NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SCSE21-0868

Distributed Dispatcher & Scheduler with Bandwidth Constraint in Edge Computing

Submitted by: Nicholas Low Zhi Wei

Supervisor: A/P Arvind Easwaran

Examiner: Prof Thambipillai Srikanthan

School of Computer Science and Engineering

A final year project report resented to the Nanyang Technological University

in partial fulfilment of the requirements of the degree of

Bachelor of Engineering

2022

Abstract

In this report, we study the distributed dispatcher & scheduler with bandwidth constraint in edge computing. We will first image and create a cloud computing environment for a centralised solution. We then consider networking bandwidth constraints and a server's computational resources before creating a centralised algorithm for dispatching and scheduling. Subsequently, we implement this algorithm in a simulation environment and confirm our understanding of the cloud computing concept and test the algorithm's correctness. After which, the cloud computing environment will be viewed from a decentralised perspective where a decentralised solution can be formulated. A final decentralised solution will be created by modifying the above centralised solution to abide by the three key characteristics of a decentralised system, concurrency, loosely coupled and independent of failure.

Acknowledgements

I would like to extend my greatest gratitude to my Final Year Project supervisor, A/P Arvind Easwaran as well as fellow PHD student Gao Chuanchao for their meticulous guidance and support throughout the past year of my Final Year Project. They both have been very approachable and knowledgeable about this report's subject matter, Cloud/Edge Computing, and provided me with the required materials for my study.

I would also like to thank Nanyang Technological University for the opportunity to take part in this Final Year Project. It has been an exceptional journey over the past year and has presented me with many invaluable experiences.

Contents

List of Figures

List of Tables

Definitions/Formulas

Undirected network graph	$G = (V, E)$
Access point	V
Edge	$E(v_i, v_j)$
Communication link	$e_{i,j}$
Propagation latency	$l_{i,j}$
Bandwidth	$b_{i,j}$
Total number of edge servers	K
Edge server	k
Edge server resources	S_k
Application	m
Application resource requirement	s_m
Task	R
Task arrival time	a_R
Task data to process	y_R
Task deadline	d_R
Task computing time on edge server	$p_{R,k}$
Task computing time on cloud server	$p_{R,cloud}$
Propagation latency of cloud server	L
Upload latency of task and server	$\delta\uparrow_R = \sum_{e_{i,j} \in p_{R,k}} l_{i,j} + \frac{y_R}{min_{e_{i,j} \in p_{R,k}} b_{i,j}}$
Download latency of task and server	$\delta\downarrow_R = \sum_{e_{i,j} \in p_{R,k}} l_{i,j}$
Wait latency	$w_R = w_{i,R} + w_{k,R}$
Wait latency for uploading	$w_{i,R}$
Wait latency for computing	$w_{k,R}$
Time to complete task	$T = \delta\uparrow_R + \delta\downarrow_R + w_R + p_{R,k}$
Completion time of task on edge server	$C_R = a_R + \delta\uparrow_R + \delta\downarrow_R + w_R + p_{R,k}$
Completion time of task on cloud server	$C_R = a_R + 2 \times L + \frac{y_R}{B} + w_{cloud,R} + p_{R,k}$
Scheduling sequence	Q
Percentage of tasks completed	N

Average completion time	ACT
List of master orchestrators	O
Master orchestrator scheduling table	h
List of scheduling tables	H
Task score	(N, ACT)

Table 1: Definitions and formulas

1. Introduction

1.1. Background

The Fourth Industrial Revolution has brought to reality the concept of smart homes as smart technology is introduced into everyday devices. Cloud computing is a technological concept where servers which are situated far away from the end users provide the necessary computational resources to run various Internet of Things (IoT) scenarios. With the explosive growth of IoT devices, cloud servers face the problems of allocating computational resources to meet the sheer number of requests from the large volume of end users and of achieving low latency for time sensitive requests.

Edge computing is an extension of cloud computing which performs computation and data storage at locations close to the end users. Edge servers will process requests from the end users and only send relevant data back to the cloud. This helps reduce the amount of data sent to and processed at the central cloud server thus reducing the bandwidth required of cloud services. It also achieves low latency for time sensitive requests as most requests are processed on edge servers close to the end users while heavier tasks can enjoy a wider bandwidth back at the cloud server. This allows the heavier tasks to be processed quicker and meet the low latency requirements from the end users.

Creating an efficient scheduling and allocation algorithm for edge computing is crucial as it directly affects the efficiency of both the edge's and cloud's resources. These will impact the end users' experience when using their IoT devices. Although there have been approaches to formulate scheduling and allocation algorithms for edge computing over the years, they are not without their shortcomings [1-6].

[1] uses the ant colony optimisation heuristic to generate a scheduling algorithm which gives the edge server provider the maximum profit, total revenue subtracted by execution cost. [2] address the point where these algorithms do not consider the available networking bandwidth and thus introduced a greedy scheduling algorithm which reduced missed task deadlines by up to 60%. However, it in turn loses the maximising of profit for edge server providers. [3] presents an algorithm which allows for a best fit of resource allocation without the use of a main orchestrator but does not solve the problem of task scheduling. Individually, these algorithms do not solve both the scheduling and allocation problems of edge computing.

1.2. Project Objective

This report aims to find a solution by decentralising a centralised edge computing solution. This combines the advantages of the reviewed research and addresses their respective shortcomings by playing off each of their strengths against their weaknesses. The proposed algorithm will focus on using an algorithm that allows schedulers to greedily schedule task requests independently and communicate with other schedulers to avoid overlaps. This allows them to share their request load as well as available resources, resulting in the optimal allocation of the computational resources. The greedy algorithm will then effectively schedule tasks and prevent clogging of the network's bandwidth. These key points will enable the edge server to cater to numerous end users while still achieving low latency for time sensitive requests.

2. Literature Review

There have been many studies in the cloud/edge computing network field. These studies span across a wide range of factors, covering many considerations when formulating an edge computing network solution.

