

A  
Major Project Report  
on  
**COST MINIMIZATION IN CLOUD COMPUTING WITH  
DEADLINE CONSTRAINT**

Submitted in partial fulfilment of the requirements for the award of the degree of  
Bachelor of Technology

by  
**P Phanindra Varshit**  
(20EG105430)

**T Abhinav Reddy**  
(20EG105444)

**P Siddartha**  
(20EG105436)



Under the guidance of  
**E. Radha Krishnaiah**  
Assistant Professor,  
Department of CSE

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
ANURAG UNIVERSITY  
VENTAKAPUR (V), GHATKESAR (M), MEDCHAL (D), T.S - 500088  
TELANGANA  
(2023-2024)**

## DECLARATION

We hereby declare that the report entitled “**Cost Minimization In Cloud Computing With Deadline Constraint**” submitted to the **Anurag University** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology (B. Tech)** in **Computer Science and Engineering** is a record of original work done by us under the guidance of **E. Radha Krishnaiah, Assistant Professor** and this report has not been submitted to any other university for the award of any other degree or diploma.

Place: Anurag University, Hyderabad

P Phanindra Varshit  
(20EG105430)

T Abhinav Reddy  
(20EG105444)

P Siddartha  
(20EG105436)



## CERTIFICATE

This is to certify that the project report entitled “**Cost Minimization In Cloud Computing With Deadline Constraint**” being submitted by **Mr. P Phanindra Varshit** bearing the Hall Ticket numbers **20EG105430**, **Mr. T Abhinav Reddy** bearing the Hall Ticket numbers **20EG105444**, **Mr. P Siddartha** bearing the Hall Ticket numbers **20EG105436** respectively in partial fulfilment of the requirements for the award of the degree of the **Bachelor of Technology in Computer Science and Engineering** to **Anurag University** is a record of bonafide work carried out by them under my guidance and supervision from 2023 to 2024.

The results presented in this report have been verified and found to be satisfactory. The results embodied in this report have not been submitted to any other University for the award of any other degree or diploma.

Signature of The Supervisor  
E. Radha Krishnaiah  
Assistant Professor

Signature Dean,  
Dr. G. Vishnu Murthy  
Department of CSE

External Examiner

## ACKNOWLEDGMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **E. RADHA KRISHNAIAH, Assistant Professor, Department of Computer Science and Engineering**, Anurag University for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved my grasp of the subject and steered to the fruitful completion of the work. His patience, guidance and encouragement made this project possible.

We would like to acknowledge our sincere gratitude for the support extended by **DR. G. VISHNU MURTHY**, Dean, Department of Computer Science and Engineering, Anurag University. We also express my deep sense of gratitude to **Dr. V. V. S. S. S. BALARAM**, Academic coordinator. **Dr. PALLAM RAVI**, Project Coordinator and project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stages of our project work.

We would like to express my special thanks to **Dr. V. VIJAYA KUMAR, Dean School of Engineering**, Anurag University, for his encouragement and timely support in our B. Tech program.

P Phanindra Varshit  
(20EG105430)

T Abhinav Reddy  
(20EG105444)

P Siddartha  
(20EG105436)

## **ABSTRACT**

The interest in performing scientific computations using commercially available cloud computing resources has grown rapidly in the last decade. However, scheduling multiple workflows in cloud computing is challenging due to its non-functional constraints and multi-dimensional resource requirements. Scheduling algorithms proposed in literature use search-based approaches which often result in very high computational overhead and long execution time. In this project, a Deadline-Constrained Cost Minimisation (DCCM) algorithm is proposed for resource scheduling in cloud computing. In the proposed scheme, tasks were grouped based on their scheduling deadline constraints and data dependencies. Compared to other approaches, DCCM focuses on meeting the user-defined deadline by sub-dividing tasks into different levels based on their priorities. Simulation results showed that DCCM achieved higher success rates when compared to the state-of-the-art approaches.

## **TABLE OF CONTENT**

<b>S. No.</b>	<b>CONTENT</b>	<b>Page No.</b>
1.	Introduction	1
	1.1 Budget Constrained Workflow	2
	1.2. Deadline Constrained Workflow	3
	1.3. Motivation	5
	1.4. Problem Definition	6
	1.5. Problem Illustration	7
	1.6. Objective of the Project	8
2.	Literature Survey	10
3.	Deadline Constrained Cost Minimization	18
	3.1. Partitioning of Workflow	19
	3.2. User Input	20
	3.3. Task Selection	21
	3.4. Grouping of tasks	25
	3.5. Virtual Machine Selection	26
	3.6. Cost Reduction	28
	3.7. Security Measures	30
	3.8 Key Generation and Delivery	32
4.	Implementation	33
	4.1. Functionality	33
	4.2. Attributes	35
	4.3. Experimental Screenshot	36

4.4. Database	39
5. Experimental Setup	41
5.1. Obtain Firebase API key	41
5.2. Setup Visual Studio Code	42
5.3. Libraries Used	43
5.4. Parameters	45
6. Discussion of Results	47
7. Summary, Conclusion and Recommendation	51
7.1. Further Evaluation and Validation	51
7.2. Real-World Deployment	52
7.3. Parameter Tuning and Optimization	52
7.4. Dual-Objective Scheduling Model Development	52
7.5. Community Collaboration and Benchmarking	52
8. Future Enhancements	53
9. References	54

### **List of Figures**

<b>Figure No.</b>	<b>Figure Name</b>	<b>Page No.</b>
1.1	Multistage Server	8
3.1	DAG Graph	19
3.2	A Sample Workflow	21

4.3.1	Uploading the Files	36
4.3.2	View the Uploaded Files	36
4.3.3	Files stored in Cloud	37
4.3.4	Files Requested	37
4.3.5	Download the Files	38
4.3.6	Secured Key	38
4.3.7	Download File	39
4.3.8	Decrypted File	39
4.4.1	Database	40
5.1.1	Firebase	42
5.2.1	VS Code	43
5.3.1	Pyrebase	44
5.3.2	Cryptography	44
5.3.3	Flask Setup	45
6.1	Comparision Graph	48
6.2	Cost Ratio	49
6.3	Mean Load	50

### **List of Tables**

<b>Table No.</b>	<b>Table Name</b>	<b>Page No.</b>
1.1	Performance Evaluation Table	8
2.1	Comparision of Existing Method from selected Strategies	16



3.5.1	VM Instance Specification	28
6.1	Comparision Evaluation	47

### **List of Abbreviations**

<b>Abbreviations</b>	<b>Full Form</b>
VM	Virtual Machine
DCCM	Deadline Constrained Cost Minimization
PSO	Practical Swarm Optimization
AES	Advance Encryption Standard
IC-PCP	IaaS Cloud Partial Critical Paths
DCLB	Deadline Constrained Level Based
DAG	Directed Acyclic Graph
SLO	Service Level Objective
IaaS	Infrastructure As Service
PaaS	Platform As Service
SaaS	Software As Service
API	Application Program Interface
TCO	Total Cost of Ownership
BDAS	Budget Driven Algorithm for Scheduling
HEFT	Heterogeneous Earliest Finish Time
BDHEFT	Budget-Driven Heterogeneous Earliest Finish Time

# 1. Introduction

Over last decade, there has been a large body of studies on scheduling problems in scientific workflows. This is due to the increase in the execution of scientific workflows in cloud computing platforms. Although this trend is advantageous, it is difficult for users to select resource configurations that are suitable for their applications [1], [2]. The focus on traditional workflow scheduling is at minimising the execution time of the workflow, using cluster, list and duplicate scheduling techniques as described in [3] and [4]. Recent surveys [5], [6] have shown that two main categories of workflow scheduling (budget-constrained and deadline-constrained) have been used frequently for resource scheduling in cloud computing in addition to multi-objective workflow scheduling. Budget-constrained scheduling strategies focus on the reduction in the time of execution of workflows while deadline-constrained strategies The associate editor coordinating the review of this manuscript and approving it for publication was Fan-Hsun Tseng. focused on the optimisation of the scheduling cost of the workflow [6], [7]. One of the main benefits of cloud computing is its scalability when applied to distributed systems. However, this often results in optimisation problems that are complex, in particular for tasks that are inter-dependent. One such problem is how to minimise cost. When cost and execution time are taken into consideration at the same time, the problem of optimising cost can be solved [8], [9], [10], [11], [12], [13]. This approach is very attractive as it can efficiently provide cost reductions for deadline-sensitive applications, as well as others such as disaster recovery, weather forecasting etc. However, a major drawback in this approach is the inconsistency in system requirements such as the acquisition delay which is the time taken for initialisation of a leased virtual machine [8], [9], instance heterogeneity [5], or the varying levels of performance in cloud systems [14], [15]. Complex systems like future networks rely increasingly on distributed intelligence for their management and as network functions are increasingly pushed towards the network edge, cloud scheduling and management solutions are necessary. We propose a dynamic resource scheduling strategy for cloud computing. To reduce processing cost, identical tasks were grouped together based on their priorities and constraints. This was done to reduce overheads from queuing, then the deadline of the entire workflow was sub-distributed across multiple groups of tasks. We implemented a resource scheduling strategy for Virtual Machines (VMs) that were dynamic and scalable. Extensive simulations were carried out to evaluate the performance of the proposed scheme. The simulation results showed that our proposed algorithm performs better than other existing task scheduling algorithms in terms of success rate at meeting the deadline and mean load.

### **1.1. Budget Constrained Workflow**

Budget-constrained workflow scheduling in cloud computing involves optimizing the allocation of resources while adhering to a predefined budget limit. This approach aims to minimize the cost of executing scientific workflows within the constraints of available financial resources.

In this context, "budget" refers to the monetary resources allocated for executing a workflow on the cloud platform. This budget could encompass various costs, including compute resources, storage, data transfer, and any additional charges associated with cloud services. The goal is to maximize the efficiency of resource utilization and minimize the overall expenditure while meeting the requirements of the workflow. One common strategy for budget-constrained scheduling is to prioritize resource allocation based on cost-efficiency metrics. This may involve selecting cheaper resource options or optimizing resource utilization to minimize costs without compromising the quality of service. For example, utilizing spot instances or preemptible VMs, which offer lower costs but may have limited availability, can be a cost-effective approach for certain workloads.

Additionally, budget-constrained scheduling may involve dynamic resource provisioning and allocation based on workload demands and pricing fluctuations. By continuously monitoring resource usage and cost metrics, the scheduler can make real-time decisions to optimize resource allocation and ensure that the workflow stays within budget constraints. Budget-constrained scheduling techniques often involve trade-offs between cost and performance metrics. For instance, sacrificing execution time or performance quality may be acceptable if it leads to significant cost savings. Therefore, the scheduler must strike a balance between cost optimization and meeting the workflow's performance objectives.

Overall, budget-constrained workflow scheduling in cloud computing requires careful planning, resource management, and optimization strategies to effectively utilize available financial resources while meeting the computational needs of scientific workflows. By employing cost-efficient resource allocation techniques and dynamic scheduling algorithms, organizations can achieve cost-effective execution of workflows in the cloud.

### **1.1.1. Advantages of Budget Constrained Workflow:**

The primary advantage of budget-constrained scheduling is its ability to optimize resource allocation to minimize costs while meeting application requirements. They can choose from a variety of pricing models, such as on-demand, spot instances, or reserved instances, to optimize resource costs based on workload characteristics and budget constraints. Cloud computing platforms offer virtually unlimited scalability, enabling organizations to scale resources up or down dynamically based on workload demands.

### **1.1.2. Challenges of Budget Constrained Workflow:**

Budget-constrained scheduling in cloud computing faces several challenges. Firstly, the dynamic nature of resource pricing introduces variability, making it difficult to accurately predict and manage costs. Fluctuations in prices can disrupt budget plans and affect cost-effectiveness. Additionally, reliance on cost-saving options like spot instances or preemptible VMs introduces concerns about resource availability and reliability, potentially impacting workflow execution. Effective budget allocation and management become crucial, requiring organizations to define constraints accurately, allocate resources efficiently, and monitor usage to ensure adherence to budget limits. Addressing these challenges is essential for successful budget-constrained workflow scheduling in cloud environments.

## **1.2 Deadline Constrained Workflow**

Deadline-constrained workflow scheduling in cloud computing is a critical aspect of managing scientific workflows with time-sensitive requirements. Unlike budget-constrained scheduling, which focuses primarily on cost optimization, deadline-constrained scheduling prioritizes meeting specific time constraints imposed by workflow tasks or applications. In this approach, the primary goal is to ensure that workflow execution completes within predefined deadlines while maximizing resource utilization and minimizing workflow execution time.

One of the key challenges in deadline-constrained workflow scheduling is to allocate resources efficiently to meet tight time constraints without compromising performance or incurring additional costs. Cloud computing platforms offer various resource provisioning options, including on-demand instances, reserved instances, and spot instances, which can be leveraged to meet deadline requirements effectively. Schedulers must dynamically allocate resources based on workflow characteristics, task dependencies, and deadline constraints to optimize workflow execution time while ensuring that deadlines are met.

Deadline-constrained workflow scheduling strategies often involve prioritizing tasks based on their deadlines and resource requirements. Tasks with tight deadlines or high resource requirements may be allocated higher-priority resources or given preferential treatment to ensure timely execution. Additionally, schedulers may employ techniques such as task replication, resource preemption, and dynamic resource scaling to meet deadline constraints and mitigate resource contention effectively.

Another challenge in deadline-constrained workflow scheduling is managing uncertainty and variability in cloud resource performance and availability. Cloud environments are inherently dynamic, with fluctuating resource availability, network latency, and performance characteristics. Schedulers must account for these uncertainties and adapt scheduling decisions dynamically to minimize the risk of deadline violations while maximizing resource utilization and performance.

Furthermore, deadline-constrained workflow scheduling often involves optimizing workflow execution pipelines to minimize critical path length and maximize concurrency. By identifying critical tasks and dependencies within workflows, schedulers can prioritize resource allocation and task scheduling to minimize workflow completion time and meet deadline constraints effectively. Techniques such as task clustering, dependency analysis, and parallel execution can help optimize workflow execution and improve overall performance.

