `mAdd`, and `mShrink`, also demonstrate I/O-intensive, CPU-light behavior. Conversely, `mBgModel` had the highest CPU utilization (99.89%) and longest runtime, despite low I/O activity. The high standard deviations in performance metrics for `mAdd` arise from its dual role in a two-level reduction structure, where the second level exhibits different runtime characteristics. Distinguishing these as separate job types could enhance profiling accuracy [52].

1.5.2 SIPHT

The SIPHT workflow is designed to identify small untranslated RNAs (sRNAs) and is composed of smaller, structurally similar sub-workflows. Workflow size is determined by the number of `Patser` jobs, which scan sequences for transcription factor binding sites (TFBSs) using position-specific scoring matrices. Their results are aggregated by the `Patser_Concate` job. Several `BLAST` jobs compare inter-genetic regions (IGRs) across multiple replicons. These include `Blast` and `Blast_QRNA`, which process hundreds of data files. Transcription terminators are identified by three jobs: `FindTerm`, `RNAMotif`, and `Transterm`. The `SRNA` job performs sRNA prediction using outputs from all prior stages, and its results are further analyzed by the `SRNA_Annotate` job to determine conservation, TFBS associations, and homology.

Profiling data indicates that `BLAST` jobs dominate the execution time, accounting for about

74% of the total runtime. The `SRNA` prediction and transcription terminator identification jobs follow in computational cost. Similar to the Epigenome workflow, SIPHT is primarily CPU-bound, with most jobs exhibiting high CPU utilization and low I/O demands. An exception is `Patser_Concate`, which has low CPU usage due to its simple concatenation task. `Blast` and `Blast_QRNA` jobs are the most I/O-intensive, reading around 800 MB and writing about 565 MB each. The `FindTerm` job also writes a significant amount of data (379 MB).

1.5.3 CyberShake

The CyberShake workflow performs earthquake hazard characterization using large datasets, despite its relatively simple structure. It utilizes Strain Green Tensor (SGT) data from finite simulations, stored in large master files for the x and y dimensions. This data quantifies ground motion relationships across regions. Additionally, a collection of future fault rupture scenarios and their variations are used to estimate seismic hazards. `ExtractSGT` jobs retrieve rupture-specific data from the master SGT files and serve as data partitioning tasks. For each rupture variation, `SeismogramSynthesis` generates synthetic seismograms, while `PeakValueCalcOkaya` computes peak intensity values, such as spectral acceleration. These outputs are aggregated and compressed by `ZipSeismograms` and `ZipPeakSA` for staging and archival.

Among the job types, `SeismogramSynthesis` is the most computationally demanding. However, due to the large master SGT file sizes, `ExtractSGT` can also be resource-intensive. Similarly, `ZipSeismograms` and `ZipPeakSA` can consume significant time during compression due to the volume of generated data. Profiling data from a large CyberShake workflow (over 800,000 jobs) executed on the Ranger cluster at TACC shows that over 97% of the total runtime is spent in `SeismogramSynthesis` jobs, indicating a key area for optimization. On average, each synthesis job reads 547 MB, exceeding the known input size (150–250 MB), implying repeated reads and highlighting further optimization potential. These jobs also exhibit high memory demands, averaging 817 MB and peaking at 1870 MB, which may exceed per-core memory limits (typically around 1 GB) on many clusters. Improper scheduling—placing one synthesis job per core—could cause memory exhaustion, leading to failures. This underlines the importance of configuring the scheduler to respect node memory capacities when deploying the workflow on different clusters.

1.5.4 Inspiral LIGO

The LIGO Inspiral Analysis workflow processes data from the coalescence of compact binary systems, including binary neutron stars and black holes. It is a highly complex workflow composed of multiple sub-workflows. While a simplified representation contains a few sub-workflows, actual executions can involve up to 100 sub-workflows and over 200,000 jobs. The `TmpltBank` jobs generate waveform parameter banks for each 2048-second block of input data, and can be executed in parallel. These parameter banks are used by `Inspiral` jobs to perform matched filtering and generate triggers. Outputs from multiple `Inspiral` jobs are aggregated and analyzed for consistency by `Thinca` jobs, which serve as data aggregation tasks. The resulting triggers

are input to `TrigBank` jobs, which produce updated template banks. These are then processed by a second set of `Inspiral` and `Thinca` jobs.