2.1. DEDAS

DEDAS is an online task dispatching and scheduling algorithm with bandwidth constraint. Their key feature is their focus on balancing computing resources as well as networking bandwidth to achieve the maximum number of user task deadlines met. Their experiments and simulations proved that the deadline miss ratio has an up to 60% reduction when compared to state-of-the-art methods. Their algorithm is a logically centralised system, which although straightforward to implement, has certain vulnerabilities such as being vulnerable to failure and not being horizontally scalable. [2]

2.2. DRAGON

DRAGON is a distributed resource assignment and orchestration algorithm. It aims to achieve optimal distribution of shared resources between different applications in their edge infrastructure. They raise concerns about application optimisation caused by their heterogeneous requirements and states that the dynamic nature of edge computing networks render centralised orchestrators unreliable. In the DRAGON algorithm, they use a scoring and voting system to facilitate agreement between their decentralised orchestrators. [3]

2.3. ONDISC

OnDisc is an online job dispatching and scheduling algorithm in edge network. This paper aims to minimise the duration between a device's task's release time and the arrival of its computational result. Each task must be offloaded to servers with upload and download delays. They assign weights to the tasks based on their latency sensitivity. This will then be minimised to achieve minimum weighted latency. Simulations with data-trace from Google produced resulted in a big reduction in weighted times compared to heuristic algorithms. [4]

3. Centralised System Overview

3.1. System Model Overview

3.1.1. Network Model

The edge computing network will be modelled as an undirected graph, $G = (V, E)$. V represents both intermediary access points as well as access points that have been assigned as an edge server. An edge $E(v_i, v_j)$ exists only if it has a communication link $e_{i,j}$ which has a propagation latency $l_{i,j}$ and a total bandwidth $b_{i,j}$. The shortest path from any access point to another access point is already calculated and known.

3.1.2. Edge Server Model

Edge servers are deployed onto access points and each access point can only host a maximum of one edge server. K represents the total number of edge servers. Each edge server k will have resources, S_k configured for specific applications and each application m has a minimum resource requirement of s_m. Thus, each edge server can only service limited number of applications where $\sum s_m \leq S_k$. The edge server will only be able to service tasks if it has the corresponding application configured. Each edge server is set up differently and their computing power is also assumed to be known. The edge servers will process incoming tasks sequentially. Tasks which have already been assigned an edge server will not be allowed to be rescheduled onto other edge servers.

3.1.3. Task Model

Tasks will arrive from users in arbitrary order and time. Each task R contains data to be processed, y_R, and a deadline d_R. The task will arrive at its nearest access point at time a_R before being dispatched to either an edge server or the cloud server for computing. Each task requests for specific applications thus it should be dispatched to edge servers that have been configured with the corresponding applications. Each edge server is configured differently thus the computing time of the task will be different on each server. A task's computing time on an edge server will be denoted by $p_{R,k}$. When a task has been assigned a target server, it will take the shortest path which has been previously predefined and stored in each access point.

3.1.4. Cloud Model

The cloud server is considered the most powerful server which access points can offload tasks for computing. The cloud server has enough resources to configure all kinds of tasks. Due to the vast number of resources and powerful computing power, we can consider tasks that have arrived at the cloud server to have no waiting time and negligible computing time $p_{R,cloud}$. However, there is a propagation latency L for communication between an access point and the cloud server. This is much larger than the internal latency of the network. The available bandwidth between access points and the cloud will be B.

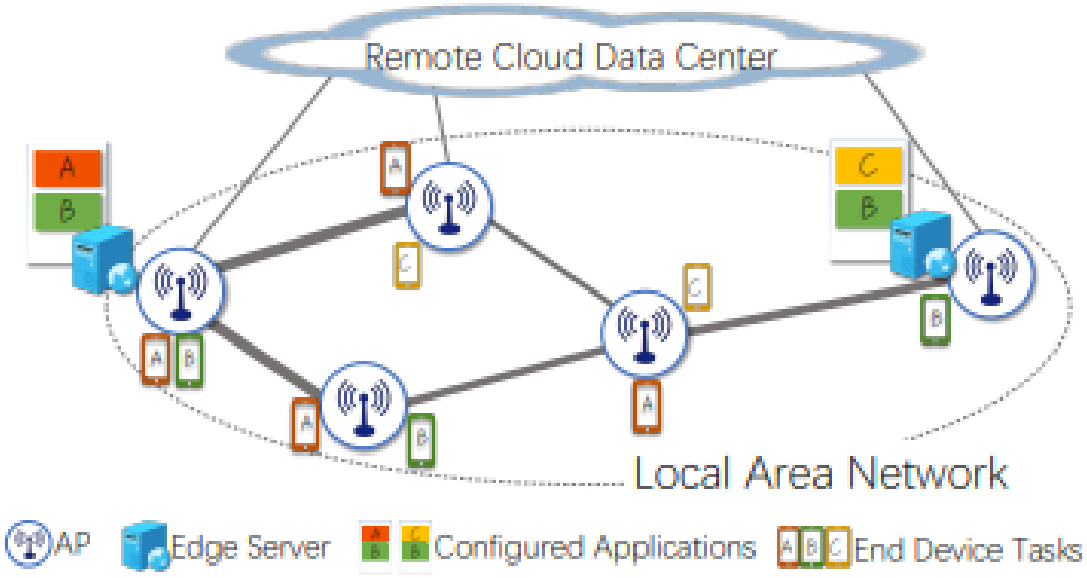

Figure 1: Cloud Computing Network Environment [2]

3.1.5. Other Considerations

Each task-target server pair will have an upload latency which consists of the summation of propagation latencies and communication latencies. $\delta\uparrow_R = \sum_{e_{i,j} \in p_{R,k}} l_{i,j} + \frac{y_R}{min_{e_{i,j} \in p_{R,k}} b_{i,j}}$

We assume that the size of the data after processing is small thus for download latency, we only include propagation latencies. $\delta\downarrow_R = \sum_{e_{i,j} \in p_{R,k}} l_{i,j}$

Each task may be required to wait before its upload and processing begins. Thus, we formulate this as $w_R = w_{i,R} + w_{k,R}$ where $w_{i,R}$ represents the wait latency before uploading and $w_{k,R}$represents the wait latency before computing.

The total time from a task's upload to the user receiving the completed task is $T = \delta\uparrow_R + \delta\downarrow_R + w_R + p_{R,k}$. If the task is sent to an edge server, the completion time will be $C_R = a_R + \delta\uparrow_R + \delta\downarrow_R + w_R + p_{R,k}$. If the task is sent to the cloud server, the completion time will be $C_R = a_R + 2 \times L + \frac{y_R}{B} + w_{cloud,R} + p_{R,k}$.

The task meets its deadline if $C_R \leq d_R$ otherwise it is declared as a missed deadline. This centralised edge computing solution aims to attain maximum number of completed tasks within deadlines and minimum average task completion time (ACT).