In summary, deadline-constrained workflow scheduling in cloud computing is essential for meeting time-sensitive requirements and ensuring timely execution of scientific workflows. By dynamically allocating resources, prioritizing tasks, and optimizing workflow execution pipelines, schedulers can effectively meet deadline

constraints while maximizing resource utilization and performance in cloud environments.

### **1.2.1. Advantages of Deadline Constrained Workflow:**

Deadline-constrained scheduling offers several advantages in cloud computing workflows. Firstly, it ensures that tasks are completed within specified time limits by prioritizing them based on their deadlines. This helps minimize the risk of deadline violations and ensures timely execution of critical tasks. Additionally, deadline-constrained scheduling optimizes resource utilization by dynamically allocating resources according to task deadlines and requirements. This approach helps prevent resource underutilization and ensures that computing resources are efficiently utilized to meet deadline constraints, ultimately enhancing workflow efficiency and performance in cloud environments.

## **1.3 Motivation**

The motivation for deadline-constrained cost minimization in cloud computing stems from the increasing demand for efficient resource utilization and cost-effective workflow execution while meeting stringent time constraints. Scientific workflows often have time-sensitive requirements, where tasks need to be completed within specific deadlines to ensure timely results. However, executing these workflows in the cloud can incur significant costs, particularly when provisioning resources with varying performance characteristics and pricing models.

By incorporating deadline constraints into the cost minimization objective, cloud users aim to optimize resource allocation and scheduling decisions to meet workflow deadlines while minimizing execution costs. This approach is motivated by the need to balance performance, cost, and time constraints effectively, especially for deadline-sensitive applications such as real-time data processing, financial analytics, and simulation-based research.

The motivation for deadline-constrained cost minimization also arises from the desire to improve resource efficiency and maximize return on investment in cloud computing environments. By minimizing costs while meeting deadline constraints,

cloud users can achieve optimal resource utilization, reduce operational expenses, and enhance the overall cost-effectiveness of their workflow execution strategies.

#### **1.4 Problem Definition**

In the realm of cloud computing, where resources and services are not provided for free, the concept of value exchange forms the cornerstone of the ecosystem. Cloud computing has garnered increasing attention due to its inherent flexibility and economic advantages. However, to sustain a healthy supply-demand relationship among the various stakeholders within the cloud environment, there must be a reciprocal exchange of value. From the perspective of cloud service providers, the primary objective revolves around profitability. These providers aim to maximize their revenue by efficiently utilizing their resources to fulfill the tasks submitted by customers. In the context of scientific workflows, which entail complex sequences of computational tasks, effective resource scheduling becomes paramount.

Efficient resource scheduling in cloud computing involves allocating computational resources in such a way that workflow execution is optimized while adhering to budget constraints. This entails making judicious decisions regarding resource provisioning, task allocation, and scheduling policies to ensure that computational resources are utilized effectively and efficiently.

By optimizing workflow execution within budget constraints, cloud service providers can achieve several objectives. Firstly, they can enhance their operational efficiency by minimizing resource wastage and maximizing resource utilization. This, in turn, allows providers to offer competitive pricing to customers while maintaining profitability. Moreover, efficient resource scheduling enables cloud service providers to better meet the diverse needs of their customers. By ensuring timely and cost-effective execution of scientific workflows, providers can enhance customer satisfaction and loyalty, leading to long-term business success.

## 1.5. Problem Illustration

Practical Swarm Optimization (PSO) draws inspiration from the collective behaviour of natural swarms, like bird flocks or ant colonies, to solve computational problems. While it's a fascinating approach, its direct application to cloud computing scenarios presents several challenges.

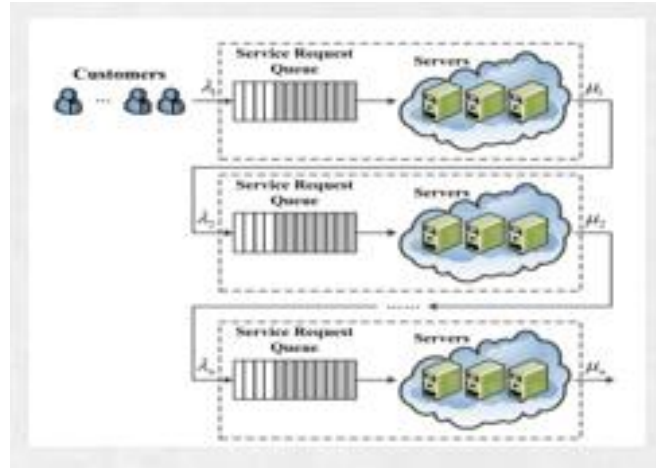
One of the primary limitations of PSO in cloud computing is its lack of consideration for deadlines. In many cloud-based applications, tasks must be completed within specific time frames to meet user requirements or service level agreements. Since PSO typically doesn't incorporate deadlines into its optimization process, it may not be suitable for such time-sensitive applications.

Moreover, PSO may not always be the most cost-effective optimization technique in cloud computing environments. Efficient resource allocation and cost minimization are crucial aspects of cloud computing, and PSO may not excel in these areas compared to other optimization techniques.

Another challenge arises from the use of multistage servers in PSO. While these servers can enhance computational efficiency by processing tasks in multiple stages, they can also introduce complexities such as traffic congestion in server queues. This congestion can lead to incomplete tasks or delays in processing, undermining the overall effectiveness of PSO in cloud environments. PSO does not inherently provide robust security measures for data stored in the cloud. In today's data-driven world, protecting sensitive information from unauthorized access or data breaches is paramount. Therefore, additional security measures may be necessary to safeguard data when employing PSO in cloud computing applications.

Furthermore, the problem of multistage servers introduces another issue: if one task fails to execute successfully, it can potentially block subsequent tasks in the queue from being processed, even if they are capable of being executed. This can lead to inefficiencies and disruptions in workflow execution, diminishing the reliability and performance of PSO in cloud environments.





**Figure 1.1. Multistage Server**

Task	Memory Utilization	Estimated Cost	Time
Task1	100%	100%	Time bounded
Task2	100%	100%	Time bounded
Task3	0%	100%	Delay
Task4	0%	100%	Delay

**Table 1.1. Performance Evaluation Table**

## 1.6. Objective of the Project

The objective of deadline-constrained cost minimization in cloud computing is to optimize resource utilization and minimize the overall cost of executing scientific workflows while meeting strict deadline constraints. This objective is particularly relevant in scenarios where timely completion of workflows is essential, such as real-time data processing, financial transactions, and critical decision-making applications.

By integrating cost considerations with deadline constraints, the goal is to achieve a balance between minimizing the financial expenditure associated with resource provisioning and maximizing the likelihood of meeting workflow deadlines.

This involves dynamically allocating cloud resources, such as virtual machines, storage, and network bandwidth, to execute workflow tasks efficiently while keeping costs within budgetary limits.

The primary aim is to identify cost-effective resource configurations and scheduling strategies that enable workflows to meet their specified deadlines without overspending on unnecessary resources or incurring penalties for missed deadlines. This requires sophisticated optimization algorithms and scheduling techniques that consider factors such as resource availability, pricing models, task dependencies, and performance requirements.

## 2. Literature Survey

The problem of scheduling scientific workflows has been well studied in the literature. Several strategies have been proposed for effective resource provisioning and scheduling of scientific workflows [16]. These algorithms are based on heuristics, meta-heuristics and search-based techniques [10]. Generally, resource scheduling algorithms focus on the reduction in execution time of workflows with two main factors in consideration, monetary cost and deadline. Resource allocation in cloud computing comprises of two phases, resource provisioning and scheduling. Resource provisioning is responsible for the determination of resources required for the execution of workflows. Resource scheduling pays attention to the scheduling, placement and execution of tasks. Although most scheduling algorithms in cloud computing systems propose resource provisioning and scheduling algorithms for cloud computing systems. Prior research shows that much attention has been paid to resource scheduling strategies [17], [18]. One of the widely used approaches in resource scheduling is Particle Swarm Optimisation (PSO) which was introduced by the authors in [19]. PSO is a self-adaptive optimisation technique that relies on a particles' social behaviour to solve task allocation and resource scheduling problems [2]. In [20], the authors proposed a resource scheduling strategy based on PSO. The proposed algorithm was aimed at reducing the cost of a single workflow execution and load balancing of task based on available resources. In [21], authors proposed Deadline Constrained Level Based (DCLB) which uses Level Load Balancing to refine deadline distribution as well as attaining lower communication cost. Wu et al. [22] also proposed a PSO algorithm to minimise cost and execution time, whilst considering budget and deadline constraints. The proposed algorithm focused on continuous optimisation problems that have no prior information. Rodriguez and Buyya [10] propose a static PSO algorithm for resource scheduling in complex scientific workflows. They introduced several characteristics of cloud services including heterogeneity of virtual machines, lease time interval, pricing model, and acquisition delay. Their algorithm effectively reduced the execution cost and met the deadline defined by the user using a global optimisation technique. One major disadvantage common to the highlighted PSO algorithm was their ability to manage heterogeneous resources. However, these algorithms are not suitable for deployment in Infrastructure as a Service (IaaS) as they do not consider the order of execution of task and the number and type of resources to be leased. The static PSO in [10] assigned tasks to instances randomly, thereby neglecting the

characteristics and structure of the workflow. In [8], the authors developed an IaaS Cloud Partial Critical Paths (IC-PCP) for the IaaS model. IC-PCP scheduled all task to the same VM instance which must be the cheapest instance that must meet the deadline given by the user. In IC-PCP, the critical path or set of tasks that is related to the exit node must first be found, then scheduled on the same VM. ICPCP does not add additional overhead to the critical paths. However, the algorithm does not consider the boot time of VMs. Such shortcomings were identified by Calheiros and Buyya [23] who extended the algorithm Enhanced IC-PCP Algorithm with Replication (EIPR), by replicating task using idle instances and surplus in budgets. These enhancements show the possibility of meeting user defined deadlines by an increase in the replication of tasks. Such replicated computation resulted in a very high computational overhead. Although EIPR showed promising results in the mitigation of variable performance effects by the exploitation of the billing schemes and cloud elasticity. Is performance is low when the task execution time is near the billing period.

We discussed the resource scheduling techniques proposed for scientific workflows in cloud computing. In [24], a budget constrained algorithm was proposed to deal with optimisation in workflow scheduling. In this approach, the total budget as defined by the user was proportionally distributed to each task based on the average time of execution. Arabnejad and Barbosa [9] introduced the concept of worthiness as an attribute in scientific workflow applications. In this approach, worthiness was used for resource selection and were determined based on cost and time of execution. Wu et al. [25] propose a budget-constrained algorithm PCB-B 2 . This approach used a binary search method to execute its budget distribution. In [26], the authors proposed a task swapping strategy for scientific workflows. In this approach, tasks with smaller execution time were executed before others, which lowered the optimisation cost. In [21], the authors proposed a level based (DCLB) scheduling algorithm for cloud computing. DCLB is aimed at reducing the makespan of a workflow while meeting the user-defined deadline at the same time. The authors divided the workflow into logical levels using parallelism strategy while achieving load balancing at the same time. However, the context of the logical level partitioning is not really clear and the task completion of the parent task is a prerequisite to the initiation of the child task. In [27], the authors proposed an algorithm for scheduling scientific workflows based on deadline distribution factor. The proposed algorithm (Just-in-time JITC) was developed to mitigate performance issues in VMs. This algorithm focused on three main objectives including; the variation in the performance of VMs, heterogeneity in cloud systems and

resource acquisition delay. Although, the authors [27] claimed that this method was very effective for deadline, it was computationally expensive to generate schedules when the deadline factor is low. Hadiri et al., [28], highlighted acquisition delay as a significant barrier in workflow scheduling. To optimise the total execution time required to complete a workflow task, they suggested a cost-effective technique (CEDA) for minimising task execution time while meeting deadlines. CEDA, on the other hand, is not ideal for large workflows. Topcuoglu et al. [3] propose some of the earliest resource scheduling algorithms for heterogeneous computing. Their two algorithms aimed at achieving fast scheduling time and high performance simultaneously. Heterogeneous Earliest Finish Time (HEFT) and Critical-Path-On-a-Processor (CPOP) are the two algorithms proposed by these authors. HEFT focuses on minimising the earliest finish time with a novel strategy called incentive-based task. On the other hand, CPOP pays attention to the prioritisation of tasks based on upward and downward ranks. One major disadvantage of HEFT is that load balancing is not considered and workflows are scheduled one at a time. In [29], the authors proposed a budget-aware algorithm (BDAS) for resource scheduling in heterogeneous e-scientific workflows. The study showed that high attention has been paid to optimisation in scientific workflows, with particular focus on time and cost constraints. BDAS approach tries to satisfy deadline and budget constraints by the introduction of heterogeneous instances on cost and time. Budget and Deadline HEFT (BDHEFT) was proposed by Verma and Kaushal [30]. BDHEFT is another extension of HEFT with emphasis on budget and deadline. The BDHEFT algorithm used a ranking mechanism for the identification of tasks, which is done in descending order. BDHEFT has shown promising results with reduction in makespan and execution cost of workflows. However, BDHEFT's dependence on prior information limits its static global point of view.

