**NANYANG TECHNOLOGICAL UNIVERSITY**

**SC4079 – Final Year Project**



**Interim Report: Evaluating Job Scheduling Algorithms in Clouds**

**Dixit Ayushman U2121836F**

# Table of Contents

[Table of Contents 2](#_Toc188885211)

[1. Introduction 5](#_Toc188885212)

[1.1. Project Overview 5](#_Toc188885213)

[1.2. Goals 5](#_Toc188885214)

[1.3. Motivation 6](#_Toc188885215)

[1.4. Problem Definition 7](#_Toc188885216)

[1.5. Contributions 7](#_Toc188885217)

[2. Related Works 8](#_Toc188885218)

[2.1. Traditional Scheduling Algorithms 8](#_Toc188885219)

[2.2. Cloud Specific Scheduling Algorithms 9](#_Toc188885220)

[2.3. Machine Learning Algorithms 9](#_Toc188885221)

[2.4. Gaps and Limitations 10](#_Toc188885222)

[3. Experiment Configuration 11](#_Toc188885224)

[3.1. Google Cloud dataset 11](#_Toc188885225)

[3.2. Tasks details 12](#_Toc188885226)

[3.3. Instances details 13](#_Toc188885227)

[3.4. Simulation setup 14](#_Toc188885228)

[3.5. Metrics tracked 15](#_Toc188885229)

[3.6. Key assumptions and considerations 16](#_Toc188885230)

[4. Stratus Algorithm evaluation 18](#_Toc188885231)

[4.1. Stratus Algorithm explanation and expectations 19](#_Toc188885232)

[4.2. Stratus Algorithm implementation 21](#_Toc188885233)

[4.3. Best fit and first fit algorithm implementation for comparison 24](#_Toc188885234)

[4.4. Stratus simulation and comparisons 27](#_Toc188885235)

[5. Q-learning (to be done individually in next part of FYP) 33](#_Toc188885236)

[6. Evaluation of algorithms (to be done as a team in next part of FYP) 35](#_Toc188885237)

[7. Proposed Project Timeline 37](#_Toc188885238)

[8. References 38](#_Toc188885239)

**Abstract**

The exponential growth of cloud computing has intensified the challenge of efficient job scheduling across distributed systems. This research addresses the critical problem of resource allocation in cloud environments, where complex job scheduling directly impacts performance and cost-effectiveness. The study focuses on analysing and evaluating job scheduling algorithms using real-world job trace data from Google Cloud, with a primary objective of optimising resource utilisation and minimising operational expenses.

The research employs a comprehensive approach, initially examining the Stratus Algorithm and subsequently exploring advanced machine learning-based scheduling techniques. By conducting a detailed comparative analysis, the project aims to identify more efficient scheduling strategies that can dynamically adapt to varying computational workloads. The methodology involves simulation and performance evaluation of different algorithms using authentic cloud platform data.

Key contributions include a systematic assessment of existing scheduling approaches and an innovative exploration of machine learning techniques for resource allocation. The findings provide insights into potential improvements in cloud resource management, offering valuable perspectives for optimising computational efficiency and reducing infrastructure costs in large-scale distributed computing environments.

# Introduction

## Project Overview

With the rapid growth of cloud computing, efficient job scheduling has become a critical challenge in managing resources across large-scale distributed systems. In cloud environments, jobs often require parallel execution across multiple virtual machines (VMs) or containers distributed across different data centres. The competition for limited resources in such environments makes it difficult to allocate them efficiently and fairly.

This project aims to analyse and evaluate job scheduling algorithms using real-world job trace data from cloud platforms, specifically Google Cloud. The primary focus will be on assessing the cost-effectiveness and resource efficiency of various scheduling algorithms. The analysis will begin with the Stratus Algorithm, a popular cost efficient algorithm for optimising job scheduling in cloud environments. Additionally, the project will explore advanced techniques, including machine learning-based algorithms, to determine if they can further enhance efficiency, resource utilisation, and cost optimisation. Finally, the project will conclude with a discussion and comparison of simulation results from the machine learning-based algorithms, along with a proposal for potential improvements to enhance scheduling efficiency.

## Goals

The primary goals of this project are:

1. **Understand Cloud Job Scheduling**: Explore how jobs are scheduled in cloud computing environments and the specific challenges involved, such as resource limitations, parallel tasks, and dynamic workloads.
2. **Evaluate the Stratus Algorithm**: Benchmark the performance of the Stratus Algorithm, focusing mainly on cost-effectiveness.
3. **Compare Other Algorithms**: Evaluate and compare other scheduling algorithms to see how they perform relative to Stratus.
4. **Propose Potential Improvements**: Based on the results, identify gaps in the existing algorithms and, if possible, suggest modifications or new approaches to improve scheduling efficiency.

## Motivation

Job scheduling is a key aspect of cloud computing because it directly impacts the cost and performance of applications running on cloud infrastructure. According to a Gartner report, cloud services spending was projected to be $675.4 billion in 2024 [1], with 30% of it being wasted[2]. In cloud environments, where services are billed based on resource usage, poorly designed scheduling algorithms can result in increased costs or inefficient use of resources.

Optimised job scheduling can reduce infrastructure costs and improve application performance . As cloud providers continue to scale their operations, there is a critical need for more effective algorithms that can handle complex, dynamic workloads and optimise costs and resource utilisation.

This projects aims to achieve this by firstly studying the performance of a well-known cost efficient algorithm – Stratus. Beyond which, another exciting opportunity in this project is the potential to apply machine learning techniques, which can learn and adapt to real-time job patterns. These techniques have shown promise in other areas and could potentially lead to more intelligent and adaptive scheduling strategies in cloud computing.

By analysing and comparing different scheduling algorithms, this project aims to find solutions that not only reduce costs but also improve the overall efficiency of cloud systems.

## Problem Definition

The rapid expansion of cloud computing has introduced significant challenges in resource management, particularly in job scheduling across distributed systems. With jobs requiring parallel execution on virtual machines or containers, the efficient allocation of resources becomes critical to ensuring cost-effectiveness and system performance. However, existing scheduling algorithms often struggle to adapt to dynamic workloads and optimise resource utilisation, leading to inefficiencies and increased operational costs. Addressing these challenges requires an in-depth analysis of scheduling algorithms, such as the Stratus Algorithm, and the exploration of innovative approaches, including machine learning-based techniques, to identify more adaptive and efficient solutions for cloud job scheduling.

## Contributions

This project contributes to the field by providing a comprehensive evaluation of job scheduling algorithms in cloud environments, using real-world Google Cloud job trace data. It benchmarks the performance of the Stratus Algorithm while exploring the potential of machine learning-based techniques to enhance scheduling efficiency and resource utilisation. Finally, it proposes potential improvements to existing algorithms, aiming to reduce costs and optimise performance in dynamic and large-scale cloud computing systems.