3.2. Centralised Algorithm

3.2.1. Scheduling Policy

Algorithm 1: Scheduling Policy

Input: Original scheduling sequence $Q_{sequence}$ and new dispatched task R

Output: New scheduling sequence Q_{best}, percentage of tasks completed N_{best}, average completion time ACT_{best}

1 $N_{best} \leftarrow 0$

2 $ACT_{best} \leftarrow \infty$

3 $Q_{best} \leftarrow \emptyset$

4 $Q'_{sequence} \leftarrow$ Add R to the end of $Q_{sequence}$

5 **IF** $Q'_{sequence}$ is feasible **THEN**

6 $\quad Q_{best} \leftarrow Q'_{sequence}$

7 $\quad ACT_{best} \leftarrow ACT(Q'_{sequence})$

8 $\quad N_{best} \leftarrow 100\%$

9 **FOR** every task R' in $Q_{sequence}$ **DO**

10 $\quad Q'_{sequence} \leftarrow$ Add R before R in $Q_{sequence}$

11 $\quad$ **IF** $Q'_{sequence}$ is feasible **AND** $ACT(Q'_{sequence}) < ACT_{best}$ **THEN**

12 $\quad\quad Q_{best} \leftarrow Q'_{sequence}$

13 $\quad\quad ACT_{best} \leftarrow ACT(Q'_{sequence})$

14 $\quad\quad N_{best} \leftarrow 100\%$

15 **IF** $Q_{best} = \emptyset, (\text{none of } Q'_{sequence} \text{ is feasible})$ **THEN**

16 $\quad Q_{best}, N_{best}, ACT_{best} \leftarrow$ Choose task to replace (Algorithm 2)

Algorithm 2: Choose task to replace

Input: Original scheduling sequence $Q_{sequence}$ and new dispatched task R

Output: Scheduling sequence after task has been replaced Q_{best}, percentage of tasks completed N_{best}, average completion time ACT_{best}

1 $N_{best} \leftarrow {size(Q_{sequence})}/{size(Q_{sequence}) + 1}$

2 $ACT_{best} \leftarrow ACT(Q_{sequence})$

3 $Q_{best} \leftarrow Q_{sequence}$

4 **FOR** every task R' in $Q_{sequence}$ **DO**

5 **IF** R and R' share transmission path or share same server k **THEN**

6 **IF** R' has started processing **THEN**

7 **CONTINUE**

8 $Q'_{sequence} \leftarrow$ Replace R' with R in $Q_{sequence}$

9 **IF** $Q'_{sequence}$ is feasible **AND** $ACT(Q'_{sequence}) < ACT_{best}$ **THEN**

10 $Q_{best} \leftarrow Q'_{sequence}$

11 $ACT_{best} \leftarrow ACT(Q'_{sequence})$

3.2.2. Dispatching Policy

Algorithm 3: Dispatching Policy

Input: Task R, set of target servers including cloud server K'

Output: Target server k_{target} to dispatch task R

```
k_target ← cloud
N_best ← 0
ACT_best ← ∞
FOR each server k in K' DO
    IF k = cloud THEN
        p_R,cloud ← 0
    N, ACT, Q ← try to dispatch R to k (Algorithm 1)
    IF N > N_best OR (N = N_best AND ACT < ACT_best) THEN
        IF task R meets deadline THEN
            k_target ← k
        ELSE
            k_target ← cloud
        N_best ← N
        ACT_best ← ACT
```

3.3. Java Implementation

The centralised algorithm will be built in the Eclipse IDE version 4.22.0 and coded, compiled and run in Java SE Development Kit version 17.0.1+12.

Java is the programming language of choice for this project due to its ease of use. It was designed to be easy to write, compile and debug. It is also object-oriented which encourages modular programs and improves code reusability. Finally, Java is platform-independent which allows its programs to be run on almost any system. [7]

Eclipse IDE is one of the leading development platforms which provides a simplified user interface to allow easy coding when developing scalable and high functionality applications with open-source capabilities. [8]

To abide by good object-oriented practices, we will implement several classes to house different modules of the edge network system. A class diagram is designed to document the classes, its functions, and their relation to one another.