The focus on traditional workflow scheduling is at minimising the execution time of the workflow, using cluster, list and duplicate scheduling techniques as described in [3], [4]. Recent surveys [5], [6] have shown that two main categories of workflow scheduling (budget-constrained and deadline-constrained) have been used frequently for resource scheduling in cloud computing in addition to multi-objective workflow scheduling. Budget-constrained scheduling strategies focus on the reduction in the time of execution of workflows while deadline-constrained strategies focus on the optimisation of the scheduling cost of the workflow [6], [7]

Tasks are integrated into a group of tasks using the horizontal grouping strategy adopted by the authors in [6]. This means that tasks in the same group have same capabilities and same depth to enable parallel processing. In addition to achieving optimisation by using different techniques and strategies, the optimisation approaches also differ in terms of the workloads they focus on. Examples include scientific workflows [9,19,20], gaming [21], medical and healthcare [17,22], education [23,24], and big data [25,26] workflows. In this paper, instead of focusing on a single type of workload, i.e. single application type, we provide a general approach based on service resource utilisation suitable for any application deployed as a SaaS service. We base the choice of the used SLOs on the data that was gathered during the measurement process and define two SLOs based on the response time QoS metric. The SLOs used in this analysis are defined by [19] according to [5] and [6] where SLO 1 refers to the maximum percentage of user requests in the last 24 hours whose response times can exceed the threshold specified by the SLO for video streaming service [20] and MRS service [22]. SLO 2 refers to the average response time value that should be less than 5 seconds on a daily basis for video streaming service (21) For the simulation of autoscaling, the autoscaling parameter was determined for each service based on the ranked parameters and their influence on the application QoS.

In order to get a smaller makespan, the algorithm attempts to use the backtracking method to select faster and more expensive resources for critical tasks as much as possible. Rodriguez et al. [17] proposed a budget-driven algorithm with fine-grained billing periods, the purpose of which is to optimize the way in which the budget is spent to minimize the application's makespan (i.e., total execution time). Arabnejad et al. [37] proposed a scheduling algorithm as Budget Distribution with Trickling (BDT) and concluded that the earlier calculation of biasing the budget distribution to the workflow would result in a lower makespan within the budget.

The fine-grained billing period is an emerging billing period in the cloud computing environment in the past two years. Flexibility is limited when coarse-grained billing periods, so that it is easy to cause inevitable waste [17]. In September 2017, Amazon announced that EC2 will use the per-second billing period from October 2017 [51]. That could negatively affect the stability of the system in case if the autoscaling decisions take too much time, and do not scale well with workload fluctuations. Shorter

autoscaling intervals that are more in line with the current trend on fine-grained billing [27] further complicate the problem, as plan-based autoscalers simply do not have enough time for making their decisions. Moreover, the plan-driven task placement can possibly delay the execution of newly arrived workflows as the scheduler will need to wait until the new plan incorporating the newly arrived tasks is constructed. In the simple case when the system would have only a single resource type any of the considered autoscalers would not make sense as each user would simply get the number of instances which its budget allows to allocate at maximum. We use autoscaling interval of one minute to be in line with the current trend on fine-grained billing [27]. Since we report the imposed system utilization, we believe that the same behavior should be observed for shorter or longer autoscaling intervals, if the utilization will be accordingly adjusted. The Static Provisioning Static Scheduling (SPSS) [21] is an offline autoscaler that creates a plan for each workflow in the ensemble, and rejects workflows that exceed the deadline or budget. BAGS [27] is a plan-based offline autoscaler that partitions workflows into bags-of-tasks and then applies a MIP-based approach to make the allocation plan. The majority of the considered works performs simulations when evaluating the proposed algorithms.

The deadline-constrained scheduling aims to minimize the cost while ensuring that the workflow's deadline is met [2,3]. On the other hand, the budget-constrained scheduling focuses on minimizing the total execution time while keeping the execution cost below the budget set by the user [4,5]. This paper focuses on investigating the problem of deadline-constrained workflow scheduling. Heuristic methods consist of problem-specific rules that aim to find satisfactory solutions within a short timeframe. Many heuristics adopt greedy strategies for quick decision-making, but this can lead to getting stuck in local optima [2][3][4]. In contrast, meta-heuristic approaches are general-purpose strategies for solving optimization problems, which need to be fine-tuned or customized for specific problems like scheduling.

There are many research papers on the topic of workflow scheduling that differ in the objective functions and/or the proposed strategy to pursue the objective efficiently, keeping in mind that, usually, the resulting scheduling problem is NP-complete [4], [38]. Some schedulers aim to minimize the workflow makespan given a resource or monetary budget [39], others to minimize the monetary cost given a

completion deadline to respect [40], [41]. To pursue their objective function different schedulers use different strategies. For instance, [40], [41] partition the workflow into paths of dependency tasks based on specific criteria and then a scheduling decision is made for each path. The schedulers proposed in [39], [42] create groups of tasks (akin super-task), rank them and perform the scheduling decision. Similarly, [18] [19] [20] [24] can cluster small tasks into larger, more compute-intensive ones and this is especially helpful when executing workflows with thousands of tasks, where the start-up time overwhelms the overall computation. For implementation of business workflows, business models like Amazon EC2 (Elastic Compute Cloud) are used [9]. The cloud computing platforms PaaS and SaaS are significantly used to deploy such type of business applications [10], [11]. On the other hand, large-scale scientific applications, such as Montage [12], [13] and CyberShake [12], [14], are dataoriented scientific applications and organized as scientific workflows. Scientific workflow management systems like Pegasus WMS (Workflow Management System) are used for implementation of scientific workflows [15]. The cloud computing platform IaaS is magnificently used for the deployment of scientific applications [10].

In addition to achieving optimisation by using different techniques and strategies, the optimisation approaches also differ in terms of workloads they focus on. Examples include scientific workflows [9,19,20], gaming [21], medical and healthcare [17,22], education [23,24], and big data [25,26] workflows. In this paper, instead of focusing on single type of workload, i.e. single application type, we provide a general approach based on service resource utilisation suitable for any application that can be deployed as a SaaS service. We base the choice of the used SLOs on the data that was gathered during the measurement process and define two SLOs based on response time QoS metric. The SLOs used in this analysis are defined by [19] according to [5] and [6] where SLO 1 refers to the maximum percentage of user requests in the last 24 hours whose response times can exceed the threshold specified by the SLO for video streaming service [20] and MRS service [22]. SLO 2 refers to the average response time value that should be less than 5 seconds on a daily basis for video streaming service [21] or 3 seconds for MRS service [23].



Sl.No	Author	Strategies	Advantages	Disadvantages
1	M. A. Rodriguez and R. Buyya,	novel heuristics tailored for the cloud resource model	propose a scheduling algorithm whose objective is to optimize a workflow's execution time under a budget constraint	This method deals with per minute billing not on cost saving methods
2	Z.-H. Zhan, X.-F. Liu, Y.-J. Gong	Evolutionary Computation (EC) algorithms	Research which including real-time scheduling, adaptive dynamic scheduling, large-scale scheduling, Mult objective scheduling	Its not practical work but gives over view on various scheduling methods.
3	M. Naghibzadeh, and D. H. J. Epema	two-phase scheduling algorithm for utility Grids,	algorithms have a polynomial time complexity which make them suitable options for scheduling large workflows.	It uses two algorithms which take more time for scheduling
4	M. A. Rodriguez and R. Buyya	Survey methods POS, DCPC	A detailed taxonomy that focuses on features particular to clouds is presented, and the surveyed algorithms	Survey Comparing various scheduling algorithms only
5	S. Pandey, L. Wu, S. M. Guru, and R. Buyya	PSO	results show that PSO can achieve: (a) as much as 3 times cost	The static PSO in assigned tasks to instances randomly, thereby

			savings as compared to BRS, and (b) good distribution of workload onto resources.	neglecting the characteristics and structure of the workflow
--	--	--	---	--

Table 2.1. Comparision of Existing Method from selected Strategies

### **3. Deadline Constraint Cost Minimization**

The proposed method for Deadline Constraint Cost Minimization (DCCM) in cloud computing offers a comprehensive solution for managing tasks, optimizing costs, and ensuring robust data security in cloud-based storage systems.

At the core of this method is the ability for users to select files for storage in the cloud and specify task deadlines. This empowers users with the flexibility to prioritize tasks based on their urgency and importance. Additionally, users provide relevant information such as file size and required computation power, which helps in accurately assessing the resources needed for task execution.

Once tasks are queued with their associated information, a sophisticated scheduling algorithm comes into play. This algorithm evaluates various factors including workload, resource availability, and task requirements to select the most suitable virtual machine (VM) for executing the tasks. The primary objective of the scheduling algorithm is to minimize costs while ensuring that deadlines are met. By dynamically allocating resources based on task characteristics and system conditions, the chosen VM optimizes the processing of tasks, thereby maximizing efficiency and resource utilization.

Before files are stored in the cloud, they undergo AES encryption to enhance data security. AES encryption is a widely recognized standard for encrypting sensitive data, ensuring that files remain protected from unauthorized access or tampering. Furthermore, decryption keys are securely generated and delivered only to authorized users, adding an extra layer of security to the data storage process.

The main objective of the proposed algorithm is to minimise cost while meeting budget and deadline constraints. The proposed scheme is made up of three main phases which include, partitioning of the workflow, deadline distribution, selection of task and resource scheduling algorithm.

### 3.1 Partitioning Of Workflow

Workflow partitioning is a critical aspect of workflow management systems, aiming to efficiently allocate tasks to resources by prioritizing and grouping them based on their functionalities and dependencies. The primary objective is to organize tasks into cohesive groups that can be executed in parallel, thereby optimizing workflow execution time and resource utilization.

In the process of workflow partitioning, tasks are initially prioritized based on their dependencies within the workflow. This involves analyzing the workflow's directed acyclic graph (DAG) to identify dependencies between tasks. Tasks that have no dependencies or whose dependencies have already been satisfied are given higher priority as they can be executed sooner.

One effective approach to task prioritization is computing the length of the critical path from the entry task to the exit task. The critical path represents the longest path through the workflow, determining the minimum time required to complete the workflow. By prioritizing tasks along this critical path, workflow partitioning ensures that the most time-consuming tasks are executed first, thereby minimizing the overall workflow execution time.

The ranking process plays a crucial role in determining the priority of tasks. It involves recursively computing ranks by traversing the DAG from the exit node towards the entry node. During this traversal, tasks are assigned ranks based on their dependencies and the critical path length. Tasks with higher ranks are considered more critical to the workflow and are therefore prioritized for execution.

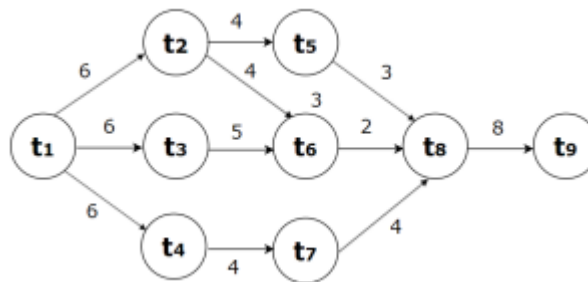


Figure 3.1. DAG Graph

This ranking process provides a systematic and efficient way to prioritize tasks, taking into account their dependencies and contribution to the overall workflow

completion time. By focusing on tasks along the critical path, workflow partitioning can effectively identify and prioritize tasks that have the greatest impact on workflow performance.

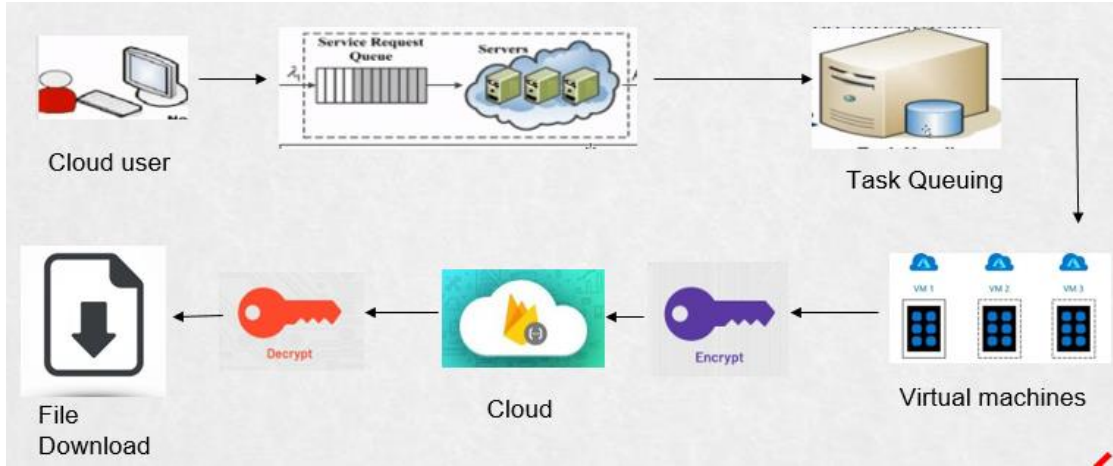
Overall, the approach of prioritizing tasks based on dependencies and critical path length in workflow partitioning is highly effective in optimizing workflow execution. It allows for the efficient allocation of resources and ensures that tasks are executed in an order that minimizes the overall completion time of the workflow.

### **3.2. User Input**

When the user interacts with the system, they are presented with an interface where they can browse and select a document from their local file system. This selection process allows the user to choose any file they wish to store in the cloud, whether it's a text document, image, video, or any other type of file.

Once the user has identified the file they want to upload, they initiate the storage process by selecting it within the interface. Alongside selecting the file, the user also specifies a deadline for when they need the task to be completed. This deadline could be based on various factors, such as project timelines, collaboration requirements, or regulatory constraints. After the file selection and deadline specification, the system proceeds to process the user's request. This may involve tasks such as encrypting the file for security, determining the optimal storage location within the cloud infrastructure, and scheduling the transfer process to meet the specified deadline.

Overall, this user input mechanism provides flexibility and control to the user, allowing them to seamlessly upload any document from their local storage to the cloud while specifying their desired completion timeframe.



**Figure 3.2. A Sample Workflow**

### 3.3. Task Selection

The proposed scheme for task execution involves holding tasks in an execution queue until their predecessors are executed, ensuring that tasks are executed in the correct order according to their dependencies within the workflow. Tasks that are grouped together can be executed simultaneously, optimizing resource utilization and workflow efficiency.