# Related Works

Job scheduling in cloud environments has been an active area of research due to the complexity of resource management in distributed systems. Numerous algorithms have been developed over the years to address the challenges associated with resource allocation, job prioritisation, and cost efficiency. In this section, we review some of the key works related to job scheduling algorithms, focusing on both traditional and more recent approaches that aim to optimise cloud environments.

## Traditional Scheduling Algorithms

Early job scheduling algorithms focused on simple heuristics and were primarily designed for grid computing environments, which later laid the groundwork for cloud-based scheduling. One of the most widely studied algorithms is First Come First Serve (FCFS), where jobs are executed in the order they arrive. While FCFS is straightforward and easy to implement, it often leads to inefficient resource usage and increased job completion times due to its lack of prioritisation.

Another common approach is the Shortest Job First (SJF) algorithm, which prioritises jobs with the smallest execution time. SJF aims to reduce overall job completion time, but it has limitations, especially in cloud environments, where estimating job execution time is not always reliable.

## Cloud Specific Scheduling Algorithms

With the advent of cloud computing, new scheduling algorithms were developed to address the unique characteristics of the cloud, such as dynamic resource allocation and the pay-per-use pricing model. The Stratus Algorithm, which is central to this project, was specifically designed for optimising cost-effectiveness in cloud environments. [3] Stratus focuses on dynamically adjusting resource allocation based on job runtime estimates, which can lead to significant cost savings by optimising the use of cloud resources. However, the performance of Stratus heavily relies on accurate job runtime predictions, which can be a challenge in real-world environments where job characteristics vary significantly.

In addition to Stratus, several other cloud-specific algorithms, such as Min-Min and Max-Min, have been explored. Min-Min selects the smallest jobs to be executed first, aiming to free up resources quickly, while Max-Min prioritises larger jobs. Both approaches have their advantages, but they tend to focus on job completion times rather than cost optimisation, which is a crucial factor in cloud environments.

## Machine Learning Algorithms

In recent years, machine learning has been applied to job scheduling to develop adaptive algorithms that can learn from past data and improve decision-making over time. Reinforcement Learning (RL), in particular, has gained attention for its ability to optimise complex scheduling scenarios.

These techniques has been demonstrated to improve both resource utilisation and cost efficiency compared to traditional heuristic-based algorithms.[4] However, one of the challenges with machine learning-based approaches is the computational cost and training time required, especially when using large-scale datasets like the ones provided by Google and Alibaba.

## Gaps and Limitations

## Despite significant advancements in job scheduling algorithms, existing approaches still face critical limitations. Traditional algorithms such as FCFS and SJF lack adaptability to dynamic and heterogeneous cloud environments, often resulting in inefficient resource utilisation and increased costs. While cloud-specific algorithms like Stratus offer cost optimisation, they are heavily dependent on accurate runtime predictions, which are difficult to achieve in real-world scenarios with unpredictable workloads. Additionally, machine learning-based algorithms show promise but require substantial computational resources and training time, limiting their scalability and practicality. This project addresses these gaps by evaluating existing methods using real-world trace data and exploring adaptive machine learning techniques to develop more efficient and scalable solutions for cloud job scheduling.

# Experiment Configuration

The performance of the Stratus Algorithm, along with other scheduling algorithms, is analysed through simulation-based experiments. Key metrics such as cost, memory usage, CPU utilisation, and the number of instances acquired are assessed. This section details the experimental setup, including the datasets, tools, and configuration parameters necessary to carry out the simulations and evaluate the algorithms effectively.

## Google Cloud dataset

To simulate a realistic cloud environment for this experiment, a dataset containing real-world job traces and virtual machine (VM) usage is utilised. Specifically, the open-source Google Cluster Data from 2011, derived from the Google cluster management system known as Borg, serves as the foundation for this study. This dataset captures 29 days of workload information from May 2011, encompassing a cluster of approximately 12,500 machines.

The dataset is hosted on Google Cloud Storage and can be accessed using the GSUtil tool. It is organised into multiple compressed files, with each file in CSV format using Unix-style line endings (ASCII LF). The rows in the CSV files are sorted by the event timestamp, which is always stored in the first column. These files lack headers, requiring a detailed understanding of the schema to interpret the data effectively.

Among the various tables available in the dataset, such as job\_events, task\_events, machine\_events, machine\_attributes, task\_constraints, and task\_usage, this experiment focuses specifically on the task\_events and machine\_events tables. The task\_events table provides critical information about task-level events, including timestamps, job and task identifiers, resource requests (CPU, memory, and disk), and task priorities. The machine\_events table captures details about machine-level events, such as timestamps, machine IDs, event types, platform details, and resource capacities (CPU and memory). These two tables form the core data used to evaluate scheduling algorithms under simulated conditions.

By leveraging this dataset, the experiment is designed to mimic a real-world cloud computing scenario, enabling the analysis of scheduling algorithms in a highly realistic and large-scale environment.

## Tasks details

The task events table provides information about tasks that are either actively running or eligible to run within the trace data. Each task's status is defined in the event\_type column, which specifies key task lifecycle events such as submission and completion. Tasks are uniquely identified using a combination of job\_ID and task\_index, ensuring clear traceability for each task within the dataset.

For this analysis, task runtimes were calculated by measuring the duration between task submission (event\_type=0) and task completion (event\_type=4). Timestamp values of 0 were excluded from the dataset, as they represent events that occurred prior to the start of the trace and are considered invalid for this study. The calculation process also involved converting timestamps from microseconds to seconds and capturing essential resource request details, such as CPU and memory. By merging submission and completion events for each task, the final dataset includes task runtimes alongside resource requests and submission timestamps, providing a detailed view of task execution patterns. This cleaned and processed data ensures a reliable foundation for evaluating scheduling algorithms.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 1: Sample task data after calculations

## Instances details

The machine events table provides detailed information about the machines present in the trace, including their resource capacities and lifecycle events. Each machine is typically represented by one or more entries. These entries include details about machine capacities, such as CPU and memory, normalised across different dimensions. Although some of the machine's capacity may be reserved for the operating system in practice, this complexity is not accounted for in this analysis to simplify the computations.

To further analyse machine instances, unique instance types were extracted based on their CPU and memory capacities. Using Google Cloud Platform (GCP) pricing standards, the price of each instance type was calculated. GCP pricing values were set at $0.03899 per CPU hour and $0.005226 per GB of memory per hour. The cost for each instance type was determined by summing the CPU and memory costs. These prices were then normalised by dividing each instance's price by the maximum price in the dataset, resulting in a "normalised price" that allows for relative comparisons between instance types.

The resulting dataset provides a sorted list of unique instance types, including their CPU and memory capacities, along with their normalised prices. This information is critical for understanding the cost implications of different machine configurations and their suitability for resource scheduling algorithms.

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 2: Sample instances data after calculations

## Simulation setup

A custom event-based workload simulator was developed to utilise workload traces as input, replicating the behaviour and decision-making process of the scheduler. This simulator enables a detailed analysis of the scheduler's performance by closely examining its decision-making logic and extracting relevant metrics. These metrics provide valuable benchmarks for comparing the Stratus algorithm's performance against other scheduling approaches.

