Deadline and Reliability Constrained Workflow Scheduling via Analytical Replica Estimation

as part of Research Internship

Maurya Samanta
UG-4,
Department of Production Engineering,
Jadavpur University

under the supervision of
Dr. Rajib Das
Department of Computer Science and Engineering,
University of Calcutta

Contents

List of Figures

List of Tables

Introduction

Cloud computing refers to the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the internet. Since the early 21st century, it has experienced rapid growth, becoming a foundational technology for supporting large-scale applications targeting global markets. Cloud platforms offer unparalleled scalability, flexibility, and cost-efficiency, making them ideal for organizations seeking to offload infrastructure management and focus on application-level concerns.

In the Infrastructure-as-a-Service (IaaS) model, cloud providers offer a wide variety of virtual machines (VMs) with different performance capabilities and pricing structures. Users can rent these VMs on a pay-as-you-go basis, meaning they are charged only for the resources consumed during execution. This model is especially suitable for applications such as scientific workflows, where computational tasks can be mapped onto cloud resources without requiring upfront infrastructure investment.

High-performance VMs, though more expensive, can execute tasks faster than their lower-performance counterparts. Therefore, users must often strike a trade-off between execution time and monetary cost. A common objective is to complete a given workflow within a user-defined deadline while minimizing the total cost incurred. This leads to a fundamental optimization problem in workflow scheduling, where the selection of VM types and task assignments must balance speed and cost.

However, cloud resources are not perfectly reliable. Failures may occur due to hardware faults, network issues, or transient errors. As a result, users may not only care about cost and execution time but also about the probability of successful completion, i.e., the reliability of their workflows.

This introduces an additional dimension to the scheduling problem: ensuring cost-effective execution within the deadline, while maintaining a target reliability level. Designing algorithms that satisfy all three constraints—cost, deadline, and reliability—is a complex but essential challenge in cloud workflow management.

Problem Statement

A scientific workflow consists of a set of tasks $\{t_1, t_2, t_3, \ldots, t_n\}$, where dependencies exist between tasks. Specifically, a task t_x is said to depend on a task t_y if the input resources of t_x are a subset of the output resources of t_y. Thus, the workflow can be mathematically modeled as a Directed Acyclic Graph (DAG). Let T be the set of all tasks, and let $E \subseteq T \times T$ be the set of directed edges that represent the dependencies. Each edge $e_{i,j} = (t_i, t_j) \in E$ represents a precedence constraint, indicating that task t_j can start execution only after task t_i has completed. For any task t_i, the set of its immediate predecessors is denoted by $\text{parent}(t_i)$, and the set of its immediate successors is denoted by $\text{child}(t_i)$. A task with no parent, i.e., $\text{parent}(t_i) = \varnothing$, is called an entry task. A task with no child, i.e., $\text{child}(t_i) = \varnothing$, is called an exit task. Each task t_i is associated with a set of input files, denoted by $\text{inputfiles}(t_i)$, a set of output files, $\text{outputfiles}(t_i)$, and an expected runtime. The files shared between tasks are assumed to have defined storage sizes, typically measured in bytes.

A set of virtual machines $VM = \{\text{VM}_1, \text{VM}_2, \text{VM}_3, \ldots, \text{VM}_n\}$ is provisioned to clients by Infrastructure-as-a-Service (IaaS) providers. Each virtual machine is characterized by a unique identifier, a fixed storage capacity (in gigabytes), a number of virtual CPUs (vCPUs), network bandwidth (in megabytes per second), and an associated on-demand cost (in USD per hour). The configuration of a VM directly influences its cost; that is, machines with higher computational capabilities incur higher charges. The computational power of a VM is typically expressed in terms of Million Instructions Per Second (MIPS). For a given virtual machine VM_i, the MIPS value is calculated as shown in Equation (2.1):

$$\text{MIPS}(\text{VM}_i) = \text{vCPU}_i \times \text{ClockSpeed}_i \times 1000 \tag{2.1}$$

where vCPU_i is the number of virtual CPUs allocated to VM_i, and ClockSpeed_i is the clock frequency (in GHz) of each vCPU.

Each virtual machine VM_i is associated with a fault probability, denoted as F_i, which quantifies the likelihood of failure during task execution. This probability is modeled as an exponential decay function with respect to execution time, given by Equation (2.2):

$$F_i = e^{-\lambda t} \tag{2.2}$$

where λ is a constant representing the failure rate, and t is the runtime of a specific task on

VM$_i$.

To mitigate the risk of failure, tasks can be replicated across multiple virtual machines with similar performance characteristics. Let x_i denote the number of such replications. The net fault probability after replication becomes:

$$F_{net} = F_i^{x_i} \tag{2.3}$$

Consequently, the success probability of the task (i.e., the probability that at least one replica completes successfully) is given by:

$$S_i = 1 - F_i^{x_i} \tag{2.4}$$

It is assumed that the VMs are totally responsible for running the workflows as well storing the input/output files. It is assumed that any number of VM instances can be launched from VM set i.e. there is no restriction on the leasing of the number of VM instances. Also, Cloud computing follows the pay-per-use model which means users are being charged for whole time duration even if they use only a fraction it. Thus, in the proposed model, each instance of leased VM is charged per second time interval. Bandwidth costs are involved in transferring files from one VM to another.