Before executing tasks, the scheme considers the distribution of deadlines among tasks to ensure that the global deadline of the workflow is achieved. This involves analyzing the deadline requirements of individual tasks and distributing them among different levels of the workflow hierarchy. By distributing deadlines in this manner, the scheme aims to balance the workload and prevent any single task from delaying the overall completion of the workflow.

To compute the data transfer time required for task execution, the scheme utilizes available virtual machine (VM) instances. These instances are used to estimate the time required to transfer data between tasks and ensure that data dependencies are satisfied within the specified deadlines.

To further ensure that the overall deadline of the workflow is met, the scheme implements sub-level deadlines for each subtask. These sub-level deadlines define the maximum allowable time for the completion of each individual task within the workflow. By setting sub-level deadlines, the scheme can effectively manage task execution and prevent any delays that could impact the overall workflow deadline.

In practice, the scheme calculates the available time for task execution by considering the elapsed time between the workflow's deadline and the estimated makespan. The makespan represents the total duration required to complete all tasks within the workflow. By comparing the available time with the estimated makespan, the scheme can determine whether tasks can be executed within the specified deadlines and take appropriate actions to meet the overall deadline of the workflow. Overall, this approach ensures efficient task execution and timely completion of workflows in cloud computing environments.

### **3.3.1. Task Prioritization and Grouping:**

Once tasks are prioritized, the scheme focuses on grouping similar tasks together for concurrent execution. This grouping strategy aims to capitalize on shared resource requirements and dependencies among tasks, allowing them to be executed simultaneously wherever possible. By executing related tasks concurrently, the scheme optimizes resource utilization and minimizes idle time, thereby enhancing overall workflow efficiency.

Furthermore, task grouping enables the scheme to exploit parallelism inherent in the workflow, leveraging available computational resources more effectively. By executing similar tasks concurrently, the scheme can distribute computational workloads more evenly across available resources, avoiding bottlenecks and accelerating overall workflow execution.

The prioritization and grouping of tasks represent fundamental steps in the proposed workflow management scheme, facilitating efficient resource allocation, minimizing execution delays, and ultimately contributing to the timely completion of complex workflows in cloud computing environments.

### **3.3.2. Execution Queue Management:**

In the devised workflow management strategy, a crucial aspect involves the meticulous handling of task sequencing to honor their dependencies accurately. This

begins by instituting an execution queue where tasks await their turn until all their prerequisite tasks, or predecessors, have been satisfactorily completed. This approach ensures that the workflow progresses in a structured manner, adhering to the predetermined sequence dictated by task dependencies. Once a group of tasks has all its dependencies fulfilled, they are strategically executed simultaneously. This concurrent execution of grouped tasks capitalizes on the optimized resource allocation and enables efficient utilization of available computational resources. By executing related tasks concurrently, the scheme not only expedites workflow execution but also maximizes resource utilization, thereby enhancing overall system efficiency.

When a task is initiated by the user, it enters into a queue within the cloud computing environment, awaiting its turn for processing. This queue serves as a holding area where tasks are organized and managed before they are executed. Along with the file selected by the user, additional information essential for task processing is included. The task encapsulates various pieces of information crucial for its execution. Firstly, it contains details about the deadline specified by the user, indicating the time by which the task needs to be completed. This deadline serves as a critical parameter for scheduling and prioritizing tasks within the queue.

Furthermore, the task includes metadata regarding the size of the file being uploaded. This information is vital for resource allocation and capacity planning within the cloud infrastructure. Tasks with larger file sizes may require more computational resources and storage space, influencing their priority and scheduling within the queue. The task specifies the computation power required for processing. This computation power encompasses various factors such as CPU, memory, and network bandwidth necessary for task execution. By including this information, the task queue can efficiently allocate resources and optimize task execution based on the specific requirements of each task.

In summary, the task queuing process involves organizing tasks within a queue based on their deadline, file size, and computation power requirements. This ensures efficient resource utilization and timely execution of tasks within the cloud computing environment.



### **3.3.3. Data Transfer Time Computation:**

In the realm of cloud computing, the computation of data transfer time stands as a critical aspect governing the efficiency and performance of data-intensive workflows. This process involves estimating the time required to transfer data between different components or entities within a cloud-based environment, such as virtual machines, storage systems, or network resources.

The data transfer time computation encompasses various factors, including the volume of data being transferred, the bandwidth of the network connection, and the latency associated with data transmission. By analyzing these factors, cloud computing systems can accurately predict the time needed to transmit data from one location to another, enabling efficient resource allocation and task scheduling.

In cloud environments characterized by distributed architectures and geographically dispersed resources, the computation of data transfer time becomes particularly crucial. Cloud service providers must optimize data transfer operations to minimize latency and maximize throughput, ensuring timely and reliable data exchange across diverse computing environments.

### **3.3.4. Available Time Calculation:**

The calculation of available time serves as a crucial aspect in the scheduling and execution of tasks within workflows. Available time refers to the duration or window of opportunity during which computational resources, such as virtual machines (VMs) or processing units, are accessible and can be allocated for executing tasks.

The computation of available time involves assessing various factors, including resource availability, workload demands, and scheduling constraints. Cloud computing platforms employ sophisticated algorithms to determine the optimal allocation of resources based on these factors, ensuring efficient utilization and timely completion of tasks.

One key aspect of available time calculation is the consideration of workload fluctuations and resource contention. Cloud environments often experience dynamic changes in resource availability due to factors such as user demand, hardware failures,

or maintenance activities. As such, cloud providers must continuously monitor resource availability and adjust task scheduling accordingly to maximize resource utilization while meeting performance objectives.

Additionally, available time calculation takes into account task dependencies and priority levels within workflows. Tasks with high priority or critical dependencies may require immediate allocation of resources to meet stringent deadlines or ensure smooth workflow execution. Cloud scheduling algorithms prioritize these tasks and allocate resources accordingly to minimize delays and optimize overall workflow performance.

In cloud environments characterized by multi-tenancy and shared infrastructure, available time calculation considers resource isolation and allocation policies to prevent resource contention and ensure fair resource distribution among users. By accurately calculating available time and dynamically allocating resources, cloud providers can optimize resource utilization, minimize task completion times, and enhance the overall efficiency of cloud-based workflows.

Available time calculation is a fundamental aspect of cloud computing, enabling efficient resource allocation and task scheduling to meet performance objectives and user requirements. By considering factors such as resource availability, workload dynamics, and task dependencies, cloud providers can optimize resource utilization and deliver reliable and scalable cloud services to users.

### **3.4. Grouping of Tasks**

In the workflow management system proposed by the authors, tasks are organized into groups using a horizontal grouping strategy outlined in [6]. This strategy entails grouping tasks together based on shared characteristics, such as capabilities and depth, to facilitate parallel processing. Rodriguez and Buyya et. al. [6] elaborate on this horizontal grouping approach, suggesting that tasks within the same group are assigned a common identifier, often represented by a specific color.

Each group of tasks may consist of one or multiple individual tasks, depending on their similarities in terms of data size, input/output requirements, computational complexity, and other attributes. The grouping ensures that tasks with homogeneous characteristics are processed together, utilizing a single immediate processor. This cohesive grouping approach is designed to enhance efficiency by enabling parallel execution of tasks with similar attributes.

Furthermore, tasks grouped together typically share common data distribution patterns and belong to the same task type or category. This uniformity within task groups simplifies resource allocation and management, as tasks with comparable attributes can be processed using similar computational resources and configurations.

Overall, the horizontal grouping strategy facilitates streamlined task management and efficient resource utilization within the workflow system. By organizing tasks based on shared characteristics and enabling parallel processing, this approach contributes to improved performance and scalability in handling complex workflows in cloud computing environments.

### **3.5. Virtual Machine Selection**

When tasks are ready for processing within the cloud environment, a scheduling algorithm steps in to optimize resource allocation and ensure timely completion while minimizing costs. This algorithm systematically evaluates available virtual machines (VMs) based on several critical factors. The algorithm considers the current workload of each VM. By analyzing the workload distribution across the available VMs, it aims to distribute tasks evenly, preventing resource bottlenecks and ensuring efficient resource utilization.

The algorithm examines the available resources on each VM. This includes factors such as CPU capacity, memory availability, and network bandwidth. By matching the task's requirements with the capabilities of each VM, the algorithm ensures that tasks are assigned to VMs equipped to handle them effectively. Moreover, compatibility between the task's requirements and VM specifications is a key consideration. The algorithm assesses whether the VM possesses the necessary software dependencies and configurations required for executing the task seamlessly.

The algorithm takes into account the deadline constraint specified by the user for each task. It evaluates the estimated processing time on each VM and selects the one that can complete the task within the specified deadline, prioritizing tasks accordingly.

By analysing these factors collectively, the scheduling algorithm identifies the most suitable VM for each task. This strategic VM selection process aims to optimize resource utilization, minimize costs, and ensure timely task completion, thereby enhancing the overall efficiency of the cloud computing environment.

#### **3.5.1. General-Purpose VMs:**

These VMs are designed to handle a wide range of workloads and applications. They offer a balance of compute, memory, and storage resources, suitable for tasks with moderate resource requirements.

#### **3.5.2. Compute-Optimized VMs:**

Optimized for compute-intensive workloads, these VMs provide high-performance CPUs and ample memory to support tasks that require significant computational power, such as data analytics, scientific simulations, and rendering.

#### **3.5.3. Memory-Optimized VMs:**

These VMs are equipped with large amounts of memory and are ideal for memory-intensive applications, such as large databases, in-memory caching, and real-time analytics. They prioritize memory capacity over CPU performance.

#### **3.5.4. Storage-Optimized VMs:**

Designed for workloads that demand high storage capacity and throughput, these VMs feature local storage disks optimized for data-intensive applications, such as database servers, content delivery networks (CDNs), and data warehousing.

#### **3.5.5. GPU-Enabled VMs:**

VMs with Graphics Processing Units (GPUs) are tailored for tasks that require accelerated graphics rendering, parallel processing, and machine learning inference. They are commonly used in industries like gaming, digital media, and artificial intelligence.

### 3.5.6. Network-Optimized VMs:

These VMs prioritize network performance and bandwidth, making them suitable for applications that rely heavily on network communication, such as web servers, video streaming, and content delivery services.

### 3.5.7. Preemptible VMs:

Also known as spot instances or preemptible instances, these VMs offer discounted pricing in exchange for the possibility of being reclaimed by the cloud provider with short notice. They are suitable for non-critical workloads and batch processing tasks that can tolerate interruptions.

### 3.5.8. Custom VM Configurations:

Some cloud providers allow users to create custom VM configurations by specifying the desired amount of CPU, memory, storage, and other resources. This flexibility enables users to tailor VMs to their specific workload requirements.

VM Types	Memory(GB)	ECU	Price (\$/h)
m3.medium	3.75	3	0.067
m4.large	8	6.5	0.126
m3.xlarge	15	13	0.266
m4.2xlarge	32	26	0.504
m4.4xlarge	64	53.5	1.008
m4.10xlarge	160	124.5	2.520

**Table 3.5.1. VM Instance Specification**

## 3.6. Cost Reduction

Cost reduction is a crucial aspect of cloud computing, and efficient resource utilization plays a significant role in achieving this objective. The selection of an appropriate virtual machine (VM) is fundamental to cost reduction strategies, as it directly impacts resource allocation and utilization efficiency. By dynamically allocating CPU, memory, and storage resources based on the requirements of the task and the available resources, the chosen VM optimizes resource utilization, thereby reducing overall costs.

One approach to cost reduction is through dynamic resource allocation, where the VM adjusts its resource allocation based on the changing demands of the workload. This dynamic allocation ensures that resources are not over-provisioned, leading to wastage, or under-provisioned, resulting in performance degradation. Instead, resources are allocated precisely when and where they are needed, maximizing efficiency and minimizing costs.

For example, consider a scenario where a data processing task requires significant CPU resources initially but transitions to a memory-intensive phase later in its execution. By dynamically adjusting the allocation of CPU and memory resources, the chosen VM can ensure optimal performance throughout the task's lifecycle while avoiding unnecessary resource provisioning. This dynamic resource allocation strategy helps reduce costs by eliminating resource idle time and minimizing the need for expensive over-provisioning.

Furthermore, the cost reduction strategy may also take into account factors such as the pricing models of cloud providers and optimization techniques specific to resource usage. Cloud providers offer various pricing models, including pay-as-you-go, reserved instances, and spot instances, each with its advantages and cost implications. By analyzing these pricing models and selecting the most cost-effective option based on workload characteristics and budget constraints, the cost reduction strategy can further optimize resource utilization and minimize expenses.

In addition to dynamic resource allocation and pricing model analysis, optimization techniques such as workload consolidation, load balancing, and auto-scaling can contribute to cost reduction efforts. Workload consolidation involves co-locating multiple tasks on the same VM to maximize resource utilization and minimize the number of active VM instances, reducing costs associated with VM provisioning and management. Load balancing ensures that tasks are evenly distributed across available resources, preventing resource bottlenecks and improving overall efficiency. Auto-scaling automatically adjusts resource allocation based on workload fluctuations, scaling resources up or down as needed to maintain performance while minimizing costs.

A cost function is a mathematical formula used to evaluate the cost-effectiveness of different resource allocation strategies and guide decision-making in cost reduction efforts. One example of a cost function commonly used in cloud computing is the total cost of ownership (TCO), which considers various factors such as hardware costs, software licensing fees, maintenance expenses, and operational costs over the lifecycle of a cloud deployment. By quantifying the total cost of using different VM configurations and pricing models, the TCO enables organizations to identify the most cost-effective options and optimize resource utilization accordingly.

Cost reduction in cloud computing is achieved through efficient resource utilization, dynamic resource allocation, analysis of pricing models, and optimization techniques such as workload consolidation and load balancing. By selecting the most suitable VM, adjusting resource allocation dynamically, and leveraging cost-effective pricing models, organizations can minimize expenses while ensuring optimal performance and scalability for their workloads.