The simulator was implemented in Python, chosen for its balance of performance and usability. Python's Pandas library was used to efficiently handle large datasets, performing tasks such as data transformation, analysis, and visualisation with ease. Additionally, its seamless integration with visualisation tools like Matplotlib enhanced the clarity of results. Python's simplicity, readability, and widespread adoption make it highly maintainable and an excellent choice for projects requiring complex data processing and analysis.

## Metrics tracked

To evaluate the performance of the scheduling algorithms in each simulation, several key metrics were tracked. These metrics provide insights into the efficiency and cost-effectiveness of the scheduling strategies:

**Price**: The total cost incurred during the simulation, calculated based on the resource usage (CPU and memory) and normalised prices of the machine instances. This metric helps assess the cost-effectiveness of the scheduling algorithm.

**CPU Utilisation**: The percentage of CPU capacity utilised across all instances during the simulation. Tracking CPU utilisation ensures that resources are being effectively allocated and not underutilised.

**Memory Utilisation**: The percentage of memory capacity utilised across all instances. Similar to CPU utilisation, this metric evaluates how efficiently the scheduler allocates memory resources.

**Number of Instances Used**: The total number of machine instances utilised during the simulation. This metric helps gauge the scalability of the scheduling algorithm and its ability to minimise the number of machines required to handle the workload.

By analysing these metrics, the simulation provides a comprehensive view of the scheduler's performance, highlighting areas of strength and identifying opportunities for improvement.

## Key assumptions and considerations

The simulation relies on several key assumptions and considerations to simplify the analysis and focus on evaluating scheduling performance:

1. **Infinite Instance Availability**: It is assumed that an infinite number of machine instances can be acquired as needed. This allows the simulation to focus solely on scheduling efficiency without being constrained by resource availability.
2. **Exclusion of Timestamp 0 Entries**: Tasks or events with a timestamp of 0 are excluded from the analysis, as these represent activities that occurred prior to the start of the workload trace and are considered invalid for this study.
3. **Exclusion of Killed Tasks**: For runtime calculations, only tasks that reach a finished state are considered. Tasks that were killed or did not complete are excluded to ensure accurate runtime assessments.
4. **Known Task Requirements**: The CPU and memory requirements, as well as the runtime of each task, are assumed to be known beforehand. This simplifies the scheduling process by eliminating the need to predict resource requirements dynamically.
5. **Static Resource Prices**: The prices for CPU and memory usage are assumed to remain constant throughout the simulation, based on predefined Google Cloud Platform (GCP) pricing standards.
6. **No Overhead Consideration**: The simulation does not account for additional overhead costs, such as instance startup times, data transfer costs, or operational delays, focusing solely on task scheduling and execution.

These assumptions streamline the simulation process and provide a controlled environment for evaluating scheduling algorithms. While they simplify the problem, they may differ from real-world constraints, and their potential impact on results is considered in the analysis.

# Stratus Algorithm evaluation

The Stratus Algorithm is designed to optimise resource utilisation and reduce cloud costs, making it an ideal baseline for this project. The evaluation of Stratus will focus on the following steps:

1. Simulation Setup: We will simulate a cloud environment using real job trace data from production systems, such as the Google Cluster Data. This simulation will allow us to replicate real-world cloud workloads and evaluate the performance of the Stratus Algorithm under realistic conditions.
2. Performance Metrics: The key metric for evaluating Stratus will be cost-effectiveness, which is crucial in cloud environments where resources are charged on a pay-per-use basis. Other secondary metrics will include:
   * Resource Utilisation: Measuring how efficiently Stratus allocates and uses available resources (CPU, memory, etc.).
   * Instances used: Analysing how instances are needed to optimise cost and resource utilisation.
3. Comparative Analysis: Stratus will be compared with other traditional scheduling algorithms, such as First Fit and Best Fit to highlight its advantages and disadvantages. By running these algorithms on the same job trace data, we can compare their performance in terms of cost savings, resource efficiency, and overall cloud workload management.
4. Results Interpretation: The results of the Stratus Algorithm will be thoroughly analysed, focusing on its strengths in cost optimisation as well as any weaknesses.. This analysis will set the foundation for comparing Stratus with other more advanced algorithms.

## Stratus Algorithm explanation and expectations

The Stratus Algorithm is a cost-aware cluster scheduler designed specifically for managing batch job execution in virtual clusters on public Infrastructure as a Service (IaaS) platforms. Stratus prioritises cost optimisation by efficiently packing tasks onto available instances using job runtime estimates. This dense packing approach ensures that resources are optimally utilised or promptly released to minimise costs. Stratus integrates seamlessly with resource managers, such as Kubernetes or YARN, which enforce its scheduling decisions while Stratus focuses on task assignment and instance scaling.

#### How Stratus Works

Jobs submitted to the virtual cluster follow a well-defined lifecycle, where Stratus makes scheduling decisions based on task runtime estimates and resource requirements:

1. Jobs are submitted to the Resource Manager (RM) with task details, including resource requests and runtimes.
2. Task requests are forwarded to the Stratus scheduler, which comprises two key components: the **packer** and the **scaler**.
3. The **packer** tightly allocates tasks to available instances using runtime binning, a strategy that groups tasks and instances based on similar remaining runtimes. Tasks are placed through two phases:
   * **Up-packing Phase**: Tasks are assigned to instances in their runtime bin or progressively larger bins.
   * **Down-packing Phase**: Tasks unable to find a suitable instance undergo a secondary attempt to locate smaller bins with sufficient resources.
4. If tasks cannot be packed, the **scaler** acquires new instances by evaluating cost-effective options based on task resource requirements and instance prices.

Stratus iteratively selects the most cost-efficient instance and assigns unscheduled tasks until all are accommodated. The scaling decisions aim to minimise costs while maintaining sufficient resources for pending tasks.

#### **Expectations**

The Stratus Algorithm is expected to influence key metrics in the following ways:

* **Price**: By densely packing tasks and minimising the number of instances required, Stratus should significantly reduce overall cloud costs. Its cost-aware scaling process ensures that only the most cost-effective instances are acquired.
* **Memory Utilisation**: Stratus is designed to allocate memory resources efficiently, with runtime binning ensuring optimal grouping of tasks. We anticipate high memory utilisation due to its packing strategy.
* **CPU Utilisation**: Similar to memory, CPU resources are expected to be utilised effectively, with tasks distributed across instances to maximise available CPU capacity.
* **Number of Instances**: Stratus is expected to use more instances than Best Fit and First Fit due to its proactive scaling mechanism, which acquires new instances when tasks cannot be packed into existing ones. Unlike Best Fit and First Fit, which attempt to fit tasks into available resources without prioritising cost or runtime alignment, Stratus focuses on optimising resource utilisation and cost by densely packing tasks using runtime binning. This approach can lead to earlier scaling decisions, as Stratus prioritises minimising resource fragmentation and aligning runtimes, potentially requiring additional instances to achieve its cost and efficiency goals.