The objective of this work is to determine an optimal mapping of the task set T of a workflow onto a set of virtual machine instances derived from a given set of VM configurations (VM$_{\text{set}}$), such that both the total price for execution (TPE) and the total execution time (TET) are minimized. This must be achieved while adhering to the user-defined deadline, satisfying the precedence constraints among tasks, and ensuring the required reliability of the workflow.

$$\begin{aligned} \text{Minimize:} \quad & \text{TPE} = \sum_i c_i t_i x_i \\ \text{Subject to:} \quad & \prod_i S_i \geq R_w \\ & \sum_i t_i \leq D \end{aligned} \tag{2.5}$$

where:

- c_i: Cost of executing the task on a VM of type i,
- t_i: Runtime of the task on VM type i,

- x_i: Number of replications of the task on VM type i,
- S_i: Success probability of the task after replication,
- R_w: Required reliability of the entire workflow,
- D: User-defined deadline.

Solution

The proposed algorithm determines the most cost-effective schedule for executing a workflow within a user-defined deadline, while satisfying a specified reliability constraint. The solution employs Lagrange Multiplier Optimization to solve the constrained optimization problem described in Equation (2.5). The algorithm consists of two main components: the **VM Allocation Algorithm** and the **Task Replication Algorithm**.

VM Allocation Algorithm

Table 1 presents the key notations used in this work. Below are the definitions and mathematical formulations of important scheduling metrics:

- **Earliest Start Time (EST$_{k,i}$)**: The earliest time at which task t_i can begin execution on VM k, given that all parent tasks have finished and data has been transferred.

$$\mathrm{EST}_{k,i} = \max_{t_j \in \mathrm{parent}(t_i)} \left(\mathrm{EFT}_j + \mathrm{CD}_{j,i}\right) \tag{3.1}$$

- **Earliest Finish Time (EFT$_{k,i}$)**: The time at which task t_i will complete execution on VM k, assuming it starts at EST$_{k,i}$.

$$\mathrm{EFT}_{k,i} = \mathrm{EST}_{k,i} + \frac{\mathrm{MI}(t_i)}{\mathrm{MIPS}(\mathrm{VM}_k)} \tag{3.2}$$

- **Communication Delay (CD$_{i,j}$)**: The time required to transfer data between tasks t_i and t_j if scheduled on different VMs.

$$\mathrm{CD}_{i,j} = \begin{cases} \dfrac{\mathrm{data\_size}_{i,j}}{\max(\mathrm{bandwidth}(\mathrm{VM}_i),\ \mathrm{bandwidth}(\mathrm{VM}_j))}, & \text{if } \mathrm{VM}(t_i) \neq \mathrm{VM}(t_j) \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

- **Upward Rank** ($r_{u,i}$): The length of the longest path from task t_i to an exit task in the workflow DAG, communication delay.

$$r_{u,i} = \mathrm{runtime}(t_i) + \max_{t_j \in \mathrm{child}(t_i)} \left(\overline{CD}_{i,j} + runtime(t_j), r_{u,j}\right) \tag{3.4}$$

- **Sub-deadline** (D_i): The sub-deadline of a task t_i is defined as:

$$D_i = \frac{r_{u,i}}{r_{u,\text{entry}}} \times D \tag{3.5}$$

- **Cost** ($C_{k,i}$): The monetary cost of executing task t_i on virtual machine VM_k.

$$C_{k,i} = \frac{\text{MI}(t_i)}{\text{MIPS}(\text{VM}_k)} \times \left(\frac{\text{price}(\text{VM}_k)}{3600}\right) \tag{3.6}$$

Table 1: Notations Used in the VM Allocation Algorithm

Symbol	Description
D	User-defined deadline in miliseconds
$\text{EST}_{k,i}$	Earliest start time of t_i on VM k
$\text{EFT}_{k,i}$	Earliest finish time of t_i on VM k
$\text{CD}_{i,j}$	Communication delay between t_i and t_j
MI_i	Million Instructions of task t_i
$MIPS_k$	Million Instructions Per Second of VM VM_k
$datasize_{i,j}$	Total data to be transferred from task t_i to task t_j in MB
$child(t_i)$	Set of all dependance tasks(children) of task t_i
$r_{u,i}$	Upward Rank of task t_i
$C_{k,i}$	Cost of executing t_i on VM k in USD
$price(VM_k)$	On-demand pricing of VM VM_k in USD/hour

Algorithm 1 describes the high-level flow of the scheduler algorithm. After some initialization, the upward ranks of the tasks are calculated using Equation (3.4). Subsequently, sub-deadlines for each task are estimated using Equation (3.5). For each task, a set of eligible VMs is prepared. The eligibility of VMs is determined based on two criteria: the total storage required by the task (sum of input and output file sizes), and the runtime of the task on the VM compared to its assigned sub-deadline. Algorithm 2 presents the logic used for estimating eligible virtual machines for a task.