Profiling data shows that `Inspiral` jobs dominate the workflow in terms of computational cost, accounting for approximately 92% of the total runtime and I/O (197 GB), with a CPU utilization of around 90%. The 29 `TmpltBank` jobs are also computationally demanding, averaging 500 seconds of runtime and 99% CPU utilization, while reading about 16 GB of data. Other jobs in the workflow are relatively lightweight, with short runtimes, low I/O demands, and minimal CPU usage. A notable characteristic of the LIGO workflow is its significant input data read volume, which is concentrated in a small subset of computationally intensive jobs [53].

1.5.5 Epigenomics

The Epigenomics workflow maps the epigenetic state of human cells and is structured as a highly pipelined application, where multiple pipelines process independent data chunks in parallel. It begins with DNA sequence data obtained from multiple lanes. These are split into smaller chunks by `fastQSplit`, based on a specified partitioning factor. Each chunk is then filtered using `filterContams` to remove noise and contaminants. The `sol2sanger` utility converts the filtered data into a format compatible with the Maq sequence mapping software, and `fastq2bfq` further transforms the data into binary fastQ format for improved efficiency. Sequence alignment is performed by the `map` utility, which aligns sequences with the reference genome. The alignment results are merged through one or more `mapMerge` stages. Post-merging, `maqIndex` is used to retrieve reads from specific genomic regions (e.g., chromosome 21), and `pileup` reformats the data for graphical display.

The `map` jobs dominate the workflow's computational cost, followed by `pileup` and `maqIndex`, which process the full aligned output. Other stages, such as `fastQSplit` and `sol2sanger`, are lightweight and primarily depend on chunk size. Profiling data from a sample execution that aligned approximately 13 million sequences shows high CPU utilization across most tasks, indicating a CPU-bound workflow. The few exceptions are data conversion tasks like `fastQSplit` and `sol2sanger`, which exhibit lower utilization. Notably, the `pileup` job averages 153% CPU utilization due to parallel execution of the `Maq` tool and the `AWK` extraction utility, both invoked by the same shell script. Since these processes run concurrently on separate cores, `pileup` could be treated as a parallel job requiring two cores. However, as there is only one `pileup` job per workflow, this has minimal impact on overall performance. Approximately 96% of the total runtime is spent in `map` jobs, highlighting them as the primary target for optimization efforts in improving workflow efficiency [54].

By leveraging these representative workflows, the effectiveness, scalability, and robustness of the proposed algorithm can be validated across different workload types and execution scenarios.

1.6 Metrics for Performance Analysis

Performance analysis is crucial for evaluating the effectiveness of task scheduling algorithms in cloud computing environments. It involves measuring various metrics, assessing multi-objective trade-offs, applying statistical validation, and understanding computational efficiency.

1.6.1 Evaluation Metrics

Evaluation metrics are critical for assessing the effectiveness and efficiency of scheduling algorithms in cloud computing. The following metrics are commonly used to measure various aspects of scheduling performance:

Makespan: It is the total time required to complete the execution of all scheduled tasks. Minimizing makespan improves overall system throughput and reduces waiting time [55]. To measure the makespan of a set of independent tasks, we have used Equation 1.8 and that of workflows, we used the Equation 1.16.

Cost: It is the monetary expense incurred by utilizing cloud resources for task execution [56]. Cost-effective scheduling aims to minimize this value while meeting user requirements. For measuring the cost of executing task and workflow, we used Equation 1.21 and 1.22 respectively.

Energy Consumption: It is the amount of energy consumed by cloud data centers during task execution [57]. Energy-efficient scheduling reduces operational costs and environmental impact. To measure the energy consumption incurred during the workflow execution, we used Equation 1.25.

Resource Utilization: It is the degree to which available computational resources are effectively employed. High resource utilization indicates efficient scheduling. For measuring resource utilization during execution, we used the Equation 1.26.

Deadline Adherence: It is the ability of the scheduling algorithm to complete tasks within user-specified deadlines, ensuring QoS. We have measured this in terms of percentage.

Budget Adherence: It is the adherence to financial constraints set by users for executing tasks on cloud resources. We have measured this in terms of percentage.

These metrics provide a comprehensive framework to evaluate and compare different scheduling strategies under various scenarios.