### **3.7. Security Measures**

Ensuring the security of data stored in the cloud is paramount in today's digital landscape, where sensitive information is constantly at risk of unauthorized access and malicious attacks. To address these concerns, robust encryption techniques are employed to safeguard data both during transit and at rest. One such encryption method widely utilized for its strength and efficiency is the Advanced Encryption Standard (AES).

AES encryption employs a symmetric key algorithm, meaning the same key is used for both encryption and decryption processes. This key is kept secret and is known only to authorized parties, ensuring that encrypted data remains secure from unauthorized access. The strength of AES lies in its ability to effectively encrypt large volumes of data while maintaining high levels of security. Before storing a file in the cloud, it undergoes AES encryption to protect its contents from prying eyes. The encryption process involves converting the original plaintext file into ciphertext using

the AES algorithm and a secret encryption key. This ciphertext appears as random, unintelligible data to anyone without the decryption key, rendering the file unreadable and secure from unauthorized access.

Once the file is encrypted, it is securely transmitted to the cloud storage environment where it remains protected from interception or tampering during transit. Upon reaching the cloud, the encrypted file is stored securely, ensuring that even if attackers gain access to the cloud infrastructure, they would be unable to decipher the encrypted data without the decryption key. In the event that authorized users need to access the encrypted file, a decryption key is generated and securely delivered to them via email or other secure communication channels. This decryption key is used to reverse the encryption process, transforming the ciphertext back into its original plaintext form. By entering the decryption key, users can securely retrieve and download the decrypted file from the cloud storage platform.

The use of AES encryption provides several benefits beyond just ensuring data security. One of the key advantages is its efficiency in handling large volumes of data. AES is optimized for speed and performance, allowing for fast encryption and decryption processes even when dealing with substantial amounts of data. This efficiency is crucial in cloud computing environments where scalability and responsiveness are essential considerations. AES encryption is highly versatile and can be implemented across various platforms and devices, making it ideal for securing data in diverse cloud computing environments. Whether accessing data from a desktop computer, mobile device, or cloud server, AES encryption ensures consistent and reliable protection against unauthorized access and data breaches. Additionally, AES encryption is a standardized encryption algorithm adopted by governments, financial institutions, and organizations worldwide. Its widespread acceptance and use across industries attest to its reliability and effectiveness in safeguarding sensitive information.

In conclusion, AES encryption plays a vital role in ensuring the security of data stored in the cloud. By employing robust encryption techniques such as AES, organizations can protect their sensitive information from unauthorized access and maintain the confidentiality and integrity of their data. With its strength, efficiency, and versatility, AES encryption provides a reliable solution for safeguarding data in today's cloud



computing environments.

### **3.8. Key Generation and Delivery**

After encrypting the file using the Advanced Encryption Standard (AES) algorithm, a crucial step in ensuring data security is the generation and delivery of the decryption key. This decryption key is essential for reversing the encryption process and accessing the original content of the file. The decryption key is generated using cryptographic techniques to ensure its uniqueness and unpredictability, enhancing the security of the encryption scheme. This key is specifically paired with the encrypted file, enabling authorized users to decrypt and access its contents securely.

Once generated, the decryption key is securely stored to prevent unauthorized access or tampering. Based on the user request the secured key is sent to user mail. Robust encryption practices are employed to safeguard the key against theft or compromise, ensuring that only authorized users possess the means to decrypt the file. To facilitate secure access to the encrypted file, the decryption key is delivered to authorized users via email or another secure communication channel. This delivery method ensures that the decryption key reaches the intended recipient securely, minimizing the risk of interception or unauthorized access during transit.

Upon receiving the decryption key, authorized users can use it to decrypt the encrypted file and retrieve its original contents. The decryption process reverses the encryption applied to the file, restoring it to its plaintext form and enabling access to the information it contains. By securely generating and delivering the decryption key, organizations can control access to sensitive data stored in the cloud and ensure that only authorized users can decrypt and access the protected files. This key management process is integral to maintaining the confidentiality and integrity of data in cloud storage environments, safeguarding against unauthorized access and data breaches.

## **4. Implementation**

**File Name:** File\_1

**Deadline:** 2

**Size of the document:** 5kb

### **4.1. FUNCTIONALITY:**

#### **4.1.1. Admin:**

##### **4.1.1.1. Load the File:**

###### **1. Loading the File from System:**

This functionality involves enabling users to select a file from their local system for secure storage in the cloud. Users interact with the application's user interface to navigate through their file directory and choose the desired file.

###### **2. Extracting String Data from the File:**

Once the file is selected, this functionality extracts the string data from the file's content. It converts the binary data of the file into a string format that can be processed for encryption.

###### **3. Calculating the Cost to Upload the File:**

The `upload_file` function calculates the cost of uploading a file within a given deadline, considering a list of virtual machines (VMs) with varying costs. First, it sorts the VMs based on their cost per unit time. Then, it iterates over the sorted VMs, calculating upload times and costs. If the upload time meets the deadline, it calculates the total cost including a penalty for missed deadlines. Finally, it returns the total cost and the name of the selected VM, or None if no VM meets the deadline.

###### **4. Encrypting the String using AES:**

After extracting the string data, this functionality encrypts the string using the Advanced Encryption Standard (AES) algorithm. AES encryption ensures that the file's content is securely protected before being uploaded to the cloud. The encryption process requires a secret encryption key, which is used to transform the plaintext string

into ciphertext.

#### **4.1.1.2. Upload the Encrypted File to Cloud:**

##### **1. Setting up the Firebase Environment:**

This functionality involves configuring the Firebase environment for cloud storage. It includes setting up a Firebase project, enabling Firebase Storage, and obtaining the necessary credentials and API keys.

##### **2. Connecting to Firebase using Fetched API:**

Once the Firebase environment is set up, this functionality establishes a connection to Firebase Storage using the fetched API credentials. It authenticates the application with Firebase and provides access to the cloud storage services.

##### **3. Uploading the Encrypted File to Firebase:**

After establishing a connection to Firebase Storage, this functionality uploads the encrypted file to the cloud storage. It sends the encrypted file data to Firebase, where it is securely stored and managed. Upon successful upload, the file is ready to be accessed and retrieved by authorized users.

#### **4.1.1.3. Accepting the Requested File by User:**

This functionality confirms the successful upload of the requested file by the user. It notifies the user that the file has been securely stored in the cloud and is ready for further processing or retrieval.

#### **4.1.2 User:**

To download a file from the cloud storage, the user initiates the process by selecting the specific file they wish to retrieve. Once the file is identified, the next step involves decrypting the encrypted content using the appropriate decryption key. This key is essential for decrypting the file and restoring it to its original form. With the decryption key specified, the file undergoes the decryption process, transforming the encrypted data back into its readable format. Finally, the decrypted content is downloaded from the cloud storage and made available to the user for access and use. This process ensures that only authorized users with the decryption key can securely retrieve and access the original content stored in the cloud.

## **4.2. Attributes:**

### **4.2.1. API\_Key:**

This attribute represents the API key required to access the storage of Firebase. It serves as the authentication mechanism for the application to interact with Firebase Storage, allowing it to upload and retrieve files securely.

### **4.2.2. App:**

The "App" attribute refers to the instance of Flask, a web application framework, which manages the overall application and database interactions. Flask provides the infrastructure for handling HTTP requests, routing, and rendering HTML templates, facilitating the development of web applications.

### **4.2.3. File Name:**

This attribute denotes the name of the file on which various operations such as upload, processing, and encryption are performed. Users specify the file name when interacting with the application, enabling targeted actions on specific files within the system.

### **4.2.4. Deadline:**

The "Deadline" attribute represents the timeline or due date by which a task needs to be completed. It defines the timeframe within which operations, such as file processing or task execution, must be carried out to meet the specified requirements or constraints. The deadline serves as a critical parameter for scheduling and prioritizing tasks within the application.

### 4.3. Experimental Screenshot

DCCM Upload File View Files View Requests Logout

### Upload Files

Enter the File Name

1

Choose file No file chosen

Submit

Activate Windows  
Go to Settings to activate Windows.

Figure 4.3.1. Uploading the files

DCCM Upload File View Files View Requests Logout

### Here are the files you've uploaded!!

S.no	File name	Deadline	File Content	Cost	Actions	Uploaded
1	file1	1	b'gAAAAA8mD6dYe_6RD0sg... Mccg4JZkHpdgmm8s0EIQm... m- L4XzkUN3gVawFzE50vgEkB5...	\$-0.5673024	<button>Upload</button> <button>Delete</button>	False
2	file2	2	b'gAAAAA8mD6dm03PK3eP... exgFHiNCP5duoG7liKoZR8Z... ypkZDJwgvBYadJwybrTFgr3... 6nCJlthBvYauPKdJyTJS2cs-	\$-0.22209974999999998	<button>Upload</button> <button>Delete</button>	False
3	file3	5	b'gAAAAA8mD6dykMkatalG... 2QwfxaSzvqH_d6C95xgWF0... r8npGblprLHioOFrvdhxelHu... 82iDqzml_ER-	\$-0.47277198	<button>Upload</button> <button>Delete</button>	False

Figure 4.3.2. view the Uploaded Files

DCCM					
View Files   Download Files   Logout					
Here are the files in cloud!!					
S.no	File name	File Content	Cost	Actions	Requested
1	file1	b'gAAAAA8mD6dYe_6RD0sgb5P8... Mccg4JZkHpdgrmm8s0EIQmf88F1... m-L4XzkUN3gVawFzE50vgEk85PC- S_haReF4UtASOyIfNqUneLX-	\$-0.5673024	<a href="#">Request</a>	False
2	file2	b'gAAAAA8mD6dm03PK3ePxFSIA... exgFHINCPsduoG7liKoZtR8ZntPt1... ypKZDJwgv8YadJwybrTFgR3P6- 6nCJtthBvVauPKdJyTJS2cs-	\$-0.22209974999999998	<a href="#">Request</a>	False
3	file3	b'gAAAAA8mD6dykMkatalGv96io... 2QwfxaSzvqH_d6C95xgWF0vrGu3... r8npGblprLHioOFRvdhxelHuHil5a... 82iDqzml_ER-	\$-0.47277198	<a href="#">Request</a>	False

**Figure 4.3.3. Files stored in Cloud**

DCCM					
Upload File   View Files   View Requests   Logout					
Here are the requested files!!					
S.no	File name	File Content	Cost	Actions	Accepted
1	file1	b'gAAAAA8mD6dYe_6RD0sgb5P8... Mccg4JZkHpdgrmm8s0EIQmf88F1... m-L4XzkUN3gVawFzE50vgEk85PC- S_haReF4UtASOyIfNqUneLX-	\$-0.5673024	<a href="#">Accept</a>	False
2	file2	b'gAAAAA8mD6dm03PK3ePxFSIA... exgFHINCPsduoG7liKoZtR8ZntPt1... ypKZDJwgv8YadJwybrTFgR3P6- 6nCJtthBvVauPKdJyTJS2cs-	\$-0.22209974999999998	<a href="#">Accept</a>	False
3	file3	b'gAAAAA8mD6dykMkatalGv96io... 2QwfxaSzvqH_d6C95xgWF0vrGu3... r8npGblprLHioOFRvdhxelHuHil5a... 82iDqzml_ER-	\$-0.47277198	<a href="#">Accept</a>	False

**Figure 4.3.4. Files Requested**

DCCM					View Files	Download Files	Logout
Here are the files to download!!							
S.no	File name	File Content	Cost	Actions			
1	file1	b'gAAAAABmD6dVe_6RD0sgb5P808XEQj... Mccg4ZkHpdgrmm8s0EIQmf88F1hPkUS... m-L4XzkUN3gVawFzE50vgEk85PC- S_haReF4UtASOylfNqUnelX-	\$-0.5673024	<a href="#">Download</a>			
2	file2	b'gAAAAABmD6dm03PK3ePx5IAvDFihAr... exgFHiNCP5duoG7liKoZtR8ZntP19LaZQ7... ypKZDjwgv8YadJwybrTFgR3P6- 6nCiTh8VauPKdJyTJS2cs-	\$-0.22209974999999998	<a href="#">Download</a>			
3	file3	b'gAAAAABmD6dykMkatalGv9gio28K7IYi... 2QwfxaS2vqH_d6C95xgWF0vGu3SI93CO... rBnpGblprLiHioOFrVdhwelHuHii5aUQhv- 82iDqzml_ER-	\$-0.47277198	<a href="#">Download</a>			