The algorithm’s proactive approach to resource management makes it an ideal candidate for optimising workload execution in cloud environments, and we expect it to outperform traditional schedulers in terms of cost and resource efficiency. The evaluation will confirm whether Stratus meets these expectations while identifying areas for potential improvement.

## Stratus Algorithm implementation

The implementation of the Stratus Algorithm involves several key components to simulate the packing and scaling process for scheduling tasks in a cloud environment. Below is an explanation of the implementation with pseudocode for its main parts:

**1. Initialisation**

The StratusScheduler class initialises with a list of available instance types and maintains task and instance bins. Each instance and task is tracked along with metrics such as CPU and memory usage.

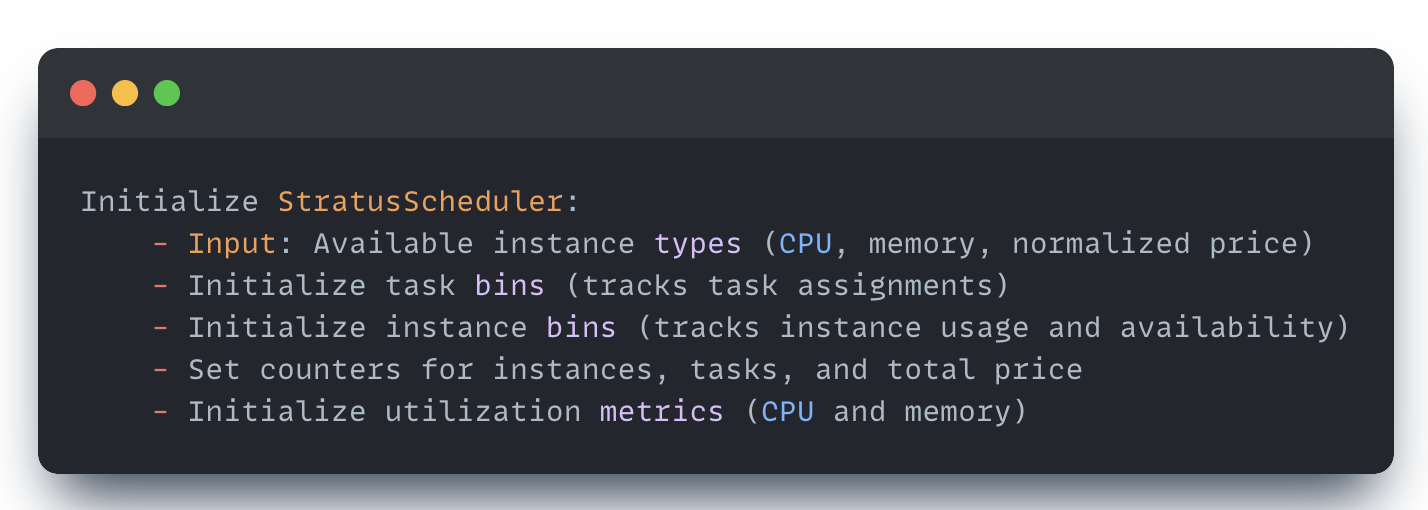


Figure 3: Pseudocode for initialisation

**2. Task and Instance Freeing**

Tasks and instances are periodically checked for expiration based on their runtime and timestamp. Expired tasks are removed, and resources are freed from their respective instances. Instances are also freed once all tasks are completed.

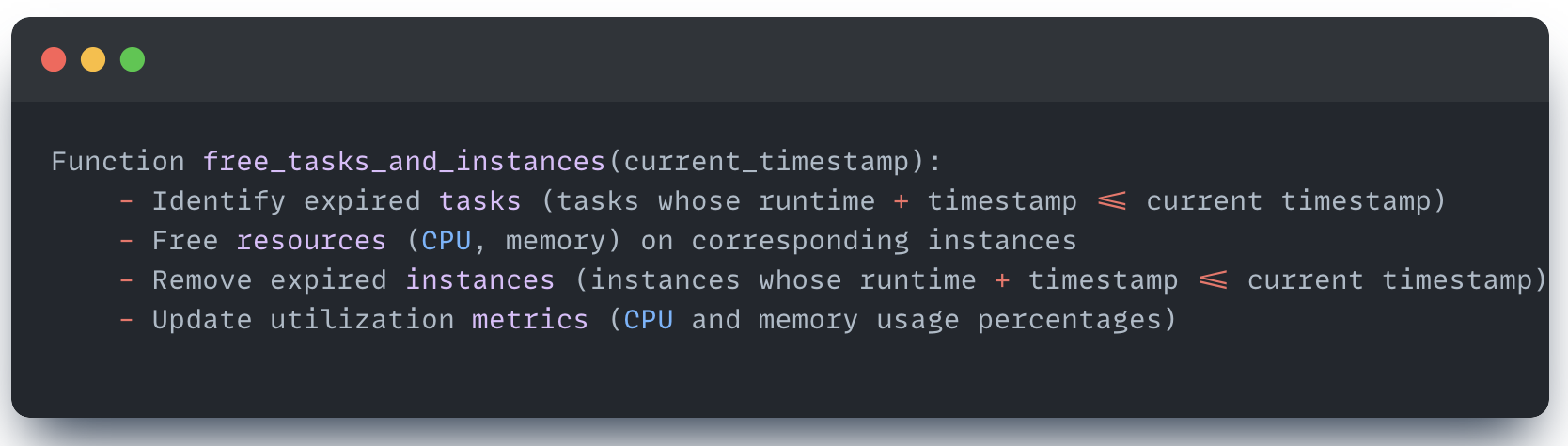


Figure 4: Pseudocode for freeing instances and tasks

**3. Runtime binning**

Tasks and instances are grouped into bins based on their runtime using exponential binning. This ensures efficient task placement and resource utilisation.

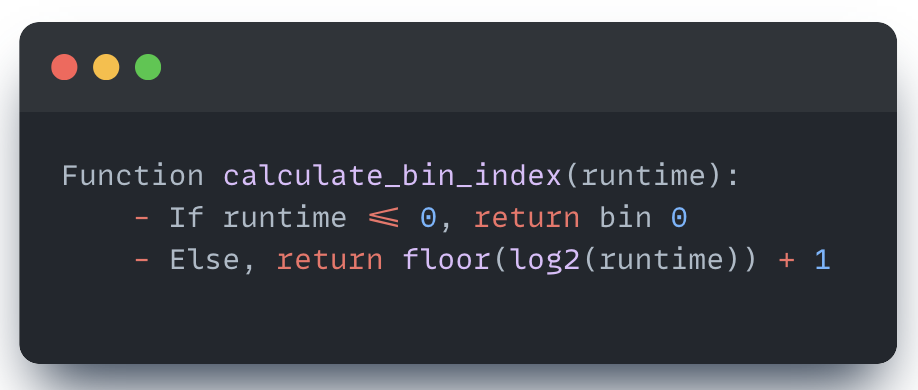


Figure 5: Pseudocode for runtime binning

**4. Task packing (packer)**

The packer attempts to assign tasks to existing instances in two phases:

1. **Up-packing Phase**: Tasks are packed into instances within the same bin or higher bins, prioritising instances with the closest matching runtime.
2. **Down-packing Phase**: Tasks are packed into instances in lower bins if no suitable instance exists in the current or higher bins.