3.3.1. Class Diagram

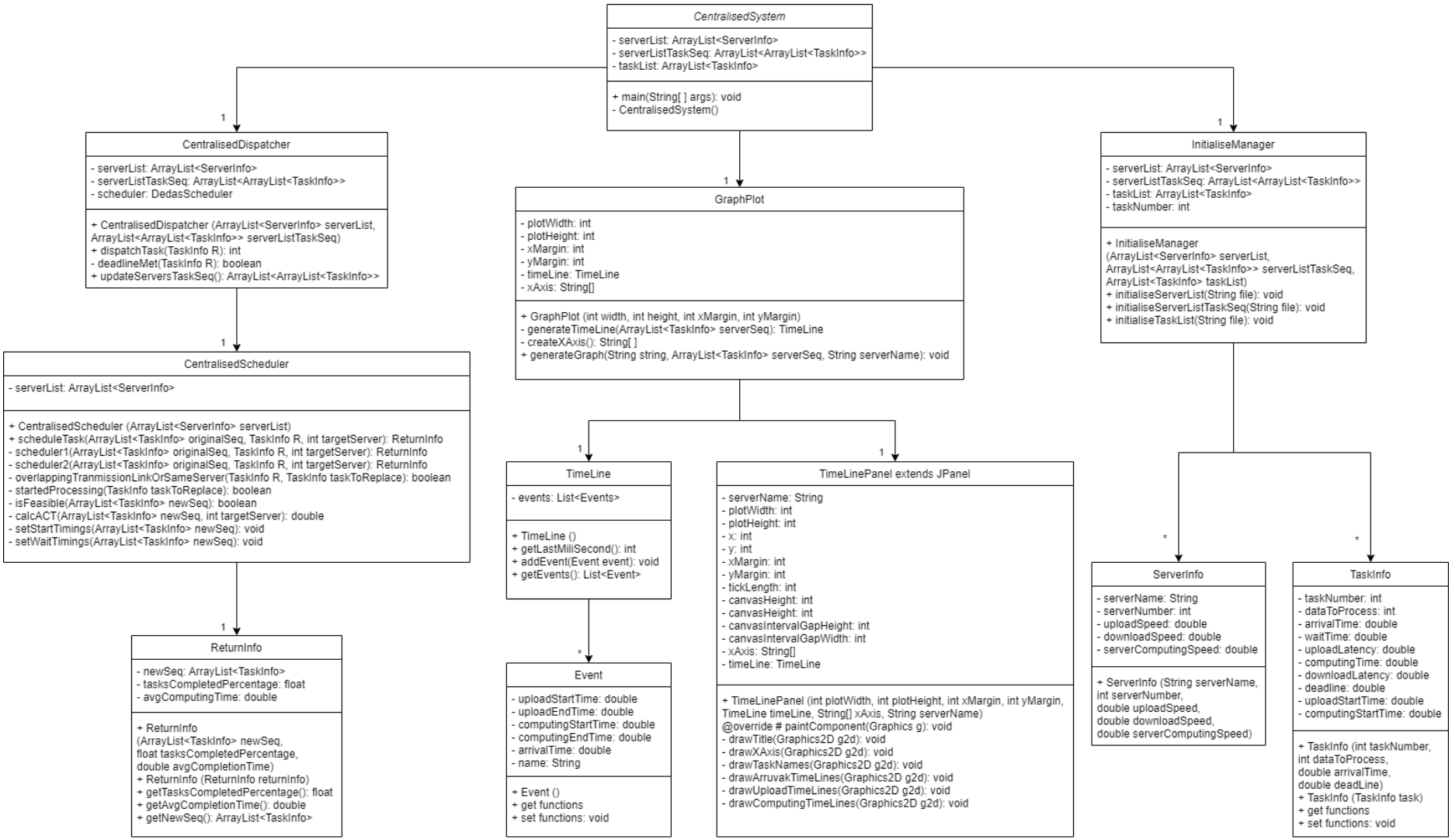

Figure 2: Class Diagram of Centralised System

3.3.2. Code Breakdown

The main program runs in the CentralisedSystem class file. This class controls three other classes which works together to run the edge network simulation. These three classes are the CentralisedDispatcher, GraphPlot and InitialiseManager.

The InitialiseManager class is responsible for reading configuration files to setup the cloud and edge servers, as well as their respective current task schedules. It also handles the setup of the input task stream which will simulate tasks arriving from users at an arbitrary order and time. It uses the "javax.xml.parsers" and the "org.w3c.dom" libraries to extract data from predefined xml files and organise and prepare them into ServerInfo and TaskInfo class modules for the main CentralisedSystem class to read.

The CentralisedDispatcher class handles the Algorithm 3 to dispatch incoming tasks. It contains another class CentralisedScheduler which runs Algorithms 1 and 2 to schedule given tasks and their target servers. The CentralisedScheduler has a ReturnInfo module which will contain the required information such as the percentage of completed tasks and the average completion time. This module is to be returned to the CentralisedDispatcher for comparison to determine the best target server for any given task.

The GraphPlot class main role is to draw the schedules of each server to allow us to visualise task sequence of the servers at any point of the simulation. It extends on the JPanel class from the Java Swing package to help draw the server's schedule. It also houses the TimeLine and Event classes which contain information of each task which will be used when drawing the server's schedule.

In our simulation, the CentralisedSystem class runs three InitialiseManager functions, initialiseServerList, initialiseServerTaskSeq and initialiseTaskList to setup the servers and the incoming task list. The GraphPlot is then called to plot out the original servers' sequences. This task list is put into a while loop which pops the head of the list and puts it into the dispatching algorithm. This is repeated until all the tasks in the task list has been popped off. After which, GraphPlot is called again to plot out the final servers' sequences. At this point, we can analyse and evaluate the performance of the edge network system.

3.3.3. Evaluation

In this simulation, we populate the “serversList.xml” configuration file with 3 servers, 1 cloud server and 2 edge servers. The edge servers’ settings have been configured with fixed upload and computing speeds while its download latency is fixed to a very small delay. The cloud server’s upload and download latency is fixed to a larger latency. Refer to Appendix A for detail setup information.

The “serversTaskSeq.xml” holds information pertaining each server’s current scheduled tasks. They have been configured where cloud server is preloaded with 3 tasks in its current schedule and the edge servers are preloaded with 2 tasks each. Refer to Appendix B for detail setup information.

The “taskList.xml” file is setup with 6 tasks which have been configured with random settings. Refer to Appendix C for detail setup information.

The simulation yields the following results.

```
-----Centralised System-----
InitManager: Initialising Servers...
InitManager: Cloud, Edge 1, Edge 2, servers added.
InitManager: Initialising Servers Current Schedule...
InitManager: All servers task list initialsied.
InitManager: Initialising Tasks to be added...
InitManager: 6 tasks added.
```

Figure 3: Centralised System initialising log

We can see that for the first task, in this case, the task number starts from task 8 as there are already 7 tasks in all the servers, the dispatcher calls the scheduler to schedule the task on each server. The scheduler then provides a score of task completion rate and the average completion time. After returning the score, the dispatcher will check if the deadline is met before entering the comparing phase. In the event where the deadline is not met, that server will not be considered when selecting the target server of the task. N is first compared and the one with the highest percentage is selected. In events of a tie, the server with the lowest ACT will be chosen, in this case, it was server 1.

```
Centralised System: Sending task 8 to dispatcher...
Scheduler: Attempting to schedule task on server 0
Scheduler: No tasks replaced! Score: N=100%, ACT=1.05
Scheduler: Attempting to schedule task on server 1
Scheduler: No tasks replaced! Score: N=100%, ACT=0.22
Dispatcher: Checking if task 8 met deadline on server 1...
Dispatcher: Task 8 deadline met on server 1 (N=100%, ACT=0.22 ms).
Scheduler: Attempting to schedule task on server 2
Scheduler: No tasks replaced! Score: N=100%, ACT=0.38
Dispatcher: Best server = 1, Score: N=100%, ACT=0.22 ms
Centralised System: Task 8 has been dispatched to edge server 1.
```

Figure 4: Task 8 dispatching/scheduling log

Similarly for task 9, 10 and 11, their performance on each server is evaluated and server 1 is chosen again as the target server due to the low ACT.

```
Centralised System: Sending task 9 to dispatcher...
Scheduler: Attempting to schedule task on server 0
Scheduler: No tasks replaced! Score: N=100%, ACT=1.05
Scheduler: Attempting to schedule task on server 1
Scheduler: No tasks replaced! Score: N=100%, ACT=0.26
Dispatcher: Checking if task 9 met deadline on server 1...
Dispatcher: Task 9 deadline met on server 1 (N=100%, ACT=0.26 ms).
Scheduler: Attempting to schedule task on server 2
Scheduler: No tasks replaced! Score: N=100%, ACT=0.44
Dispatcher: Best server = 1, Score: N=100%, ACT=0.26 ms
Centralised System: Task 9 has been dispatched to edge server 1.
```