By estimating eligible virtual machines (VMs) in advance, the scheduling process is significantly optimized. Instead of evaluating every task on all available VMs, the algorithm restricts scheduling decisions to only those VMs that meet the task's resource requirements. This pre-filtering step not only reduces computational overhead but also contributes to cost efficiency, as it ensures that only feasible and capacity-sufficient VMs are considered during allocation.

Algorithm 1 Workflow Scheduling Main Procedure

```
Require: DAG workflow in XML format containing $n$ tasks
Require: VM dataset with $k$ virtual machines
Ensure: A cost-optimized, deadline-aware task schedule with reliability
1: taskMap ← parseXML()
2: topoSortedJobs ← topologicalSort(taskMap)
3: upwardRanks ← calculateUpwardRanks(topoSortedJobs) according to Equation (3.4)
4: assignSubdeadlines(upwardRanks) according to Equation (3.5)
5: getEligibleVMs(topoSortedJobs, vms)
6: totalCost ← schedule(topoSortedJobs)
7: Display(totalCost)
```

Algorithm 2 Eligible VM Estimation

```
Require: List of tasks sorted according to upward ranks
Require: List of available VMs
Ensure: Estimate eligible VMs for each task
1: for each task $j$ in tasks do
2:     Initialize eligibleVMs$_j$ ← [ ]
3:     Compute requiredStorage from input/output files
4:     for each VM $v$ in VMs do
5:         Compute runtime ← $\frac{\text{MI}(j)}{\text{MIPS}(v)}$
6:         if runtime $<$ subDeadline$(j)$ and requiredStorage fits in $v$ then
7:             Add $v$ to eligibleVMs$_j$
8:     Assign eligibleVMs$_j$ to job $j$
```

Algorithm 3 outlines the core scheduling logic. For each virtual machine (VM), a tracking map is maintained to record the tasks assigned to it along with their respective start and finish times. This information is later used to compute the total cost of utilizing that VM.

For every task in the `UpwardRankSortedJobs` list, initial values for minimum cost, earliest start time (EST), and earliest finish time (EFT) are set to zero. The algorithm then iterates over the pre-estimated eligible VMs for the current task. Each task is tentatively scheduled on all its eligible VMs to evaluate the associated execution cost. The VM offering the lowest cost is selected for the task, and the VM-task schedule map is updated accordingly. During scheduling, the bandwidth cost is taken care of when the parent of the present task has been executed on a different Virtual Machine.

Algorithm 3 Task Scheduling with Cost Minimization

```
Require: Sorted task list by upward rank $\mathscr{J}$, list of VMs $\mathscr{V}$, VM availability map, file size
    and dependency maps
 1: Initialize totalCost $\leftarrow 0$
 2: Initialize schedule list $\mathscr{S} \leftarrow [\,]$
 3: Initialize vmTaskMap for each VM $\in \mathscr{A}$ with empty list
 4: for each task $t \in \mathscr{J}$ do
 5:     minCost $\leftarrow \infty$, bestVM $\leftarrow$ null
 6:     bestEST $\leftarrow 0$, bestEFT $\leftarrow 0$
 7:     for each VM $v \in \text{eligibleVMs}(t)$ do
 8:         est $\leftarrow \text{vmAvailability}[v]$
 9:         for each parent task $p \in \text{parents}(t)$ do
10:             if $p$ is scheduled then
11:                 commTime $\leftarrow 0$
12:                 if $\text{assignedVM}(p) \neq v$ then
13:                     Compute data size from file map
14:                     Compute bandwidth of $v$
15:                     commTime $\leftarrow$ dataSize/bandwidth using Equation (3.3)
16:                 est $\leftarrow \max(\text{est}, \text{endTime}(p) + \text{commTime})$ using Equation (3.1)
17:         Compute runtime $\leftarrow \text{MI}(t)/\text{MIPS}(v)$
18:         Compute EFT $\leftarrow \text{est} + \text{runtime}$ using Equation (3.2)
19:         if EFT $\leq \text{subDeadline}(t)$ then
20:             Compute billing-based cost of execution on VM $v$
21:             Compute normalized delay bias
22:             if cost is minimal or delay bias is better then
23:                 Update $\text{minCost}, \text{bestVM}, \text{bestEST}, \text{bestEFT}$
24:     Assign $t$ to bestVM, update scheduling records and costs
25: return Final schedule and total cost
```

Task Replication Algorithm

Nature of the Optimization Problem

The objective function

$$\sum_i c_i t_i x_i$$

is linear in the decision variables x_i. However, the reliability constraint

$$\prod_i S_i = \prod_i (1 - f_i^{x_i}) \geq R_w$$

is non-linear and multiplicative, since each success probability depends exponentially on x_i. This product of non-linear functions introduces non-linearity into the constraint space.

Therefore, the optimization problem becomes non-linear due to the following reasons:

- The presence of exponential terms $f_i^{x_i}$ in the success probability expressions.
- The multiplicative form of the reliability constraint $\prod_i (1 - f_i^{x_i})$, which prevents direct linear optimization.
- The use of logarithmic transformations required to linearize or approximate the constraint.

Use of Lagrangian Optimization

Lagrangian optimization is a powerful method to solve constrained optimization problems, particularly when:

- The objective function and/or constraints are non-linear.
- Direct analytical or linear programming solutions are not feasible due to coupling between decision variables.