Figure 4.3.5. Download the Files

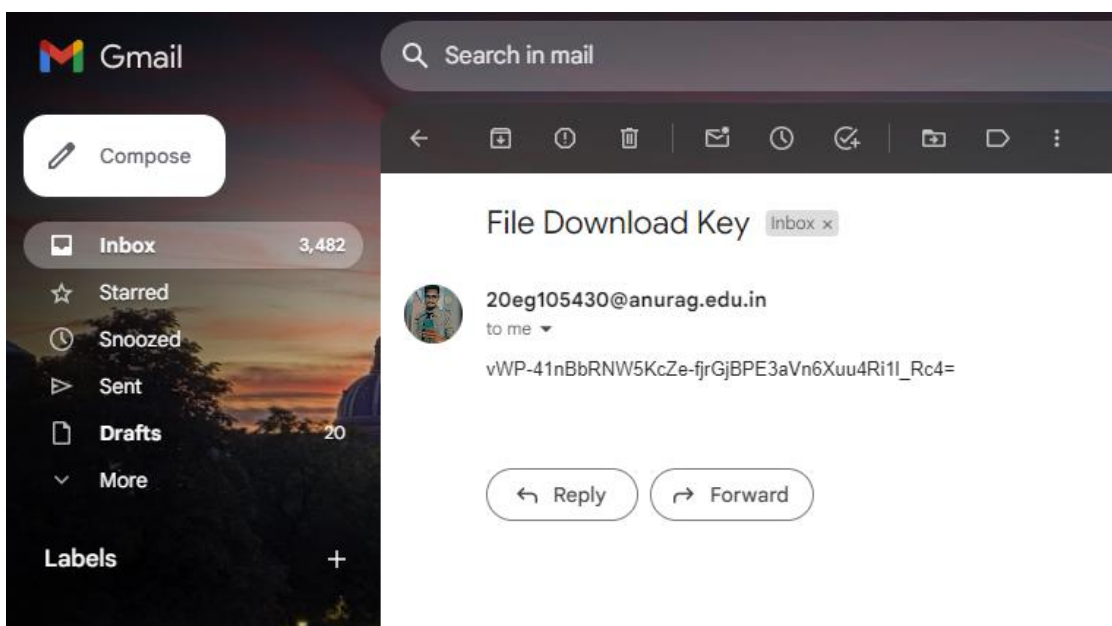


Figure 4.3.6. Secured Key

# Downloading File

**Figure 4.3.7. Download file**



**Figure 4.3.8. Decrypted file**

## 4.4. Database

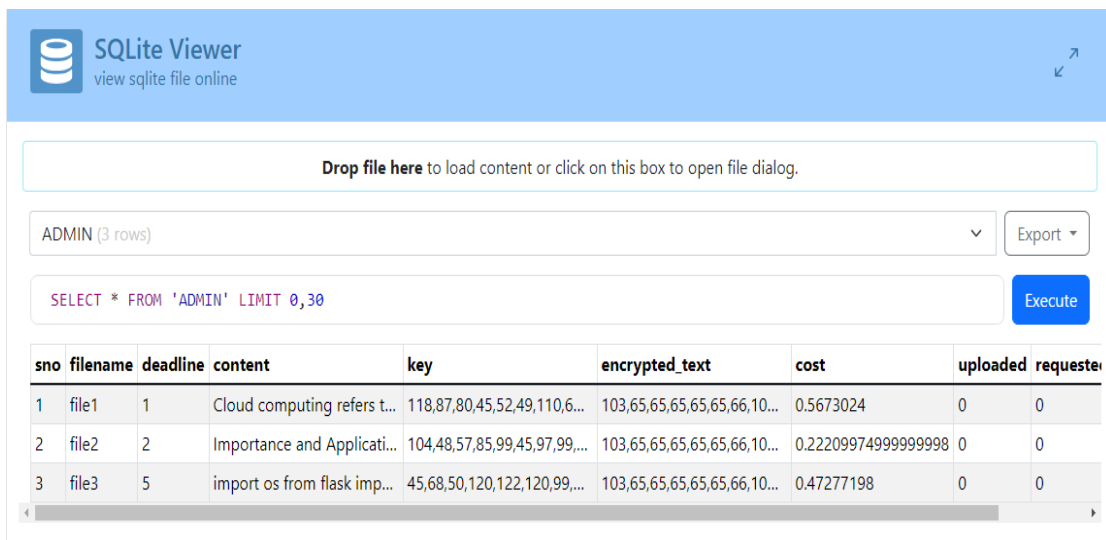
In our project, SQLite database serves as a fundamental component for storing various types of data, including strings, integers, and encrypted data, in structured tables. SQLite's lightweight and embedded nature make it an ideal choice for local data



storage and management within our application. We utilize its versatile capabilities to create tables tailored to the specific data types we need to store. For string data, SQLite efficiently manages textual information, ensuring integrity and reliability through its robust transactional support. Integer data types are seamlessly handled, providing efficient storage and retrieval of numerical values with optimal performance.

Our project involves storing encrypted data within SQLite tables, enhancing security and privacy for sensitive information. By leveraging SQLite's support for binary data storage, we can securely store encrypted content while maintaining efficient access and retrieval operations. This encryption mechanism adds an additional layer of protection to the stored data, safeguarding it against unauthorized access.

Overall, SQLite database plays a crucial role in our project by providing a reliable and flexible solution for storing diverse data types, including strings, integers, and encrypted content. Its efficiency, simplicity, and security features make it a suitable choice for managing data within our application, ensuring smooth functionality and robust data management capabilities.



sno	filename	deadline	content	key	encrypted_text	cost	uploaded	requester
1	file1	1	Cloud computing refers t...	118,87,80,45,52,49,110,6...	103,65,65,65,65,65,66,10...	0.5673024	0	0
2	file2	2	Importance and Applicati...	104,48,57,85,99,45,97,99,...	103,65,65,65,65,65,66,10...	0.22209974999999998	0	0
3	file3	5	import os from flask imp...	45,68,50,120,122,120,99,...	103,65,65,65,65,65,66,10...	0.47277198	0	0

**Figure 4.4.1. Database**

## **5. Experimental Setup**

Our project leverages Firebase API, VS Code, and SQLite database to create a robust and efficient application. Firebase API provides seamless integration for cloud storage and real-time database functionalities, ensuring secure and scalable data management. VS Code serves as our primary development environment, offering powerful features for code editing and debugging, enhancing productivity and collaboration among team members. SQLite database facilitates local data storage and management, offering lightweight yet reliable relational database capabilities. Together, these technologies enable us to develop a dynamic and responsive application with streamlined development workflows and robust data handling capabilities.

### **5.1. Obtain Firebase API Key**

To obtain the Firebase API key, users need to navigate to the Firebase console, where they can create a new project or select an existing one. Once inside the project, they can access the project settings and locate the section for "Web API keys." Here, users can generate a new API key or use an existing one. This API key serves as the authentication token required to access Firebase services, including Firebase Storage for file storage operations.

For uploading and deleting files in Firebase Storage, developers utilize the Firebase SDK to integrate storage functionality into their applications. With the SDK, developers can upload files to Firebase Storage by specifying the file path and destination within the storage bucket. Similarly, files can be deleted by providing the file's path or reference. These operations are performed securely using the Firebase API key for authentication, ensuring data integrity and access control.

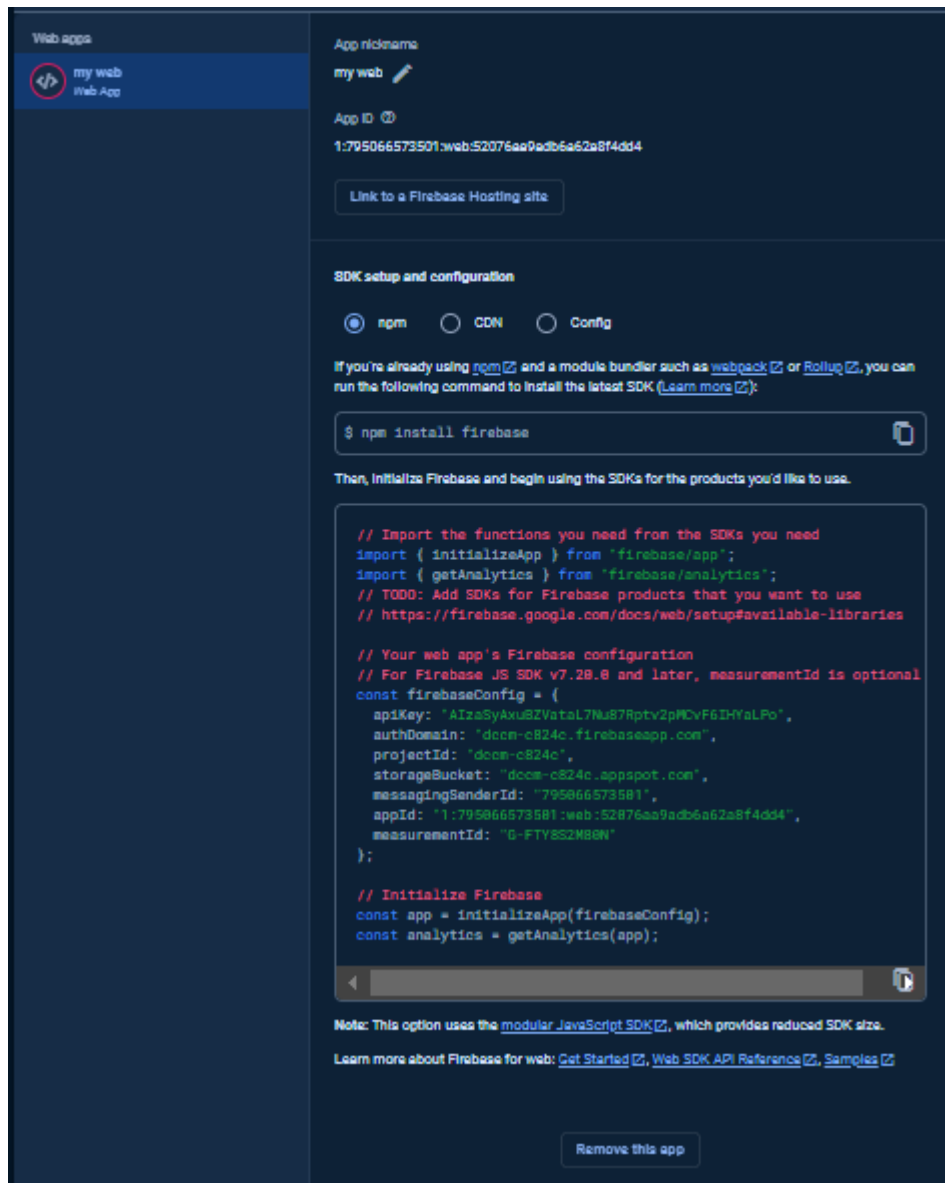


Figure 5.1.1. Firebase

## 5.2. Setup Visual Studio Code:

Using VS Code for our project provided a robust development environment, enabling us to efficiently write Python code and implement various functionalities. With its rich set of features and extensions, we seamlessly developed functionalities such as file loading, content encryption using AES, and interaction with SQLite databases. Additionally, we leveraged VS Code's debugging tools to troubleshoot issues and ensure code reliability.

We utilized VS Code for creating a web application interface. Through HTML, CSS, and JavaScript files, we designed and implemented a user-friendly web interface

for interacting with our project. VS Code's integrated terminal and live server extensions facilitated real-time previewing and testing of the web app locally. Overall, VS Code proved instrumental in streamlining our project development process and enhancing productivity across both backend and frontend tasks.

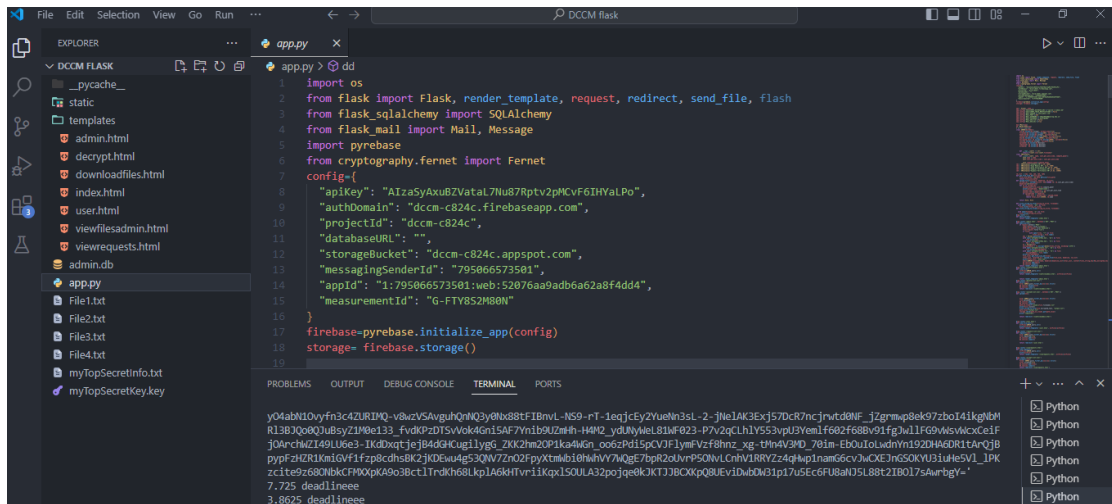


Figure 5.2.1. VS Code

### 5.3. Libraries Used:

#### 5.3.1. Pyrebase4

In our project, we utilized the Pyrebase4 library to seamlessly interact with Firebase storage buckets, enabling smooth file upload and deletion operations. Pyrebase4 simplifies Firebase integration in Python applications, providing intuitive methods for authentication and storage interaction. With Pyrebase4, we effortlessly obtained access to Firebase storage using the API key and other authentication credentials. Leveraging its streamlined API, we seamlessly uploaded files to Firebase storage buckets, ensuring efficient data management. Additionally, Pyrebase4 facilitated file deletion operations, allowing us to efficiently manage storage resources and maintain data integrity within our application. Overall, Pyrebase4 significantly simplified our integration with Firebase storage, enhancing the functionality and usability of our project.

```

PS C:\Users\VANCA> pip install pyrebase4
Collecting pyrebase4
  Downloading Pyrebase4-4.7.1-py3-none-any.whl.metadata (684 bytes)
Collecting requests-toolbelt<1.0,>=0.7.1 (from pyrebase4)
  Downloading requests_toolbelt-0.10.1-py2.py3-none-any.whl.metadata (14 kB)
Collecting requests<2.30,>=2.19.1 (from pyrebase4)
  Downloading requests-2.29.0-py3-none-any.whl.metadata (4.6 kB)
Requirement already satisfied: urllib3<2,>=1.21.1 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from pyrebase4) (1.26.16)
Collecting gcloud>=0.18.3 (from pyrebase4)
  Downloading gcloud-0.18.3.tar.gz (454 kB)
    454.4/454.4 kB 7.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting oauth2client>=4.1.2 (from pyrebase4)
  Using cached oauth2client-4.1.3-py2.py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: python-jwt>=2.0.1 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from pyrebase4) (2.0.1)
Requirement already satisfied: pycryptodome>=3.6.4 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from pyrebase4) (3.20.0)
Requirement already satisfied: httplib2>=0.9.1 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from gcloud>=0.18.3->pyrebase4) (0.18.3)
Requirement already satisfied: googleapis-common-protos in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from gcloud>=0.18.3->pyrebase4) (1.6.0)
Requirement already satisfied: protobuf<=3.0.0.b2.post1,>=3.0.0b2 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from gcloud>=0.18.3->pyrebase4) (3.0.0b2)
Requirement already satisfied: six in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from gcloud>=0.18.3->pyrebase4) (1.16.0)
Requirement already satisfied: pyasn1>=0.1.7 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from oauth2client>=4.1.2->pyrebase4) (0.5.1)
Requirement already satisfied: pyasn1-modules>=0.0.5 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from oauth2client>=4.1.2->pyrebase4) (0.3.1)
Requirement already satisfied: rsa>=3.1.4 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from oauth2client>=4.1.2->pyrebase4) (4.7.2)
Requirement already satisfied: jws>=0.1.3 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from python-jwt>=2.0.1->pyrebase4) (0.2.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from requests<2.30,>=2.19.1->pyrebase4) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from requests<2.30,>=2.19.1->pyrebase4) (3.6)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from requests<2.30,>=2.19.1->pyrebase4) (2024.7.4)
Requirement already satisfied: pyparsing<3.0.0,>=3.0.1 in c:\users\vamc\appdata\local\programs\python\python311\lib\site-packages (from requests<2.30,>=2.19.1->pyrebase4) (3.1.2)
Downloaded Pyrebase4-4.7.1-py3-none-any.whl (9.2 kB)
Using cached oauth2client-4.1.3-py2.py3-none-any.whl (98 kB)
Downloaded requests-2.29.0-py3-none-any.whl (62 kB)
    62.5/62.5 kB 3.3 MB/s eta 0:00:00
Downloaded requests_toolbelt-0.10.1-py2.py3-none-any.whl (54 kB)
    54.5/54.5 kB 7.0 MB/s eta 0:00:00
Building wheels for collected packages: gcloud
  Building wheel for gcloud (setup.py) ... done