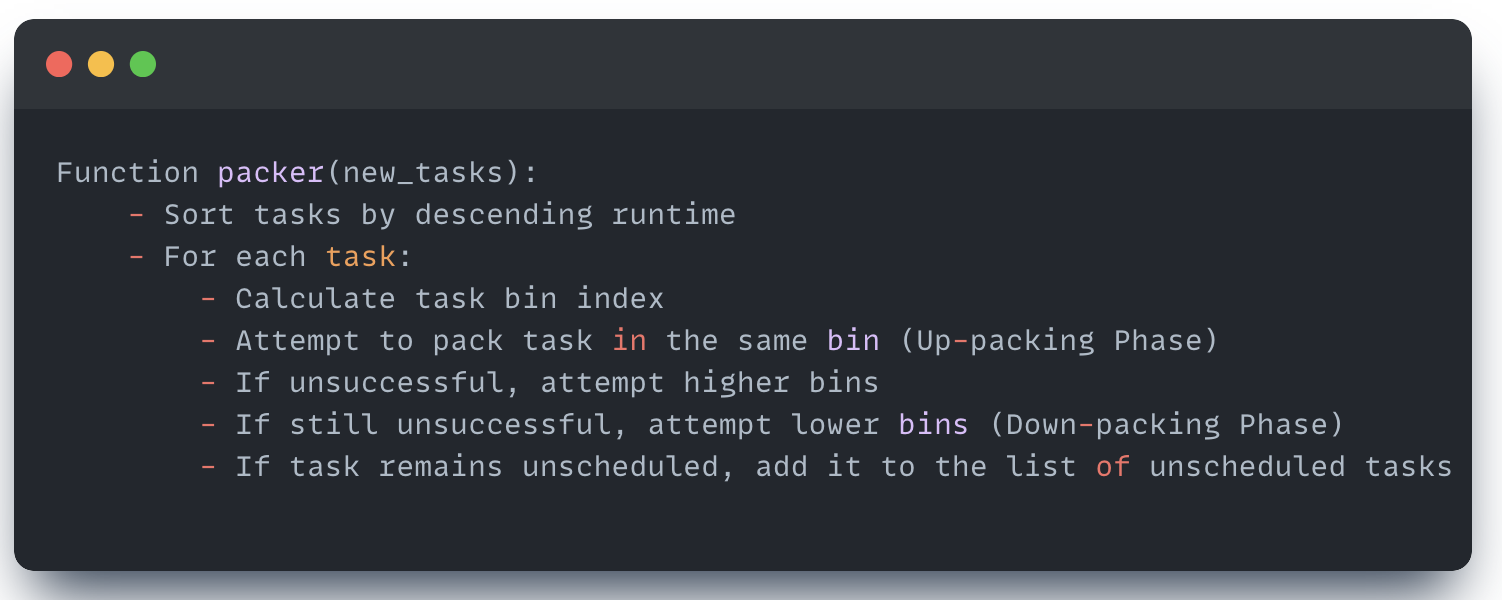


Figure 6: Pseudocode for task packing

**5. Scaling (scaler)**

When tasks cannot be packed into existing instances, the scaler acquires new instances. The scaler evaluates task groups and instance types to determine the most cost-effective combination for unscheduled tasks.

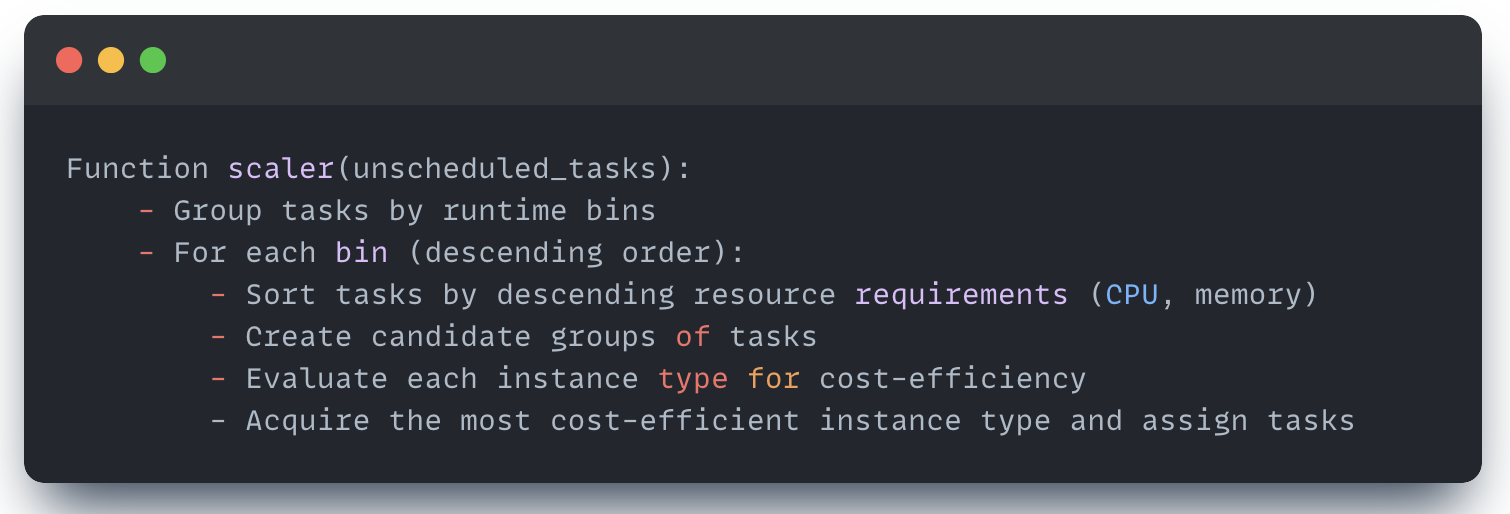


Figure 7: Pseudocode for scaling

**6. Task Assignment**

Tasks are assigned to instances based on resource availability, and instance usage metrics are updated accordingly.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 8: Pseudocode for task assignment

**7. Key metrics and tracking**

The implementation tracks the following key metrics:

**CPU Utilisation**: The percentage of CPU resources used across all active instances.

**Memory Utilisation**: The percentage of memory resources used across all active instances.

**Total Cost**: Accumulated cost of acquired instances based on their runtime and pricing.

**Number of Instances Acquired**: Tracks the total number of instances provisioned during the simulation.

This implementation encapsulates the Stratus Algorithm’s core functionality, emphasising cost-aware task packing and dynamic instance scaling. The pseudocode provides a high-level overview of how Stratus efficiently schedules tasks to optimise resource utilisation and minimise costs in a cloud environment.