Lagrangian Formulation for Replica Count Optimization

We aim to minimize the total price of execution (TPE) given by:

$$\text{TPE} = \sum_i c_i t_i x_i$$

subject to the reliability constraint:

$$\prod_i (1 - f_i^{x_i}) \geq R_w$$

This is a non-linear optimization problem due to the multiplicative nature of the reliability constraint. To make it tractable, we introduce a change of variables:

$$y_i = f_i^{x_i} \quad \Rightarrow \quad x_i = \frac{\ln y_i}{\ln f_i}$$

This transforms the reliability constraint into:

$$\prod_i (1 - y_i) \geq R_w \quad \Rightarrow \quad \sum_i \ln(1 - y_i) \geq \ln R_w$$

Assuming small values of y_i, we approximate using the first-order Taylor expansion:

$$\ln(1 - y_i) \approx -y_i \quad \Rightarrow \quad \sum_i y_i \leq \ln\left(\frac{1}{R_w}\right)$$

Now, substitute the transformed variable into the cost function:

$$\text{TPE} = \sum_i c_i t_i x_i = \sum_i A_i \ln y_i \quad \text{where} \quad A_i = \frac{c_i t_i}{\ln f_i}$$

To solve the constrained minimization, we use the method of Lagrange multipliers. The Lagrangian is:

$$\mathscr{L}(y_1, y_2, \ldots, y_n, \lambda) = \sum_i A_i \ln y_i + \lambda \left(\sum_i y_i - \ln\left(\frac{1}{R_w}\right)\right)$$

We now compute the partial derivatives with respect to each y_i:

$$\frac{\partial \mathscr{L}}{\partial y_i} = \frac{A_i}{y_i} + \lambda$$

Setting the derivative to zero for optimality:

$$\frac{A_i}{y_i} + \lambda = 0 \quad \Rightarrow \quad y_i = -\frac{A_i}{\lambda}$$

This gives the expression of y_i in terms of the Lagrange multiplier λ. The corresponding values of x_i are then computed in code using this gradient-based logic.

Algorithm 4 presents the optimization of replica counts x_i for each task in a workflow, aiming to minimize the Total Price of Execution (TPE) under a global reliability constraint. It begins with a reliability-based estimation of the minimum number of replicas using exponential failure modeling for each task. Subsequently, it applies gradient-based updates combined with a penalty method to iteratively refine the replica counts. At each step, the gradients of cost and reliability are evaluated and balanced to ensure cost efficiency while satisfying the reliability constraint. This process continues until the overall system reliability requirement is fulfilled.

Algorithm 4 OptimizeReplicaCounts: Replica Optimization under Reliability Constraint

```
Require: List of Jobs jobs, List of VMs vms, Failure rate lambda, Deadline deadline, Required system reliability requiredReliability
Ensure: Replica counts $x[i]$ for each job $i$
 1: Initialize array $x[i] \leftarrow 1.0$ for all jobs
 2: for each job $i$ in jobs do
 3:     if no eligible VMs for job $i$ then
 4:         Set $x[i] \leftarrow 1.0$ and continue
 5:     Determine fastest MIPS and cheapest cost from eligible VMs
 6:     Compute runtime $t_i \leftarrow \frac{\texttt{job.mi}}{\texttt{fastestMIPS}}$
 7:     Compute task reliability $r_i \leftarrow e^{-\lambda \cdot t_i}$
 8:     Set target reliability per task $r_{\text{target}} \leftarrow \texttt{requiredReliability}^{1/n}$
 9:     if $r_i \geq r_{\text{target}}$ then
10:         $x[i] \leftarrow 1.0$
11:     else
12:         Estimate minimum replicas:
                $$x[i] \leftarrow \left\lceil \frac{\ln(1 - r_{\text{target}})}{\ln(1 - r_i)} \right\rceil$$
13: Evaluate initial system reliability
14: if initial reliability $\geq$ requiredReliability then
15:     return $x$
16: Initialize: mu (penalty multiplier), learningRate, muLearningRate, maxIters
17: Precompute fastestMIPS and cheapestCosts per job from eligible VMs
18: for iteration = 1 to maxIters do
19:     Compute current total cost and system reliability
20:     Calculate constraint violation: requiredReliability − systemReliability
21:     if violation $> \varepsilon$ then
22:         for each job $i$ do
23:             Compute runtime and failure rate $f_i$
24:             Compute cost gradient: cheapestCosts$[i]$ · runtime
25:             Compute reliability gradient:
                    $$\frac{\partial R_{\text{sys}}}{\partial x[i]} \approx R_{\text{sys}} \cdot f_i^{x[i]} \cdot \ln(f_i)/(1 - f_i^{x[i]})$$
26:             Combine gradients to compute total gradient:
                    totalGradient ← costGradient − penalty · reliabilityGradient
27:             Update:
                    $x[i] \leftarrow \max(1.0, \min(10.0, x[i] - \texttt{learningRate} \cdot \texttt{totalGradient}))$
28:         Update penalty multiplier: mu ← mu + muLearningRate · violation
29:     else
30:         Break (constraints satisfied)
31: return final array $x$
```

By employing this algorithm, the number of replicas required for each task is determined in advance. This proactive approach significantly reduces the overall execution cost, as it avoids the need to perform trial-and-error replication during the actual scheduling phase. Consequently, the optimal number of replicas can be estimated without executing tasks, leading to more efficient scheduling decisions.