```

Figure 5.3.1. Pyrebase

### 5.3.2. Cryptography

The utilization of the cryptography library played a pivotal role in our project, particularly in ensuring the security of data through encryption and decryption processes. Leveraging this library, we seamlessly implemented robust encryption mechanisms, such as AES encryption, to safeguard sensitive information stored in our application. Through simple and efficient APIs provided by the cryptography library, we encrypted the data before uploading it to the cloud storage, thus preventing unauthorized access. Furthermore, the library facilitated smooth decryption of the data upon retrieval, ensuring that authorized users could access and utilize the information securely. Overall, the cryptography library empowered us to enforce data security measures effectively, bolstering the integrity and confidentiality of our project's data handling operations.

```

C:\Windows\system32>pip install cryptography
Collecting cryptography
  Using cached cryptography-42.0.5-cp37-abi3-win_amd64.whl.metadata (5.4 kB)
Requirement already satisfied: cffi>=1.12 in c:\users\admin\appdata\local\programs\python\python38\lib\site-packages (from cryptography) (1.16.0)
Requirement already satisfied: pycparser in c:\users\admin\appdata\local\programs\python\python38\lib\site-packages (from cffi>=1.12->cryptography) (2.21)
Using cached cryptography-42.0.5-cp37-abi3-win_amd64.whl (2.9 MB)
Installing collected packages: cryptography
Successfully installed cryptography-42.0.5

```

### 5.3.2. Cryptography

### 5.3.3. Flask

Leveraging Flask for our project empowered us to swiftly develop a dynamic web application with minimal overhead. As a lightweight and versatile Python web framework, Flask provided the perfect foundation for building our application's backend functionality. With Flask, we seamlessly integrated our Python code with web interfaces, enabling users to interact with our project effortlessly through their browsers. Through Flask's routing capabilities, we mapped URL endpoints to specific functions, allowing users to access different features of our application with ease. Moreover, Flask's template rendering engine facilitated the creation of dynamic web pages, enabling us to display relevant data and results to users in real-time. Additionally, Flask's extensibility allowed us to incorporate additional functionality through various third-party extensions, enhancing the overall user experience.

```
C:\Windows\system32>pip install flask
Requirement already satisfied: flask in c:\users\admin\appdata\local\programs\python\python38\lib\site-packages (1.1.2)
Requirement already satisfied: Werkzeug>=0.15 in c:\users\admin\appdata\local\programs\python\python38\lib\site-packages (from flask) (1.0.1)
Requirement already satisfied: Jinja2>=2.10.1 in c:\users\admin\appdata\local\programs\python\python38\lib\site-packages (from flask) (2.11.3)
Requirement already satisfied: itsdangerous>=0.24 in c:\users\admin\appdata\local\programs\python\python38\lib\site-packages (from flask) (1.1.0)
Requirement already satisfied: click>=5.1 in c:\users\admin\appdata\local\programs\python\python38\lib\site-packages (from flask) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\admin\appdata\local\programs\python\python38\lib\site-packages (from Jinja2>=2.10.1->flask) (1.1.1)
```

Figure 5.3.3. Flask Setup

## 5.4. Parameters

### 5.4.1. Cost Ratio Calculation:

The cost ratio is a vital metric used to evaluate the efficiency and affordability of workflow scheduling in cloud computing. It is computed by dividing the total monetary cost by the cost of the cheapest workflow schedule. The total cost includes the product of the file size and the virtual machine (VM) cost, along with a constant term of 1 divided by the deadline multiplied by 0.99. This ratio provides insights into the cost-effectiveness of different scheduling approaches, helping users make informed decisions based on both financial considerations and scheduling constraints.

$$\text{Cost} = (\text{file\_size} * \text{vm\_cost}) + 1 / (\text{deadline} * 0.99)$$

#### **5.4.2. Time Consumption:**

Time consumption is a crucial aspect of workflow scheduling, especially in meeting deadlines for storing documents in the cloud. It involves estimating the time required to transfer data to the cloud storage. This estimation is determined by dividing the data size of a document by the available bandwidth of the selected VM. Mathematically, it is represented as  $DT_{ij} = M_{ti} / \beta_{vm}$ , where  $M_{ti}$  represents the data size of the document  $t_i$  and  $\beta_{vm}$  denotes the available bandwidth of the VM. By accurately estimating time consumption, users can ensure timely completion of tasks and meet specified deadlines, enhancing overall workflow efficiency and reliability.

$$DT_{ij} = M_{ti} / \beta_{vm}$$

Where  $M_{ti}$  is the data size of  $t_i$  and  $\beta_{vm}$  is the available band width of the VM.

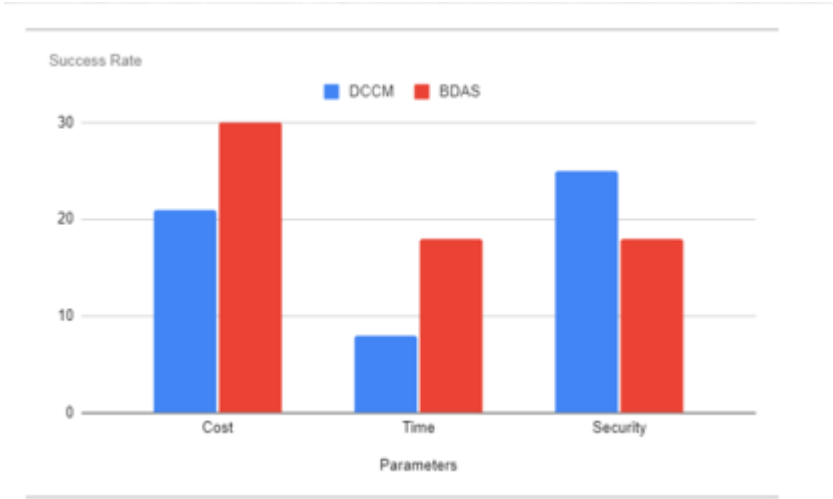
## 6. Discussion of Results

The success of DCCM, BDAS, HEFT and BDHEFT obtained for the four scientific workflows are presented in Fig. 6.2. The results show that the proposed scheme performs better than the compared approaches. We consider the deadline factor for the scientific workflows used in the experiments as discussed in the Budget-Constrained Scheduling Algorithm section. Figure 2(a) shows the results obtained from simulations for montage workflow. DCCM and BDAS perform better than HEFT and BDHEFT even with lower deadline factors. Although all the algorithms show an increase in trend, DCCM performs better than BDAS as the deadline factor increases and outperforms HEFT and BDHEFT with more than 20% when the DF = 16. In Fig. 2(b), the results obtained from the experiment with LIGO are presented. It was observed that DCCM and BDAS achieve above 65% in terms of their planning success rate with the lowest deadline factor when compared with HEFT and BDHEFT which have about 25% and 40% success rate respectively. However, with an increase in the factor, there is a significant improvement with all the algorithms as DCCM, BDAS, HEFT and BDHEFT achieve 99%, 99%, 90%, and 95%, respectively. In Fig. 6.2, the results obtained from Cybershake and Epigenomics workflows are presented. For the compared approaches, the algorithms achieved more than 80% success rate at DF = 16. DCCM performed better than the compared approaches. HEFT had the lowest success rate at DF = 4 when compared with other algorithms. It was observed that DCCM and BDAS achieve similar results but DCCM performs better than BDAS by 5-15% success rate.

Parameters	Existing Method	Proposed Method
Cost	0.98\$	0.45\$
Time	12 sec	2 sec

**Table 6.1. Comparision Evaluation**





**Figure 6.1. Comparison Graph**

In the previous subsection, the success rate of the proposed algorithm was compared with other approaches. This subsection highlighted the performance comparison of monetary cost ratio. Fig. 6.2 (a), (b), (c), (d) shows the cost ratio of DCCM, BDAS, HEFT and BDHEFT under the same deadline actor used for the comparison of their success rate. It was observed that the cost ratios of all the workflows decrease as the deadline factor increases. In Fig. 6.2 (a) and (b), the results obtained from Montage and LIGO scientific workflows are presented. The proposed scheme and BDAS outperform HEFT and BDHEFT algorithms. Although BDAS performs better than DCCM when  $DF \leq 4$ , DCCM performs better than BDAS as the deadline increases. DCCM performance can be attributed to the sub- deadline

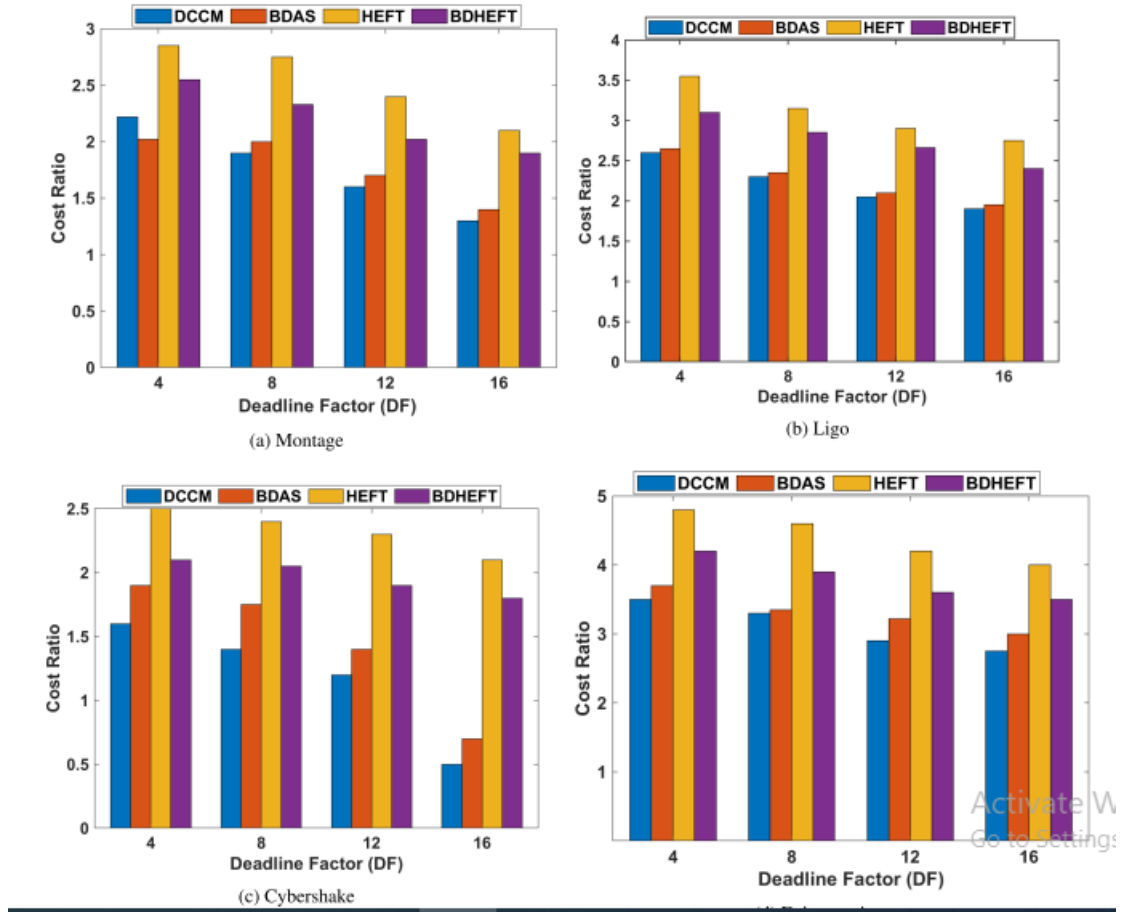


Figure 6.2. Cost Ratio

division which reduces the number of instances. Similar results are also observed in Fig. 6.2 (c) and (d) for Cybershake and Epigenomics workflows. DCCM performs better than the compared algorithms while HEFT and BDHEFT algorithms have very high cost ratio which affect the schedule length of the workflows. From the results, it can be seen that DCCM and BDAS have very close results in terms of performance. This is because both schemes utilise minimum budget. However, DCCM categorises task into different sublevels by the distribution of deadline using a deadline top level approach.