## Best fit and first fit algorithm implementation for comparison

To compare the Stratus Algorithm with simpler scheduling strategies, Best Fit and First Fit algorithms were implemented. Both algorithms aim to allocate tasks to available instances but differ in their task-to-instance assignment logic. Below is a brief description of their implementation with pseudocode highlighting their unique aspects.

**Best Fit Algorithm**

The Best Fit algorithm assigns tasks to the instance with the closest remaining runtime that can accommodate the task's resource requirements (CPU and memory). This approach minimises resource fragmentation and ensures better runtime alignment across tasks.

Key Logic:

* Identify eligible instances that can fit the task.
* Assign the task to the instance with the smallest difference between the instance's remaining runtime and the task's runtime.

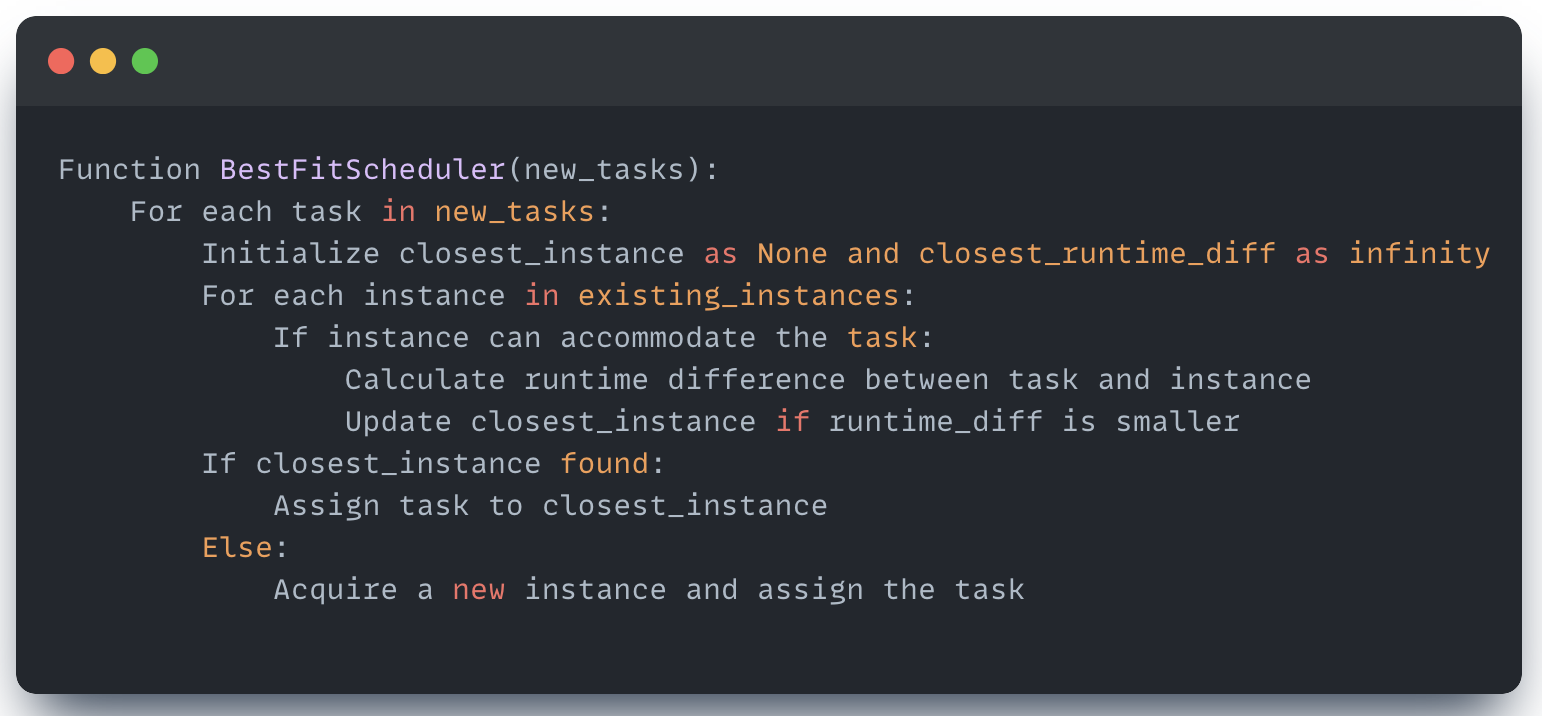


Figure 9: Pseudocode for Best fit algorithm

**First Fit Algorithm**

The First Fit algorithm assigns tasks to the first available instance that can accommodate them. To improve accuracy and avoid bias from instance order, the instance list is randomised before each assignment. This ensures the evaluation remains fair and more representative of diverse scenarios.

Key Logic:

Iterate through instances in a randomised order to identify the first instance that can fit the task.

If no existing instance is suitable, acquire a new instance.

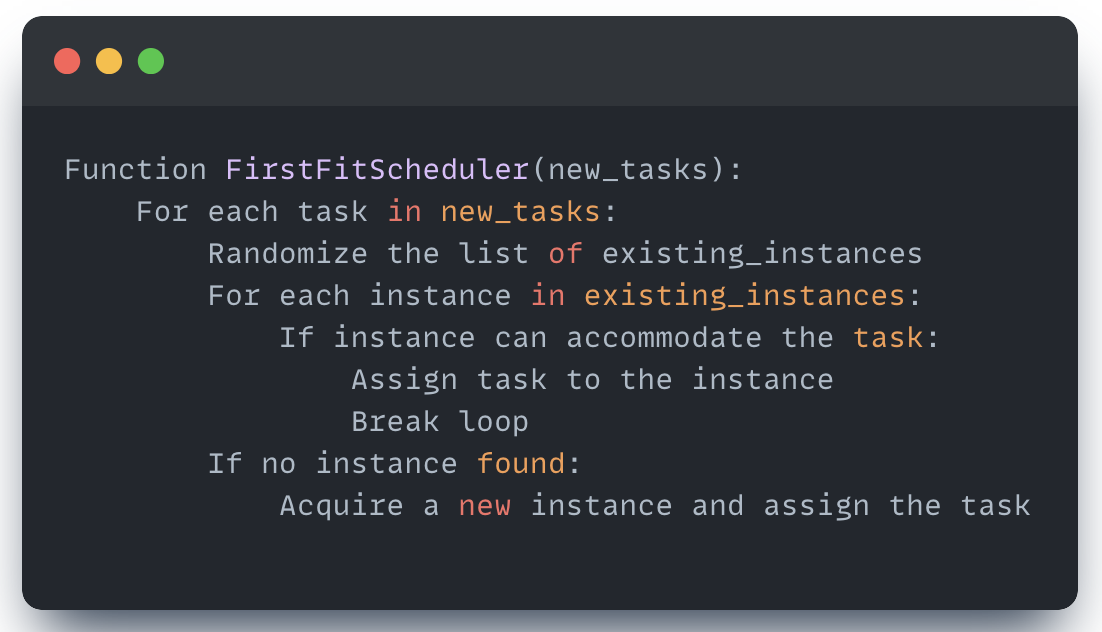


Figure 10: Pseudocode for First fit algorithm

The Best Fit algorithm focuses on optimal placement by selecting the most suitable instance based on runtime alignment, while the First Fit algorithm prioritises simplicity and speed by assigning tasks to the first viable instance. The randomisation step in First Fit ensures unbiased evaluation, especially when simulating diverse cloud environments. Both implementations serve as baseline algorithms for comparing resource utilisation, cost efficiency, and scheduling performance with Stratus.