The calculated replica counts are leveraged within the scheduler function to replicate tasks onto additional VMs selected from each task's `eligibleVMs` list, as outlined in Algorithm 5. Replica assignment follows a similar strategy as primary task allocation—considering VM availability and data transfer delays to maintain execution feasibility. It is assumed that replicas are launched in parallel, provided that the selected VMs are available at the moment of scheduling. This parallelism ensures that the makespan of the overall workflow is minimally impacted, while enhancing reliability through redundancy.

To simulate the probabilistic nature of task execution on cloud VMs, a random number uniformly distributed in the range [0, 1] is generated for each task execution attempt. This value is compared against the computed reliability of the task on the selected VM. If the reliability is less than or equal to the generated random number, the task is considered to have successfully executed on that VM. Upon the first successful execution, no further replicas are scheduled or executed for that task, as a single successful instance suffices to generate the required output for its dependent tasks.

Conversely, if the execution fails (i.e., the random number is less than the task reliability), subsequent replicas are launched according to the precomputed replica count. If none of the replicas succeed in the current scheduling round, the replication loop is re-entered until at least one successful execution is achieved, thereby ensuring task-level fault tolerance and reliability compliance.

Algorithm 5 Replica Assignment for Scheduled Tasks

```
Require: Replica counts array replicaCounts, Eligible VMs for each task, Pre-scheduled
    task with assigned best VM
Ensure: Replicas are assigned to additional VMs, respecting deadlines and minimizing cost
 1: Let $R_i \leftarrow \texttt{replicaCounts}[i]$
 2: if $R_i < 1$ then
 3:     $R_i \leftarrow 1$
 4: Assign main task to bestVM
 5: Update startTime, endTime, and availability of bestVM
 6: Add main task to schedule
 7: replicasAdded ← 1
 8: Create list candidateReplicas ← eligibleVMs \ {bestVM}
 9: Sort candidateReplicas by ascending $\frac{\texttt{cost}}{\texttt{reliability}}$
10: Initialize RNG with fixed seed
11: successAchieved ← false
12: for each vm in candidateReplicas do
13:     if replicasAdded $\geq R_i$ or successAchieved then
14:         break
15:     Estimate est ← availability[$vm$]
16:     Compute runtime ← $\frac{MI_i}{MIPS_{vm}}$
17:     Compute eft ← est + runtime
18:     if eft > $\texttt{subDeadline}_i$ then
19:         continue
20:     Create a replica copy of task $T_i$
21:     Assign replica to $vm$
22:     Set replica's startTime and endTime
23:     Update availability[$vm$] ← eft
24:     Add replica to schedule and VM-task map
25:     replicasAdded ← replicasAdded + 1
26:     Generate $r \sim \mathscr{U}(0,1)$
27:     Compute reliability $\rho \leftarrow \exp(-\lambda \cdot \texttt{runtime})$
28:     if $\rho \leq r$ then
29:         successAchieved ← true
```

Results

The proposed scheduling and replica optimization algorithm was evaluated using three scientific workflows: *Montage*, *Sipht*, and *Epigenomics*. The virtual machine configurations were sourced from AWS EC2 instance data, ensuring realistic modeling of cost, performance, and network heterogeneity.

The VM dataset includes a range of instance types with varying compute power (measured in MIPS), bandwidth, and storage capacity. The median characteristics of the VM pool used in our simulations are as follows:

- **Median MIPS**: 131871
- **Median Bandwidth**: 52 MBps
- **Median Storage**: 3333 GB

The performance of the proposed approach was analyzed under varying system reliability thresholds and deadline constraints. Key metrics observed include the Total Price of Execution (TPE) and end-to-end workflow reliability.

Simulations were conducted on a local machine configured as follows:

- **Operating System**: Windows 10
- **Processor**: 13th Gen Intel(R) Core(TM) i5-1335U (1.30 GHz)
- **Installed RAM**: 8.00 GB
- **System Type**: 64-bit operating system, x64-based processor

The workflows used in this simulation—*Epigenomics*, *Sipht*, and *Montage*—each consist of 100 tasks. To evaluate the performance of the scheduling and replication strategy under varying temporal constraints, each workflow was executed across a range of 10 distinct deadlines. These deadlines were systematically derived by incrementally increasing the smallest time possible to execute the workflow by 10% at each step.

- **Epigenomics**: This workflow simulates a bioinformatics pipeline used for mapping and analyzing epigenetic markers such as DNA methylation. It involves several data preprocessing and alignment steps, characterized by moderate computation and communication requirements with parallel branches.

- **Sipht**: The Sipht workflow identifies and annotates non-coding RNAs in bacterial genomes. It consists of a large number of small, mostly independent tasks, making it suitable for evaluating task-level parallelism and replica assignment strategies.
- **Montage**: Montage is an astronomy workflow that generates custom science-grade astronomical image mosaics. It is highly I/O-intensive and exhibits a deep dependency structure, useful for evaluating scheduling efficiency under tight deadlines and data transfer constraints.