1.6.2 Pareto Optimality Metrics

In multi-objective optimization for cloud scheduling, Pareto optimality metrics evaluate how well the solutions balance conflicting objectives. These metrics help identify non-dominated solutions that provide trade-offs among objectives like makespan, cost, and energy consumption.

S-Metric: Also known as spread metric, commonly referred to as the S-metric, is utilized to evaluate the distribution uniformity among a set of obtained solutions in a multi-objective

optimization context. A lower S-metric value signifies a more uniform distribution, which in turn reflects higher solution quality [58].The S-metric is formally defined in Equation 1.27:

$$S = \sqrt{\frac{1}{N_p} \sum_{i=1}^{N_p} (d_i - \bar{d})^2}, \tag{1.27}$$

where N_p denotes the total number of non-dominated solutions, d_i represents the distance to the nearest neighbor in the objective space, and $\bar{d}$ is the average of these distances, computed as Equation 1.28:

$$\bar{d} = \frac{1}{N_p} \sum_{i=1}^{N_p} d_i. \tag{1.28}$$

Hypervolume: This metric quantifies the volume of the objective space dominated by the Pareto front solutions relative to a reference point. A higher hypervolume indicates better coverage of the Pareto front.

Dominance Metric: The dominance metric quantifies the extent to which one set of solutions outperforms another in terms of Pareto dominance. As defined in Equation 1.29, $D(A, B)$ represents the proportion of solutions in set B that are dominated by at least one solution in set A. A higher value of $D(A, B)$ indicates a greater dominance of set A over set B.

$$D(A, B) = \frac{|\{b \in B \mid \exists a \in A,\, a \geq b\}|}{|B|} \tag{1.29}$$

These metrics provide a comprehensive evaluation of the Pareto front's quality, balancing coverage and diversity of solutions in the solution space in multi-objective scheduling problems.

1.6.3 Statistical Analysis

Statistical analysis is essential to validate the significance and reliability of the results obtained from scheduling algorithms. It helps in comparing different approaches and ensuring that observed differences are not due to random variation.

T-test: The t-test is a statistical technique used to determine whether the difference between the means of two groups is statistically significant. It is based on the null hypothesis (H_0), which posits that there is no meaningful difference between the group means. The test involves computing the t-statistic and corresponding p-value to assess this assumption. If the absolute value of the t-statistic exceeds a critical threshold from the t-distribution, or if the p-value is less than the predefined significance level, the null hypothesis is rejected. In this study, a significance level of 0.05 was adopted.

ANOVA (Analysis of Variance): ANOVA is a statistical method used to evaluate whether there are significant differences among the means of two or more independent groups. The test is based on a null hypothesis (H_0) that assumes no significant difference exists between group

means. ANOVA employs two key metrics: the F-statistic and the F-critical value. The decision to reject the null hypothesis is made by comparing the computed F-statistic to the corresponding F-critical value. If the F-statistic exceeds the F-critical threshold, the null hypothesis is rejected, indicating statistically significant differences among the group means. In this study, ANOVA was applied to compare the performance of the proposed scheduling algorithm with state-of-the-art methods across various scientific workflows.

By incorporating these statistical methods, we can assess the effectiveness of proposed scheduling techniques and validate their experimental findings.

1.6.4 Computational Complexity

Computational complexity analysis evaluates the efficiency of scheduling algorithms in terms of time and space requirements. It helps in understanding the scalability and practicality of the proposed methods for real-world cloud computing environments.

The complexity is often expressed using Big O notation, which describes how the algorithm's runtime or memory usage grows with respect to the input size, such as the number of tasks or resources.

Efficient scheduling algorithms should balance optimality and computational cost to ensure timely task execution without excessive overhead, especially in large-scale cloud systems.

1.7 Contribution of the Thesis

This thesis presents several notable contributions to the advancement of task scheduling in cloud:

1. **Comprehensive Survey on Task Scheduling in Cloud Environment:** A comprehensive and structured review has been carried out, analyzing a wide range of task scheduling approaches within cloud computing environments. This includes classical heuristic methods such as HEFT and Round Robin, alongside various metaheuristic strategies encompassing evolutionary algorithms, physics-inspired models, bio-inspired techniques, and mathematics-driven methods. The review further explores different hybrid configurations, including heuristic-heuristic, heuristic-metaheuristic, and metaheuristic-metaheuristic combinations. In addition to traditional optimization techniques, the study also incorporates learning-based approaches, offering a holistic overview of state-of-the-art scheduling strategies for cloud task scheduling.