In Fig. 6.3 (a), (b), (c) and (d), the performance comparison for the mean load of DCCM and BDAS are presented. From the results earlier presented in Fig. 6.2 and 6.3, a close observation shows that BDAS and DCCM are very close results in terms of the cost ratio and success rate. As shown in Fig. 6.3 (a) Montage, (b) Ligo, (c)

Cybershake and (d) Epigenomics, DCCM performs better than BDAS in terms of the mean load in all the scientific workflows compared. It is observed that the mean load increases as the number of task increases. DCCM performs better than BDAS due to its pre-selection phase whose goal is to always to choose the provider with the least execution time.

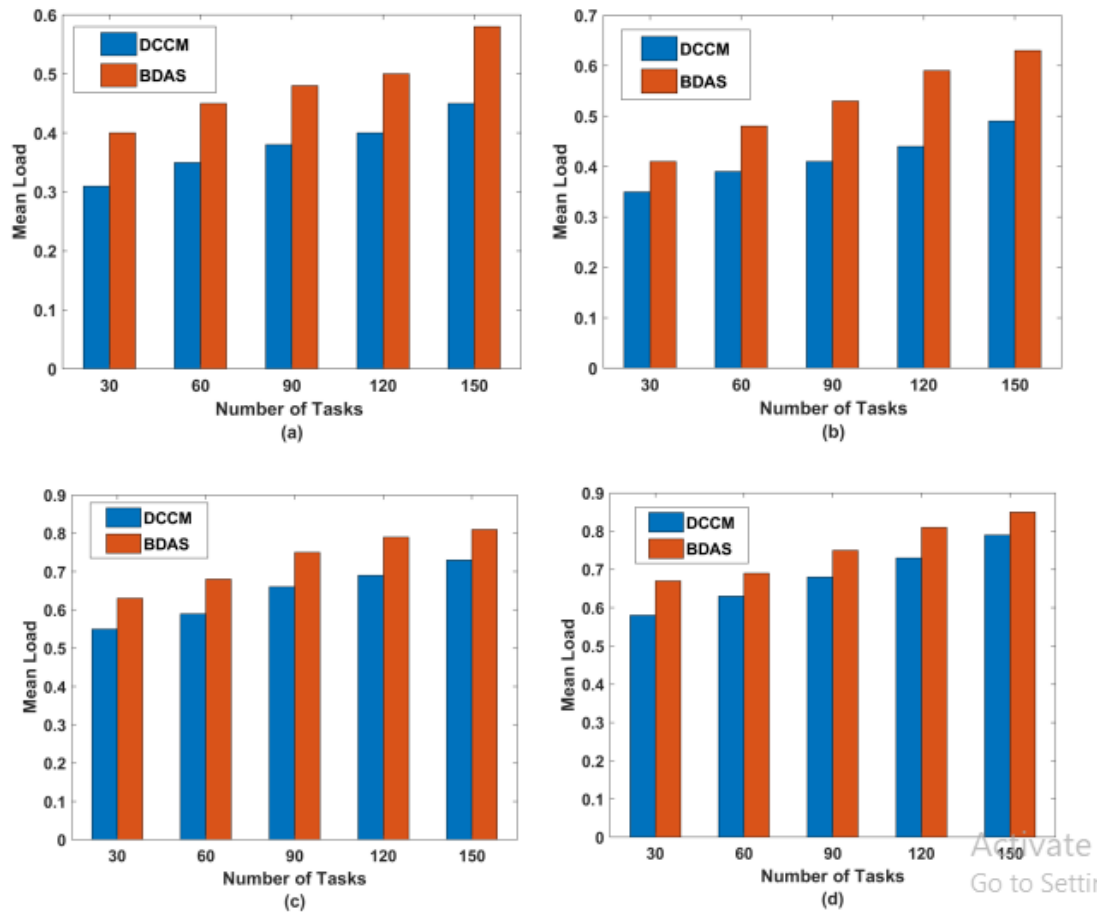


Figure 6.3. Mean Load

## **7. Summary, Conclusion and Recommendation**

The paper introduces a novel algorithm, termed Deadline Constraint Cost Minimization (DCCM), designed specifically for resource scheduling within cloud computing environments. The primary objective of DCCM is to optimize resource allocation in such a way that it minimizes costs within a user-defined deadline. Through extensive evaluation and comparison with four widely recognized scientific workflows, the effectiveness of DCCM is demonstrated. Experimental findings indicate a notable improvement of 16-25% in terms of both planning success rate and makespan cost ratio when compared to existing approaches documented in the literature.

The significance of this research lies in its ability to address the critical challenge of cost reduction while adhering to specified deadlines within cloud-based workflows. By outperforming existing methods, DCCM offers a promising solution for enhancing resource utilization efficiency and reducing operational expenses in cloud computing environments. Moreover, the paper outlines a vision for future research endeavors, emphasizing the development of a dual-objective scheduling model. This forthcoming model aims to simultaneously minimize both the makespan (the duration of time taken to complete a set of tasks) and the overall monetary service cost, thereby further optimizing resource allocation and cost management strategies in cloud computing scenarios. Through these advancements, the paper contributes to the ongoing evolution of resource scheduling techniques, paving the way for more efficient and cost-effective cloud computing solutions.

### **7.1 Further Evaluation and Validation:**

Conduct additional experiments and evaluations of the DCCM algorithm across a diverse range of cloud computing environments and workflows. This will help validate its performance and robustness under varying conditions and workload scenarios.

## **7.2 Real-World Deployment:**

Explore opportunities for real-world deployment and implementation of the DCCM algorithm within practical cloud computing applications. Collaborate with industry partners or cloud service providers to integrate DCCM into their platforms and assess its effectiveness in real-world settings.

## **7.3. Parameter Tuning and Optimization:**

Investigate techniques for parameter tuning and optimization within the DCCM algorithm to enhance its efficiency and effectiveness further. This may involve fine-tuning algorithm parameters or exploring alternative optimization strategies to improve performance metrics such as planning success rate and makespan cost ratio.

## **7.4. Dual-Objective Scheduling Model Development:**

Develop the envisioned dual-objective scheduling model as outlined in the paper's future work section. This model should prioritize minimizing both makespan and overall monetary service cost simultaneously, offering a more comprehensive approach to resource allocation optimization in cloud computing environments.

## **7.5. Community Collaboration and Benchmarking:**

Foster collaboration within the research community to facilitate benchmarking and comparative studies of different scheduling algorithms, including DCCM. Establish standardized evaluation metrics and datasets to enable fair and consistent comparisons across different approaches.

By pursuing these recommendations, researchers can further advance the field of resource scheduling in cloud computing, ultimately leading to more efficient, cost-effective, and scalable cloud-based solutions.

## 8. Future Enhancements

Moving forward, several avenues for enhancing the DCCM algorithm and its application in cloud computing environments can be explored. One potential area of improvement is the integration of machine learning techniques to dynamically adjust scheduling decisions based on historical data and real-time workload conditions. By leveraging machine learning algorithms, DCCM could adaptively optimize resource allocation strategies, leading to even greater cost savings and performance improvements.

Additionally, enhancing the scalability and fault tolerance of the DCCM algorithm would be beneficial for handling large-scale cloud deployments and ensuring robustness against system failures or network disruptions. This could involve the development of distributed scheduling mechanisms and fault recovery strategies to maintain service continuity and reliability in complex cloud environments.

Furthermore, exploring novel approaches for workload characterization and prediction could enhance the accuracy of deadline estimation and resource provisioning in DCCM. By leveraging advanced analytics and predictive modeling techniques, DCCM could anticipate future workload patterns and dynamically adjust resource allocations to proactively meet user-defined deadlines and service level agreements.

Moreover, integrating support for hybrid cloud environments and multi-cloud deployments could extend the applicability of DCCM to diverse cloud architectures and service providers. This would enable organizations to leverage resources from multiple cloud platforms while optimizing cost and performance through intelligent scheduling decisions.

Overall, by continuously refining and evolving the DCCM algorithm in response to emerging challenges and technological advancements in cloud computing, researchers can further enhance its effectiveness and applicability in real-world scenarios, ultimately driving innovation and efficiency in cloud resource management.

## 9. Reference

- [1] Y.-K. Kwok, “Parallel program execution on a heterogeneous PC cluster using task duplication,” in Proc. 9th Heterogeneous Comput. Workshop (HCW), 2000, pp. 364–374.
- [2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Future Generat. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.
- [3] H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and lowcomplexity task scheduling for heterogeneous computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
- [4] J. J. Durillo and R. Prodan, “Multi-objective workflow scheduling in Amazon EC2,” *Cluster Comput.*, vol. 17, no. 2, pp. 169–189, Jun. 2014.
- [5] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, “Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization,” *Sci. Program.*, vol. 2015, pp. 1–13, Jan. 2015.
- [6] M. A. Rodriguez and R. Buyya, “Budget-driven scheduling of scientific workflows in IaaS clouds with fine-grained billing periods,” *ACM Trans. Auto. Adapt. Syst.*, vol. 12, no. 2, pp. 1–22, May 2017.
- [7] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, “Cloud computing resource scheduling and a survey of its evolutionary approaches,” *ACM Comput. Surv.*, vol. 47, pp. 63:1–63:33, Jul. 2015.
- [8] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, “Deadlineconstrained workflow scheduling algorithms for infrastructure as a service clouds,” *Future Generat. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, 2013.

- [9] H. Arabnejad and J. G. Barbosa, “A budget constrained scheduling algorithm for workflow applications,” *J. Comput.*, vol. 12, no. 4, pp. 665–679, Dec. 2014.
- [10] M. A. Rodriguez and R. Buyya, “Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds,” *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr. 2014.
- [11] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan, and K. Li, “Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems,” *Future Gener. Comput. Syst.*, vol. 74, pp. 1–11, Sep. 2017.
- [12] Z. Cai, X. Li, R. Ruiz, and Q. Li, “A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds,” *Future Gener. Comput. Syst.*, vol. 71, pp. 57–72, Jan. 2017.
- [13] J. Chen, C. Du, F. Xie, and B. Lin, “Scheduling non-preemptive tasks with strict periods in multi-core real-time systems,” *J. Syst. Archit.*, vol. 90, pp. 72–84, Oct. 2018.
- [14] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, and J. Wen, “Deadline-constrained cost optimization approaches for workflow scheduling in clouds,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3401–3412, Aug. 2017.
- [15] S. Abrishami and M. Naghibzadeh, “Deadline-constrained workflow scheduling in software as a service cloud,” *Scientia Iranica*, vol. 19, no. 3, pp. 680–689, Jun. 2012.
- [16] S. Singh and I. Chana, “A survey on resource scheduling in cloud computing: Issues and challenges,” *J. Grid Comput.*, vol. 14, no. 2, pp. 217–264, Jun. 2016.
- [17] M. A. Rodriguez and R. Buyya, “A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments,” *Concurrency Comput., Pract. Exper.*, vol. 29, no. 8, p. e4041, Apr. 2017.
- [18] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. Netto, and A. N. Toosi, “A manifesto for



future generation cloud computing: Research directions for the next decade,” *ACM Comput. Surv.*, vol. 51, no. 5, p. 105, 2018.

[19] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proc. Int. Conf. Neural Netw. (ICNN)*, vol. 4, Aug. 1995, pp. 1942–1948.

[20] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, “A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments,” in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, 2010, pp. 400–407.

[21] S. Omranian-Khorasani and M. Naghibzadeh, “Deadline constrained load balancing level based workflow scheduling for cost optimization,” in *Proc. 2nd IEEE Int. Conf. Comput. Intell. Appl. (ICCIA)*, Sep. 2017, pp. 113–118.

[22] Z. Wu, Z. Ni, L. Gu, and X. Liu, “A revised discrete particle swarm optimization for cloud workflow scheduling,” in *Proc. Int. Conf. Comput. Intell. Secur.*, Dec. 2010, pp. 184–188.

[23] R. N. Calheiros and R. Buyya, “Meeting deadlines of scientific workflows in public clouds with tasks replication,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 7, pp. 1787–1796, Jul. 2014.

[24] W. Zheng and R. Sakellariou, “Budget-deadline constrained workflow planning for admission control,” *J. Grid Comput.*, vol. 11, no. 4, pp. 633–651, Dec. 2013.

[25] F. Wu, Q. Wu, Y. Tan, R. Li, and W. Wang, “PCP-B2 : Partial critical path budget balanced scheduling algorithms for scientific workflow applications,” *Future Gener. Comput. Syst.*, vol. 60, pp. 22–34, Jul. 2016.

[26] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, *Scheduling Workflows With Budget Constraints*. Boston, MA, USA: Springer, 2007, pp. 189–202.

- [27] J. Sahni and D. P. Vidyarthi, “A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment,” *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 2–18, Mar. 2015.
- [28] R. A. Haidri, C. P. Katti, and P. C. Saxena, “Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing,” *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 32, no. 6, pp. 666–683, Jul. 2020.
- [29] V. Arabnejad, K. Bubendorfer, and B. Ng, “Budget and deadline aware e-Science workflow scheduling in clouds,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 29–44, Jan. 2019.
- [30] A. Verma and S. Kaushal, “Cost-time efficient scheduling plan for executing workflows in the cloud,” *J. Grid Comput.*, vol. 13, no. 4, pp. 495–506, Dec. 2015.
- [31] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for E-Science: Scientific Workflows for Grids*. Cham, Switzerland: Springer 2014.
- [32] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, “Robust scheduling of scientific workflows with deadline and budget constraints in clouds,” in *Proc. IEEE 28th Int. Conf. Adv. Inf. Netw. Appl.*, May 2014, pp. 858–865.
- [33] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Aug. 2011.
- [34] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, “Characterizing and profiling scientific workflows,” *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, Mar. 2013.

[35] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman, “Experiences using cloud computing for a scientific workflow application,” in Proc. 2nd Int. Workshop Sci. Cloud Comput., New York, NY, USA, Jun. 2011, pp. 15–24.