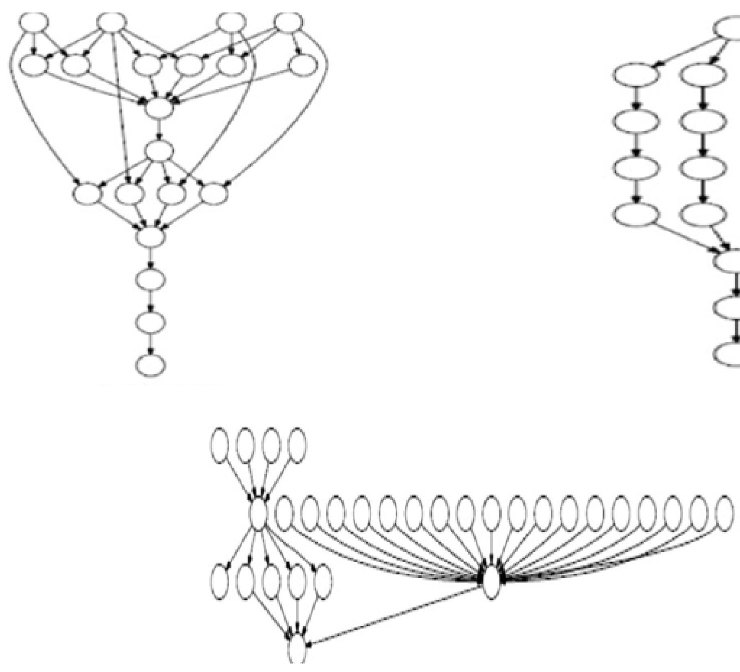
Figure 1: Workflow structures of Montage, Epigenomics and Sipht

Given that the proposed algorithm computes task-level sub-deadlines based on Equation (3.5), it follows that more relaxed (softer) workflow deadlines allow for proportionally larger sub-deadlines for individual tasks. This increased temporal flexibility enables the scheduler to assign tasks to a broader range of virtual machines, including cost-efficient but slower instances.

As a result, the observed trend in the cost-versus-deadline plots reflects a consistent decrease in total execution cost with softer deadlines. This behavior validates the effectiveness of the algorithm in exploiting temporal slack to reduce expenditure while maintaining system reliability.

To evaluate the impact of reliability constraints on execution cost, the three workflows were

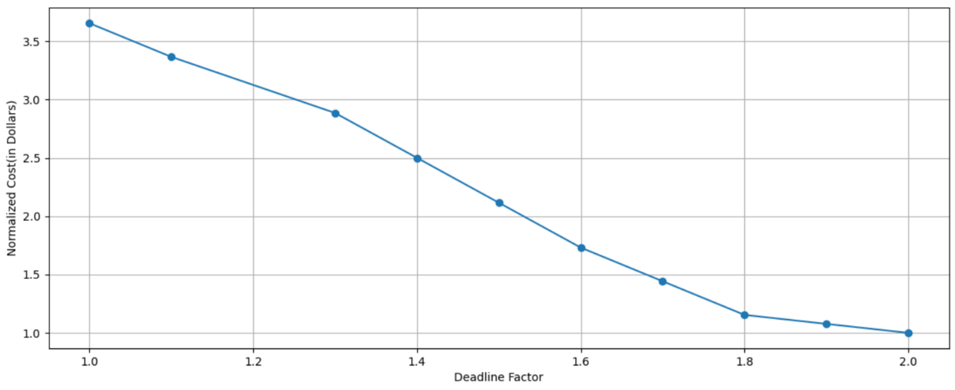

Figure 2: Effect of varying deadlines on total execution cost for the Montage workflow

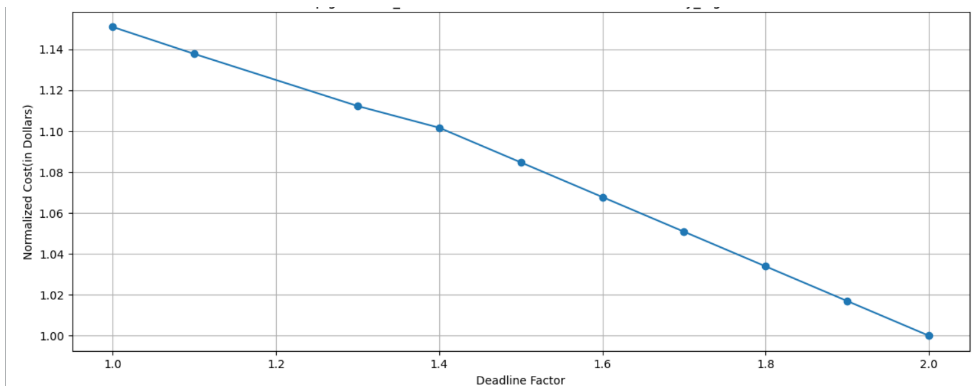

Figure 3: Effect of varying deadlines on total execution cost for the Epigenomics workflow