2. **Deadline and Budget - Constrained Archimedes Optimization Algorithm for Workflow Scheduling in Cloud :** This work presents a novel multi-objective metaheuristic, the Deadline and Budget Constrained Archimedes Optimization Algorithm (ADB), designed to address workflow scheduling under strict deadline and budget constraints. A new technique is introduced to dynamically determine feasible deadline and

budget intervals, complemented by a fitness function that integrates these constraints while favoring solutions with lower makespan and cost.

3. **MLPOA: Multi-Layer Perceptron Optimized Archimedes Approach for Workflow Scheduling in Cloud:** This study introduces an advanced hybrid metaheuristic scheduling framework that enhances the Archimedes Optimization Algorithm (AOA) by incorporating a ML-based dynamic parameter tuning mechanism. Specifically, a Multi-Layer Perceptron (MLP) regressor is employed to model the relationship between AOA parameters and scheduling performance, facilitating automated parameter optimization and reducing reliance on manual configuration. A constraint-aware fitness function is designed to simultaneously minimize makespan, cost, and energy consumption while strictly adhering to deadline and budget requirements.

4. **Multi-Objective Workflow Scheduling in Cloud using Archimedes Optimization Algorithm:** This work proposes the Modified Local Escaping Archimedes Optimization (MLEAO) Algorithm, a novel multi-objective metaheuristic aimed at enhancing workflow scheduling by overcoming local optima entrapment and simultaneously optimizing makespan and cost. The algorithm leverages the Heterogenous Earliest Finish Time (HEFT) scheduling strategy for initialization, providing a head start toward solutions with favorable makespan. A multi-objective fitness function is employed to guide the search process, balancing improvements in both makespan and cost. A local escaping operation has also been implemented to avoid local optima entrapment.

5. **MRFOBL : An Energy-Efficient Opposition-Based Learning Enhanced Manta Ray Foraging Approach for Workflow Scheduling in Cloud:** This study presents a hybrid workflow scheduling algorithm that synergistically combines Manta Ray Foraging Optimization (MRFO), multiple Opposition-Based Learning (OBL) strategies, and a Local Escaping Operator (LEO) to enable energy-efficient scheduling in cloud computing environments. The algorithm begins with a hybrid HEFT-OBL initialization to generate high-quality starting solutions, while diverse OBL variants are integrated to maintain population diversity and mitigate premature convergence. The inclusion of LEO further enhances the algorithm's ability to escape local optima by applying adaptive perturbations, particularly under stringent constraints. A carefully designed multi-objective fitness function guides the search toward feasible and energy-efficient solutions without sacrificing other key performance criteria such as makespan and cost, while adhering to user-defined deadline and budget limits.

6. **A2CS: Advantage Actor-Critic Strategy for Task Scheduling in Cloud:** This research introduces a RL-based scheduling framework leveraging the Advantage Actor-Critic Strategy (A2CS) to optimize task execution in cloud computing environments. A tailored reward and advantage function is designed to promote effective decision-making by emphasizing makespan reduction and overall scheduling efficiency.

These contributions collectively address major challenges in task scheduling in the cloud, such

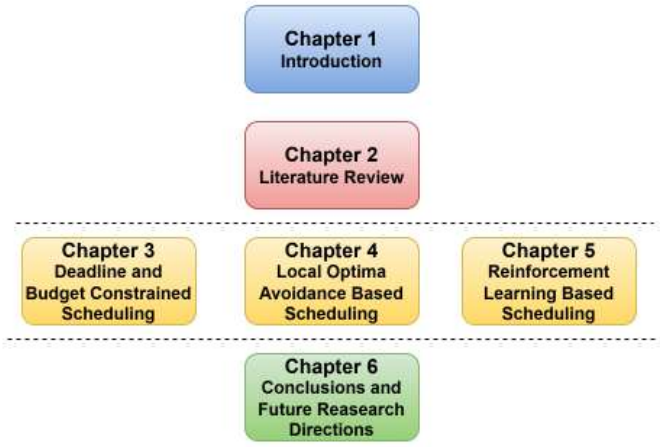


Figure 1.4: Structure of the thesis and outline of key chapters for task scheduling in cloud

as resource heterogeneity, multi-objective optimization, task dependencies, deadline and budget constraints, local optima entrapment, and energy efficiency.