## Stratus simulation and comparisons

The Stratus Algorithm was evaluated using a simulation environment built with 38,362 real-world tasks from the Google Cluster dataset. Available instance types were modeled based on standard configurations, including their CPU, memory capacities, and normalised prices. The simulation aimed to compare Stratus against baseline algorithms, such as Best Fit and First Fit, by tracking key metrics: total price, CPU utilisation, memory utilisation, and total instances used.

**Simulation results**

The outcomes of the simulation are presented below, showcasing the performance of Stratus compared to Best Fit and First Fit schedulers:

Total Price

The cumulative cost incurred by each scheduler was tracked over time. Stratus demonstrated a significant reduction in total cost, achieving approximately 60% lower costs compared to First Fit and 70% lower costs compared to Best Fit by the end of the simulation. This emphasises its effectiveness in cost-aware resource utilisation and task packing strategies.

A graph showing different schedulers

AI-generated content may be incorrect.

Figure 11: Total Price vs. Timestamp for Different Schedulers.

Average CPU Utilisation

Stratus maintained higher CPU utilisation throughout the simulation, demonstrating its efficiency in allocating computational resources compared to Best Fit and First Fit.

A graph showing the average cpu usage

AI-generated content may be incorrect.

Figure 12: Average CPU Utilisation Over Time.

Average Memory Utilisation

Similarly, Stratus showed the highest memory utilisation, reflecting its dense task-packing approach to minimise resource wastage.

A graph showing the average memory

AI-generated content may be incorrect.

Figure 13: Average Memory Utilisation Over Time.

Total Instances Used

Stratus acquired more instances than the baseline algorithms, as its runtime-aware binning mechanism prioritised runtime alignment and efficiency over limiting the number of instances.

A graph of different schedulers

AI-generated content may be incorrect.

Figure 14: Total Instances vs. Timestamp for Different Schedulers.

**Key Observations and explanations**

1. Cost Efficiency

* Stratus consistently achieved the lowest total cost compared to Best Fit and First Fit. This outcome is attributed to its cost-aware approach, which prioritises efficient resource utilisation and the selection of the most cost-effective instances during scaling.
* Best Fit and First Fit, while simpler, lacked mechanisms to optimise costs and often resulted in higher cumulative expenses due to less efficient task packing.

2. CPU Utilisation

* Stratus maintained the highest average CPU utilisation, peaking early in the simulation and remaining steady over time. This reflects its dense task-packing strategy, which minimises idle CPU resources by aligning tasks with similar runtimes on instances.
* Best Fit achieved moderate CPU utilisation, as it aims to match tasks with instances based on runtime alignment. However, it was less efficient than Stratus in leveraging available capacity.
* First Fit exhibited the lowest CPU utilisation, as its simpler approach often led to suboptimal task assignments without fully utilising instance capacities.

3. Memory Utilisation

* Memory utilisation followed a similar trend to CPU utilisation, with Stratus achieving the highest efficiency. This is due to its ability to pack tasks tightly into instances based on both runtime and resource requirements.
* Best Fit again demonstrated moderate memory utilisation, while First Fit underperformed due to its reliance on the first available instance, which often led to underutilised resources.

4. Total instances used

* Stratus required the highest number of instances compared to Best Fit and First Fit. This result aligns with its focus on runtime alignment and resource efficiency, which occasionally necessitated acquiring additional instances to ensure optimal task placement.
* Best Fit and First Fit used fewer instances but at the cost of higher resource underutilisation and total price.

The results underscore the strengths of the Stratus Algorithm in cost optimisation and resource utilisation, particularly in scenarios with diverse task requirements and runtime variations. However, its higher instance acquisition highlights a trade-off between minimising costs and maximising efficiency versus reducing the number of instances used. Best Fit and First Fit demonstrated simpler, less resource-intensive strategies but fell short in achieving the same level of efficiency and cost-effectiveness as Stratus.

This analysis reinforces the suitability of the Stratus Algorithm for cost-sensitive cloud environments that prioritise resource optimisation over instance count.

Moving forward, we will explore algorithms that could potentially outperform Stratus, with a focus on machine learning-based approaches. These algorithms will be evaluated to determine if they can match Stratus's strengths while addressing its limitations, such as higher instance acquisition.

# Q-learning (to be done individually in next part of FYP)

Q-learning algorithm offers a flexible and adaptive approach for job scheduling in cloud environments. [4]

Rough Steps for Implementation:

1. Preprocessing the Dataset: The first step is to preprocess the job trace data to a format compatible with Q-learning simulations. This will involve:
   * Extracting relevant features from the dataset, such as job submission times, resource requests (CPU, memory), and job execution status (success or failure).
   * Structuring the data to represent different states of the cloud environment, such as the available resources and the queue of jobs waiting for execution.
2. State and Action Definitions:
   * State: The state in this context will represent the current status of the cloud environment, including the number of jobs in the queue, available resources (CPU, memory, VMs), and the progress of running jobs.
   * Action: The actions will involve scheduling decisions, such as selecting which job to schedule next and determining the amount of resources to allocate for each job.
3. Simulation of Q-Learning on the Dataset:
   * A simulation environment will be set up where the Q-learning algorithm interacts with the pre-processed job trace data.
   * The algorithm will use the Q-table to store the state-action pairs, gradually updating the table as it receives feedback from the environment in the form of rewards.
   * The simulation will run multiple iterations, allowing the algorithm to explore different scheduling strategies and optimise its policy for minimising resource costs and improving job completion times.
4. Reward Structure: The reward function will guide the Q-learning algorithm toward optimal scheduling decisions. It will be designed to reward:
   * Cost Savings: Positive rewards for actions that minimise the overall cost of resource usage.
   * Efficient Job Scheduling: Rewards for reducing job completion time and utilising resources effectively.
   * Resource Optimisation: Incentives for actions that minimise idle resources or avoid over-provisioning.

This reward structure will help the algorithm focus on cost-effectiveness, which aligns with the overall goal of the project.

1. Evaluation and Comparison:
   * The performance of the Q-learning algorithm will be compared to the Stratus Algorithm using the same job trace data.
   * Key performance metrics such as cost optimisation, resource utilisation, and job completion time will be used to assess how well the Q-learning algorithm performs in a simulated cloud environment.
   * The results will provide insights into whether the Q-learning approach offers any advantages over Stratus, particularly in terms of reducing costs while maintaining efficiency.