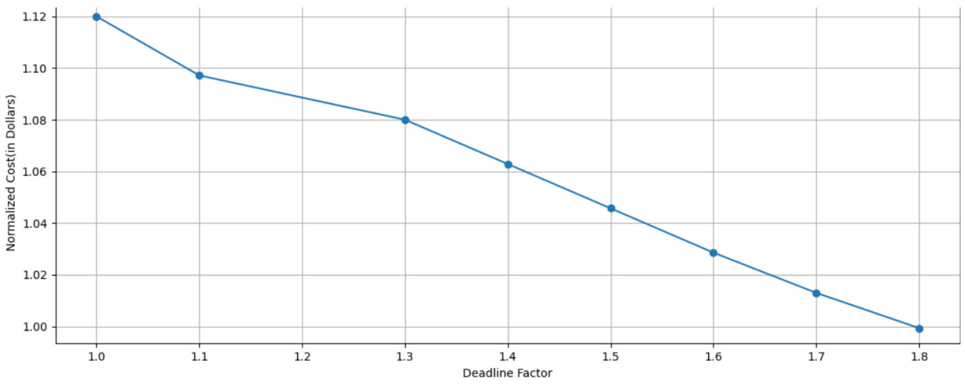

Figure 4: Effect of varying deadlines on total execution cost for the Sipht workflow

simulated at reliability levels of 0.90, 0.95, and 0.99. For each reliability level, the workflows were executed across ten deadline settings, each incrementally relaxed by 10% over the minimum possible makespan for the workflow. The average cost across these executions was recorded and plotted.

A comparative analysis was conducted between the proposed Lagrangian-based replica optimization algorithm and a baseline iterative method. In the baseline approach, replication counts are determined dynamically during scheduling, leading to increased computational overhead and higher execution costs. In contrast, the proposed method estimates replica counts prior to scheduling, allowing for more efficient resource allocation and cost reduction.

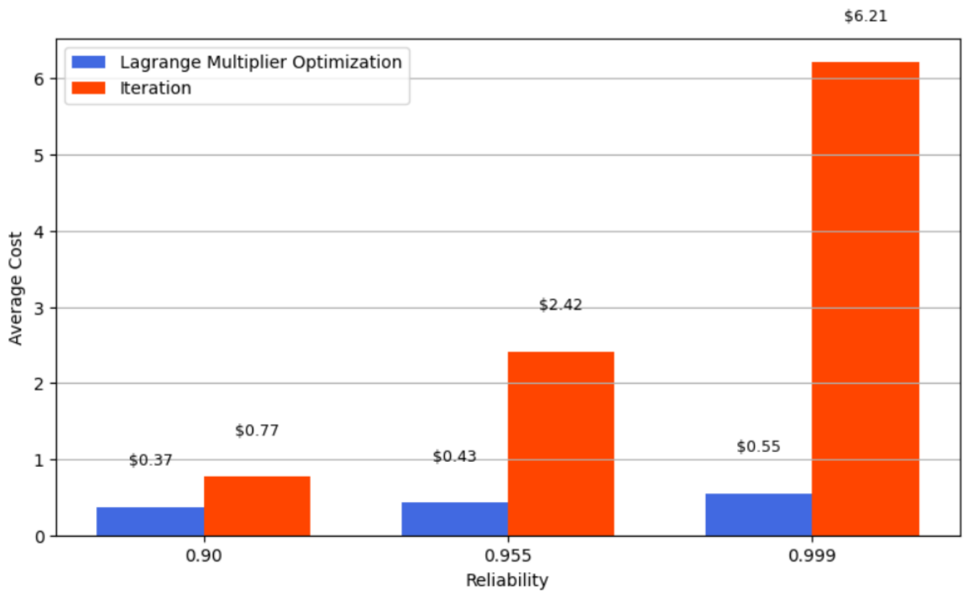

Figure 5: Effect of varying reliability on total execution cost for the Montage workflow using proposed algorithm and iterative approach

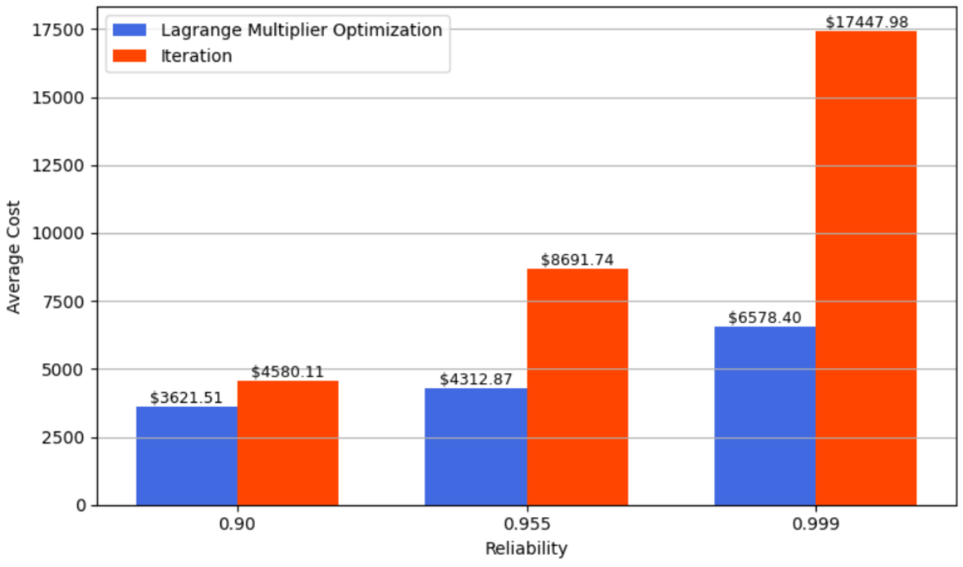

Figure 6: Effect of varying reliability on total execution cost for the Epigenomics workflow using proposed algorithm and iterative approach