1.8 Thesis Organization

This thesis systematically explores and addresses various challenges in task scheduling in the cloud. Chapter 2 is the Literature review. Chapters 3 to 5 are grounded in published research and build progressively to offer methodological innovations and experimental validations. Chapter 6 presents the conclusions and future research directions. Figure 1.4 presents the structural flow and thematic continuity of the thesis. The contents of the chapters are summarized below:

- **Chapter 2: Literature Review**
  A thorough review was conducted on task scheduling techniques in cloud environments, covering classical heuristics like HEFT and Round Robin, various metaheuristics, hybrid methods, and learning-based approaches. This study provides a broad overview of traditional and modern strategies for effective cloud scheduling.

- **Chapter 3: Deadline and Budget-Constrained Scheduling**
  This chapter explores advanced metaheuristic approaches for workflow scheduling in cloud environments under deadline and budget constraints. It first presents the Deadline and Budget-constrained Archimedes Optimization Algorithm, which dynamically determines feasible constraint intervals and optimizes makespan and cost. Subsequently, the chapter introduces the MLPOA framework, a hybrid method that enhances ADB with ML-based parameter tuning using a Multi-Layer Perceptron, further improving scheduling efficiency while maintaining strict adherence to constraints. This chapter is an extension of two research articles:

  1. Shweta Kushwaha & Ravi Shankar Singh (2025). *Deadline and budget-constrained archimedes optimization algorithm for workflow scheduling in cloud. Cluster Com-*

puting [SCIE], 28(2), 117. DOI: 10.1007/s10586-024-04702-1

2. Shweta Kushwaha, Ravi Shankar Singh, Janhavi Verma & Sanskruti Kolgane (2025). *MLPOA : Multi-Layer Perceptron Optimized Archimedes Approach for Workflow Scheduling in Cloud. 32nd IEEE International Conference on High Performance Computing.* (Under Review)

- **Chapter 4: Local Optima Avoidance-Based Scheduling**
  This chapter presents advanced multi-objective metaheuristic algorithms for workflow scheduling in cloud environments. It begins with the Modified Local Escaping Archimedes Optimization algorithm, which enhances scheduling by overcoming local optima and balancing makespan and cost using HEFT-based initialization and a local escaping mechanism. Following this, the chapter introduces MRFOBL, a hybrid approach integrating Manta Ray Foraging Optimization with Opposition-Based Learning and local escaping strategies to achieve energy-efficient, deadline and budget-constrained scheduling while maintaining solution diversity and robustness. This chapter is an extension of two research articles:
  1. Shweta Kushwaha, Ravi Shankar Singh & Kanika Prajapati (2025). *Multi-Objective Workflow Scheduling in Cloud Using Archimedes Optimization Algorithm. Concurrency and Computation: Practice and Experience [SCIE], 37(4-5), e8393.* DOI: 10.1002/cpe.8393
  2. Shweta Kushwaha, Ravi Shankar Singh & Shri Prakash Dwivedi (2025). *MRFOBL: An Energy-Efficient Opposition-Based Learning Enhanced Manta Ray Foraging Approach for Workflow Scheduling in Cloud. IEEE Transactions on Sustainable Computing [SCIE].* (Under Review)

- **Chapter 5: Reinforcement Learning-Based Scheduling**
  This chapter presents a RL-based scheduling framework utilizing the Advantage Actor-Critic Strategy to enhance task execution in cloud computing. It features a customised reward and advantage function that prioritizes minimizing makespan and improving overall scheduling performance. This chapter is an extension of a research article.
  1. Shweta Kushwaha & Ravi Shankar Singh (2025). *A2CS: Advantage Actor-Critic Strategy for Task Scheduling in Cloud. 12th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering.* (Under Review)

- **Chapter 6: Conclusions and Future Research Directions**
  The final chapter summarizes the key contributions and insights of the thesis. It critically assesses the performance and impact of the proposed approaches and outlines promising future research avenues.

This structured organization facilitates a logical progression from identifying research gaps to proposing, implementing, and evaluating innovative solutions for task scheduling in the cloud.