Figure 5: Task 9 dispatching/scheduling log

```
Centralised System: Sending task 10 to dispatcher...
Scheduler: Attempting to schedule task on server 0
Scheduler: No tasks replaced! Score: N=100%, ACT=1.05
Scheduler: Attempting to schedule task on server 1
Scheduler: No tasks replaced! Score: N=100%, ACT=0.34
Dispatcher: Checking if task 10 met deadline on server 1...
Dispatcher: Task 10 deadline met on server 1 (N=100%, ACT=0.34 ms).
Scheduler: Attempting to schedule task on server 2
Scheduler: No tasks replaced! Score: N=100%, ACT=0.49
Dispatcher: Best server = 1, Score: N=100%, ACT=0.34 ms
Centralised System: Task 10 has been dispatched to edge server 1.
```

Figure 6: Task 10 dispatching/scheduling log

```
Centralised System: Sending task 11 to dispatcher...
Scheduler: Attempting to schedule task on server 0
Scheduler: No tasks replaced! Score: N=100%, ACT=1.05
Scheduler: Attempting to schedule task on server 1
Scheduler: No tasks replaced! Score: N=100%, ACT=0.41
Dispatcher: Checking if task 11 met deadline on server 1...
Dispatcher: Task 11 deadline met on server 1 (N=100%, ACT=0.41 ms).
Scheduler: Attempting to schedule task on server 2
Scheduler: No tasks replaced! Score: N=100%, ACT=0.43
Dispatcher: Best server = 1, Score: N=100%, ACT=0.41 ms
Centralised System: Task 11 has been dispatched to edge server 1.
```

Figure 7: Task 11 dispatching/scheduling log

For task 12, its performance on server 2 is better thus it is set as the target server.

```
Centralised System: Sending task 12 to dispatcher...
Scheduler: Attempting to schedule task on server 0
Scheduler: No tasks replaced! Score: N=100%, ACT=1.05
Scheduler: Attempting to schedule task on server 1
Scheduler: No tasks replaced! Score: N=100%, ACT=0.43
Dispatcher: Checking if task 12 met deadline on server 1...
Dispatcher: Task 12 deadline met on server 1 (N=100%, ACT=0.43 ms).
Scheduler: Attempting to schedule task on server 2
Scheduler: No tasks replaced! Score: N=100%, ACT=0.41
Dispatcher: Checking if task 12 met deadline on server 2...
Dispatcher: Task 12 deadline met on server 2 (N=100%, ACT=0.41 ms).
Dispatcher: Best server = 2, Score: N=100%, ACT=0.41 ms
Centralised System: Task 12 has been dispatched to edge server 2.
```

Figure 8: Task 12 dispatching/scheduling log

For task 13, it performs the best on server 1 again thus it is set as the target server.

```
Centralised System: Sending task 13 to dispatcher...
Scheduler: Attempting to schedule task on server 0
Scheduler: No tasks replaced! Score: N=100%, ACT=1.05
Scheduler: Attempting to schedule task on server 1
Scheduler: No tasks replaced! Score: N=100%, ACT=0.44
Dispatcher: Checking if task 13 met deadline on server 1...
Dispatcher: Task 13 deadline met on server 1 (N=100%, ACT=0.44 ms).
Scheduler: Attempting to schedule task on server 2
Scheduler: No tasks replaced! Score: N=100%, ACT=0.45
Dispatcher: Best server = 1, Score: N=100%, ACT=0.44 ms
Centralised System: Task 13 has been dispatched to edge server 1.
```

Figure 9: Task 13 dispatching/scheduling log

Log to state that simulation has ended.

```
Centralised System: All tasks completed, shutting down...
-----END-----
```

Figure 10: End of simulation log

Plotting the before and after server schedules, we can see that the tasks have been rearranged to ensure maximum resource utilisation and thus achieving the lowest ACT result. Both the cloud and edge 2 servers saw no change in their schedules as all the new tasks were dispatched to edge 1 server. In the final schedule for edge 1 server, we can see that task 10 is pushed back due to the busy computing pipeline. However, the algorithm has adjusted for this and determined this sequence to be the optimal sequence for lowering the server's average task completion time.