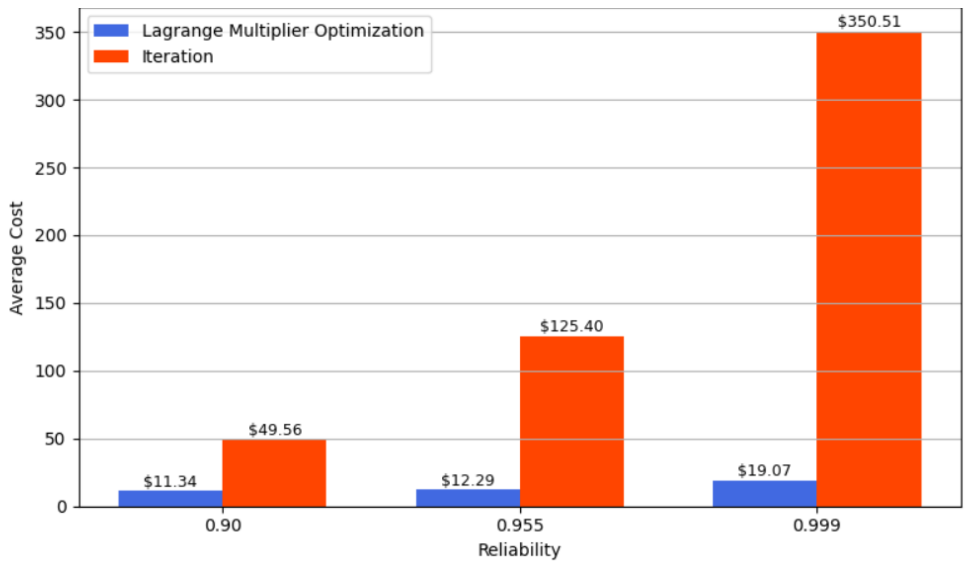

Figure 7: Effect of varying reliability on total execution cost for the Sipht workflow using proposed algorithm and iterative approach

Conclusion

This study presents an efficient and reliability-aware workflow scheduling framework that integrates replica count optimization and cost-effective task assignment under deadline constraints. By leveraging a combination of Lagrangian optimization for estimating optimal replica counts and a heuristic scheduler that considers VM eligibility, availability, and communication delays, the proposed method demonstrates substantial improvements in cost efficiency while meeting stringent reliability targets.

Experimental evaluations were conducted on three real-world scientific workflows—*Montage*, *Epigenomics*, and *Sipht*—using VM configurations based on AWS EC2 instance data to ensure realism in modeling. Simulations performed under varying reliability levels (0.90, 0.95, 0.99) and increasing deadline flexibility showcased a clear trend: softer deadlines allow more relaxed task placement, reducing cost, while higher reliability thresholds predictably increase replication and cost. The results consistently showed that our optimization-based approach outperforms traditional iterative methods in terms of cost, especially under tight constraints.

Furthermore, statistical validation using two-way ANOVA confirmed that the choice of replica estimation method has a significant impact on cost, whereas variations in reliability level do not significantly affect it. This underscores the importance of a smart pre-allocation strategy in large-scale workflow environments.

Overall, the proposed approach enables scalable, reliable, and cost-aware execution of workflows in heterogeneous cloud environments. Future work can explore adaptive online scheduling, integration with spot pricing models, and energy-aware optimizations for further gains in sustainability and resource efficiency.

References

1. C. K. Swain and A. Sahu, "Reliability-ensured efficient scheduling with replication in cloud environment," *IEEE Systems Journal*, 2022.

2. E. Deelman, D. Gannon, M. Shields, and I. Taylor, “Workflows and e-Science: An overview of workflow system features and capabilities,” *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

3. H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

4. J. Yu and R. Buyya, “Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms,” *Scientific Programming*, vol. 14, no. 3–4, pp. 217–230, 2006.

5. Q. Peng, H. Jiang, M. Chen, J. Liang, and Y. Xia, “Reliability-aware and deadline-constrained workflow scheduling in mobile edge computing,” in *Proceedings of the 2022 IEEE International Conference on Communications (ICC)*, Chongqing, China, 2022.

6. S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, K. Vahi, and N. Wilkins-Diehr, “Characterization of scientific workflows,” in *2008 Third Workshop on Workflows in Support of Large-Scale Science*, 2008, pp. 1–10.

7. S. Pandey, W. Voorsluys, S. Jha, and R. Buyya, “A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments,” in *24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 400–407.