# Evaluation of algorithms (to be done as a team in next part of FYP)

The evaluation of the various job scheduling algorithms will be a collaborative effort, where each team member will be responsible for simulating a specific algorithm (done in the previous part). By analysing the performance of each algorithm under similar conditions, we will be able to compare their strengths and weaknesses in a comprehensive and systematic manner.

1. Individual Simulations: Each team member will run simulations of their chosen scheduling algorithm, using the same real-world job trace data from Google. These simulations will be conducted in a controlled environment to ensure consistency across the evaluations. The key performance metrics, including cost-effectiveness, resource utilisation and fairness, will be recorded for each algorithm.
2. Performance Comparison: Once the individual simulations are complete, the team will come together to compare the results. We will create comparative charts and tables to analyse which algorithms perform better under specific conditions. For example, some algorithms may excel at minimising costs, while others may be more effective at reducing job completion time or maximising resource efficiency.
3. Identifying Gaps and Potential Improvements: After evaluating each algorithm’s performance, we will collectively identify any gaps or inefficiencies in their scheduling strategies. For instance, one algorithm may struggle with resource utilisation under certain workloads, while another may have high costs due to inefficient job prioritisation.
4. Exploring New/Hybrid Algorithms: Based on the insights gained from the evaluation, the team will brainstorm potential new approaches or hybrid algorithms that combine the strengths of the different scheduling methods. For example, we may explore a hybrid model that leverages the cost-saving features of the Stratus Algorithm while incorporating the adaptive learning capabilities of machine learning algorithms. By merging the best aspects of the evaluated algorithms, we hope to design a more efficient and cost-effective scheduling solution for cloud environments.
5. Final Analysis and Documentation: Once we have explored the performance of all algorithms and potentially developed new or hybrid approaches, the final step will be to document the comparative analysis. This will include detailed results, charts, and explanations for why certain algorithms performed better in specific scenarios. The team will also outline any proposed new algorithms and highlight their potential benefits over existing methods.

# Proposed Project Timeline

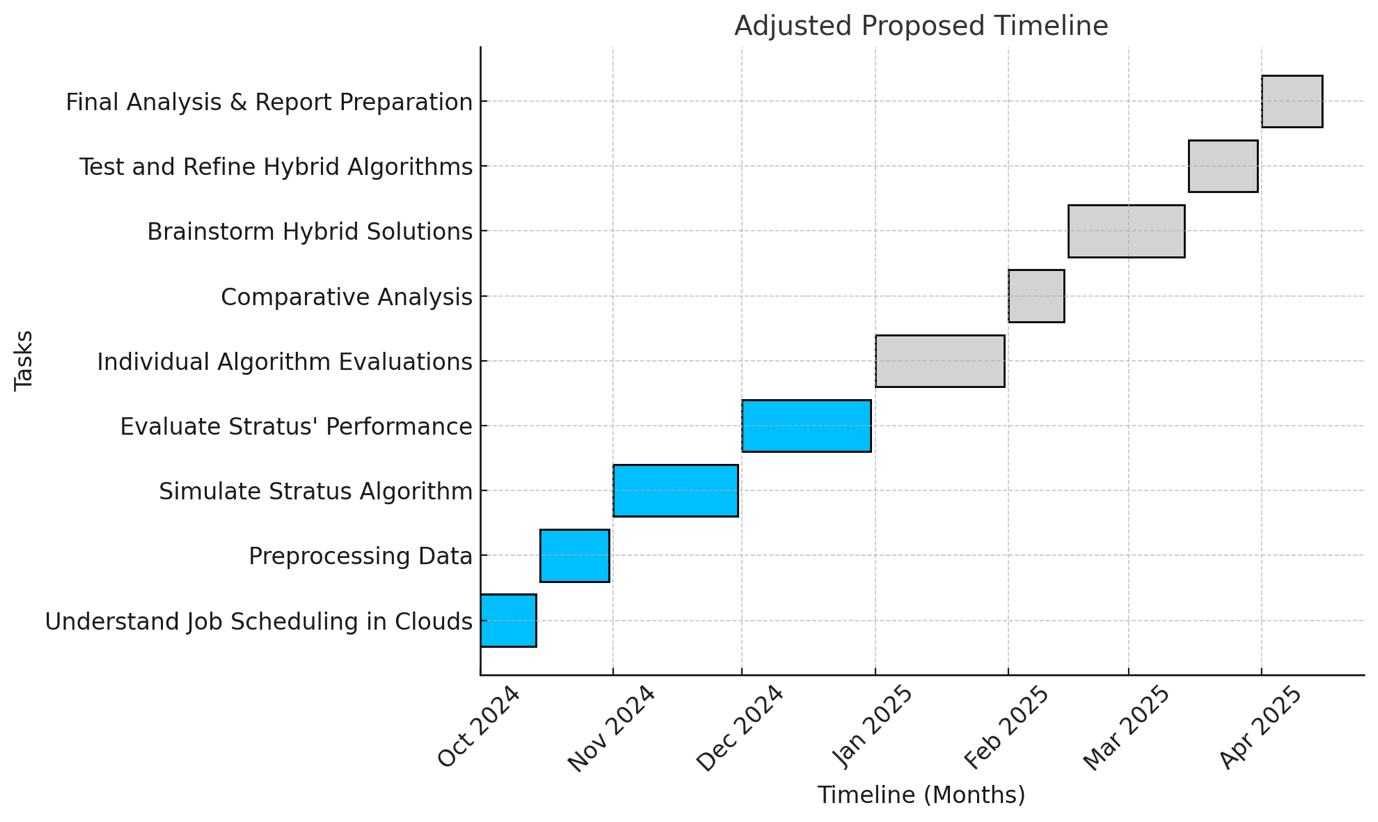


Figure 15: Gantt Chart of FYP Project

# References

1. Gartner. (2023, November 13). Gartner forecasts worldwide public cloud end-user spending to reach $679 billion in 2024. Gartner Newsroom. <https://www.gartner.com/en/newsroom/press-releases/11-13-2023-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-679-billion-in-20240>

2. Cyfuture Cloud. (2024, November 19). Companies waste up to 30% of their cloud spending due to inefficiencies. Cyfuture Cloud Blog. <https://cyfuture.cloud/blog/companies-waste-up-to-30-of-their-cloud-spending-due-to-inefficiencies/>

3. Chung, A., Park, J. W., & Ganger, G. R. (2018, October 11–13). *Stratus: Cost-aware container scheduling in the public cloud*. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '18)* (pp. 121–134). Carnegie Mellon University.

4. Tong,s., Chen, H., Deng, X., Li, K., & Li, K. (2020, February). A scheduling scheme in the cloud computing environment using deep Q-learning. Information Sciences, 512, 1170–1183. https://doi.org/10.1016/j.ins.2019.09.072