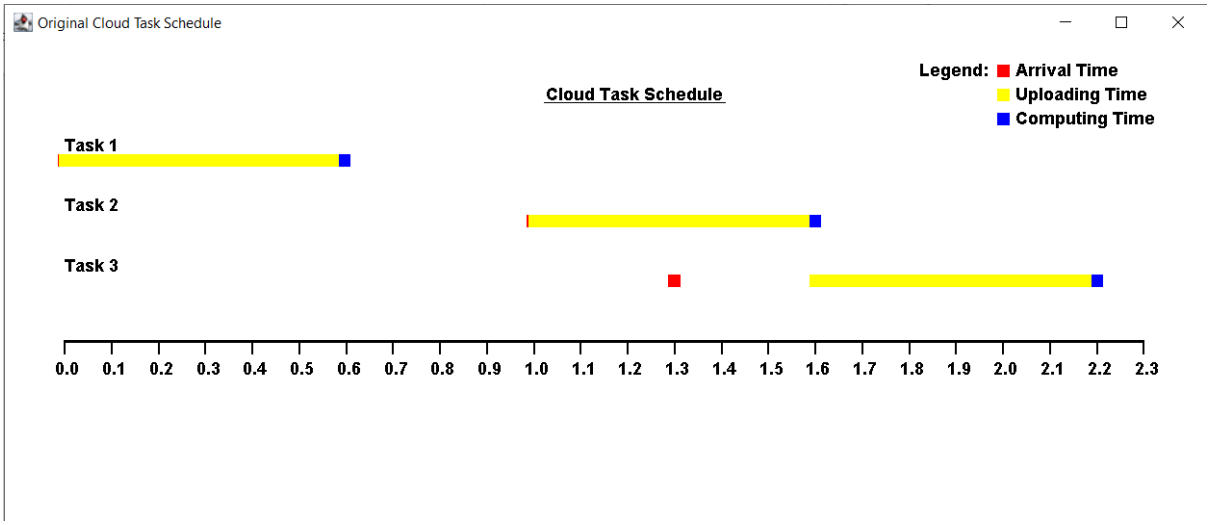

Figure 11: Original cloud server task sequence graph

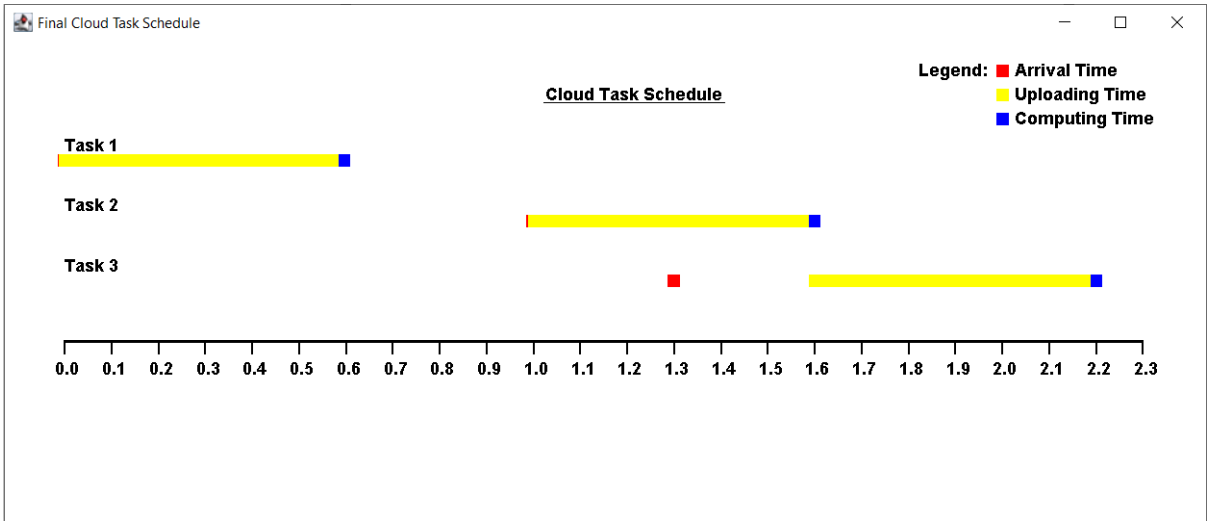

Figure 12: Final cloud server task sequence graph

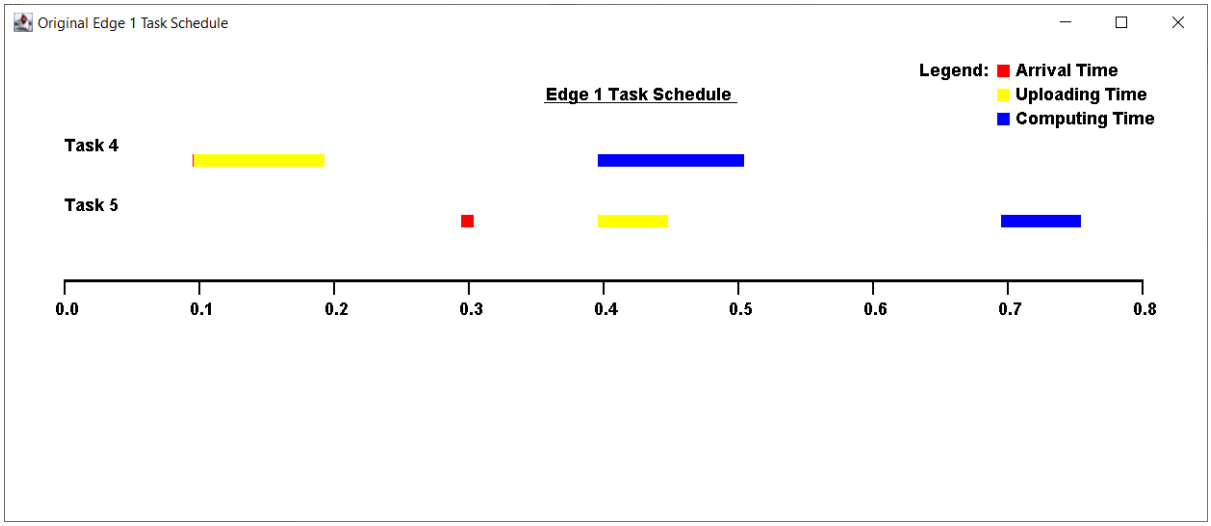

Figure 13: Original edge 1 server task sequence graph

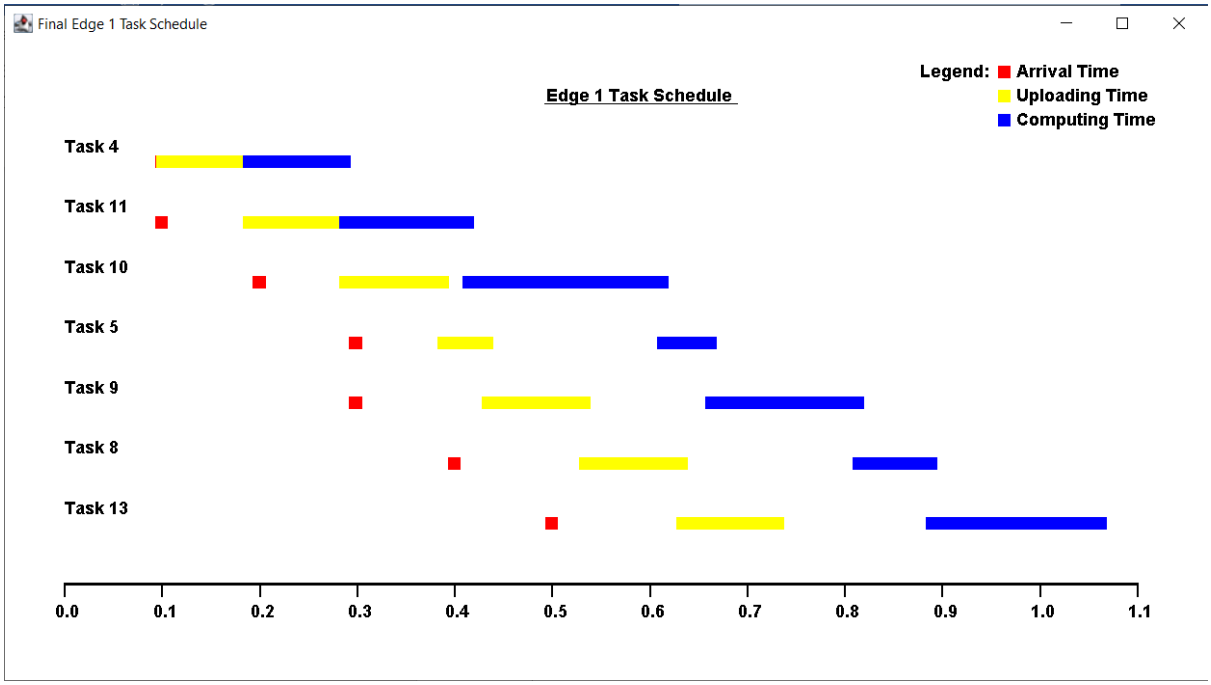

Figure 14: Final edge 1 server task sequence graph

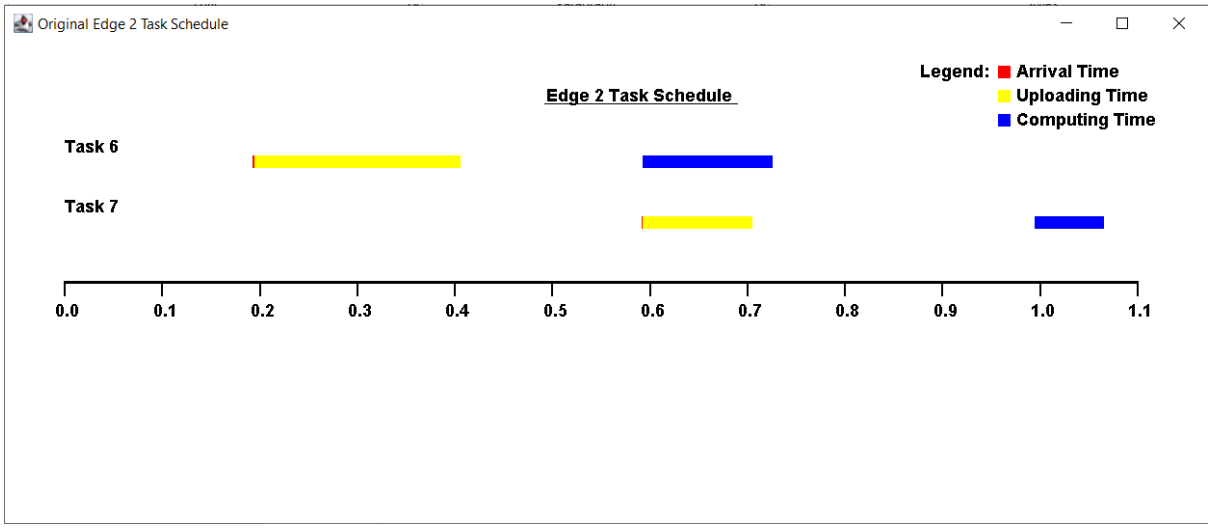

Figure 15: Original edge 2 server task sequence graph

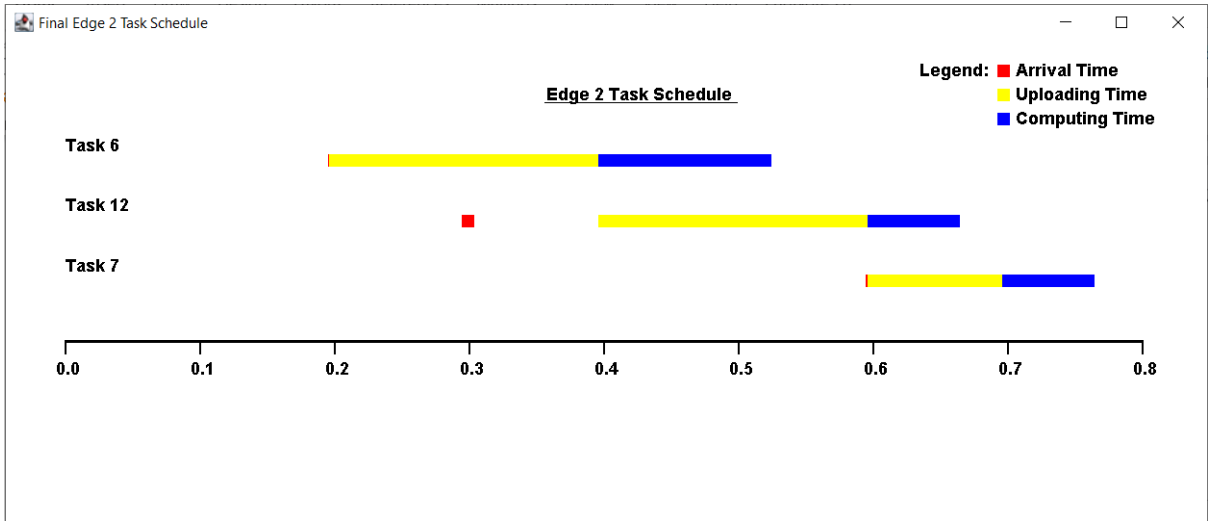

Figure 16: Final edge 2 server task sequence graph

3.3.4. Constraints

There are a few features of the centralised system that has been left out of this implementation. The upload latency ignores the bandwidth of other access points and only considers the edge's bandwidth. The download latency has been simplified to a fixed predetermined value. This was done to reduce the implementation complexity due to project time constraints. The three functions "overlappingTransmissionLinkOrSameServer", "startedProcessing" and "isFeasible" have been defaulted to return "true", "false" and "true" respectively. These functions have also been simplified to adjust for the project's schedule. The simulation results are also limited to manual inputs from the user, thus the limiting simulation to a much smaller scale as compared to a real life commercial sized environment.

4. Decentralised System Model

4.1. Problem Formulation

In edge computing, a centralised system is a set of networked computers where their actions are orchestrated by a single orchestrator. This orchestrator controls the actions of each computer, the flow of data and usage of resources by sending dedicated instructions. They usually have a bottleneck where the system can only work as fast as the orchestrator's capabilities of sending out instructions to all the networked computers. The system is also vulnerable to failure where the orchestrator may fail and thus cause the whole system to collapse.

A decentralised system is a set of networked computers that coordinate their actions by communicating with one another through the passing of messages. A decentralised system has 3 fundamental characteristics, it must support concurrency, it must be loosely coupled, and it must be independent of failures [9, 10]. With its ability to support concurrent program execution, the system will usually have higher capacity which leads to a possibly higher performance. Being loosely coupled refers to the independence of a global clock and the lack of a global memory. The last characteristic, independent of failures, allows the system to be more robust and can also lead to other benefits such as modular upgrades, where only a section of the system need to shut down for maintenance or upgrades while the rest of the system can function as usual. Comparing this to a centralised system, its is evident that a decentralised system has the means to provide better performance and robustness to an edge network.

An edge computing centralised solutions has no horizontal scalability due to the limitations of a single orchestrator and a limited vertical scalability because of price to performance of hardware components. In the case of the above centralised solution, it uses a logically centralised scheduler to orchestrate the dispatching and scheduling of requests around the edge network. To formulate a decentralised solution, we take the base system model of the centralised solution and modify it to create a decentralised situation. We then analyse the centralised solution and implement changes to allow it to satisfy the three important decentralised concepts.

The steps to take are as follows. Firstly, we take the setting of the centralised algorithm and transform it into a decentralised problem. We then determine how to modify the network setup to affirm to a decentralised system's requirements. After which, important details such as new information in the scheduler and dispatcher, and the required data that is to be broadcasted to other schedulers and dispatchers will be laid out. Finally, we investigate the different outcomes of the schedulers and dispatchers to formulate an agreement pattern for the schedulers to adhere to. This will ensure that all schedulers and dispatchers will converge to a result where there is no clash in the scheduling of requests.

4.2. System Model Design

Our centralised solution falls in a setting where end devices can offload tasks to the edge or cloud servers for computing. These are assumed to be achievable within one wireless hop. Each access point has a dispatching module that handles the selection of edge server, and a scheduling module that organises the sequence and allocates bandwidth for each task. These modules work together as an orchestrator and are logically centralised with the other access points orchestrators and from a broader perspective, can be visualised as a single centralised scheduler. The latency of running this orchestrator will be ignored as task uploading and computing time is much larger.

To model the decentralised solution, we treat each access point's orchestrator module as a standalone orchestrator called "Master Orchestrator". Each master orchestrator will cover a range of access points and will schedule and dispatch tasks to edge servers within its coverage range. An edge server may be within the coverage zone of multiple master orchestrators. Each master orchestrator may also be within the coverage zone of multiple master orchestrators and the latter will be considered the immediate master orchestrators of the former. A master orchestrator will run the dispatching and scheduling algorithm and communicate with the other master orchestrators to organise the outcome of their task's target server.

These newly created master orchestrators will contain a list of edge servers and a list of immediate master orchestrators within their communication range. The current schedule of the edge servers is expected to be always known and available. Each master orchestrator will also contain a table comprising of tasks to be dispatched and their target servers. This list of tasks and target servers will contain a score which consists of their task completion percentage and average completion time (N, ACT) and are to be broadcasted to other access points to undergo an agreement policy to ensure synchronisation of the network's actions.

Edge Server 1	Edge Server 2	…	Edge Server N
-	Task R , Score (N, ACT)	…	-

Table 2: Master Orchestrator's Task Schedule Table

The master orchestrators will also receive broadcasts from the other master orchestrators which are then put into the agreement policy to determine the best outcome for each edge server's schedule sequence. This is achieved by comparing the tasks' scores with the same target server and selecting the best fit result to be chosen as the winner of the edge server's resources. It is assumed that all master orchestrators would contain the same scheduling result after 3 rounds of broadcasts hence coming to an "agreement". The tasks that were not allocated resources will be rejected and their devices will be notified.

From this, we can see that the three fundamental characteristics of a decentralised system has been met. Because each master orchestrator does not take instructions from one another, they can run their own algorithms concurrently. This also results in the system being loosely coupled and only coordinate with each other through the passing of messages. Finally, the system is independent of failures as each master orchestrator does not need the others to function and can perfectly orchestrate tasks that they receive.

4.3. Decentralised Algorithm

4.3.1. Scheduling Policy

We use the same scheduling algorithm as the centralised algorithm above to schedule tasks and achieve the lowest average completion time. Both algorithm 1 and 2 will have the same responsibilities of scheduling a task and choosing a task to replace respectively.

4.3.2. Dispatching Policy

We use the same dispatching algorithm as the centralised algorithm above to dispatch tasks and evaluate which server has the best results. This server is then selected as the target server for the specific task. However, in this scenario, we will also return the corresponding task score (N, ACT) for the master orchestrator to attach to its task schedule table and to subsequently be broadcasted to other master orchestrators.

4.3.3. Agreement Policy

Algorithm 4: Agreement Policy

Input: Original personal table h, compared table h'

Output: New personal table h

```
1   FOR every k in h DO
2       FOR every k' in h' DO
3           IF k = k' THEN
4               IF N of k < N of k' THEN
5                   replace R, N, ACT of k with R, N, ACT of k'
                    (put compared task into personal table)
6               ELSE IF N of k = N of k' THEN
7                   IF ACT of k > ACT of k' THEN
8                       replace R, N, ACT of k with R, N, ACT of k'
                        (put compared task into personal table)
9                   ELSE IF ACT of k = ACT of k' THEN
10                      remove R, N, ACT from k
                        (reject both tasks)
```

4.3.4. Master Orchestrator

Algorithm 5: Master Orchestrator

Input: Task R, list of target servers K', list of master orchestrators O, scheduling table h

Output: Dispatch task R to target server k / Inform device that task R is rejected

```
k ← cloud
N ← 100%
ACT ← ∞
k, N, ACT ← try to dispatch R to K′ (Algorithm 3)
Insert (R, N, ACT) into h where k = k in h
WHILE(1)
    Broadcast h to O
    List of tables H ← Receive h′ from O
    agreed ← TRUE
    FOR h′ in H DO
        IF h ! = h′ THEN
            agreed ← FALSE
            h ← perform agreement policy (Algorithm 4)
    IF agreed THEN
        BREAK
IF h contains R THEN
    dispatch R to k
ELSE
    inform device that R is rejected
```

5. Conclusion, Challenges Faced, Recommendations/Future Work

5.1. Conclusion

In conclusion, this report has achieved its goal to decentralise a centralised edge computing solution. A centralised cloud environment was first theorised and established before formulating the centralised algorithm. This algorithm greedily schedules tasks and abides by the bandwidth constraints of network transfers to achieve maximum task completion rate while maintaining minimum average task completion time. This algorithm is then implemented in the Eclipse IDE where we can see the cloud computing network in simulation. The centralised cloud environment was then modified to satisfy decentralised concepts and new algorithm policies we conceived to enhance the centralised algorithm. This allowed the new decentralised algorithm to have concurrency, be loosely coupled and be independent of failures. Thus, it complies with the three key characteristics of a decentralised system.

5.2. Challenges Faced

There have been many challenges that I have faced and that were eventually overcome over the course of the Final Year Project.

At the very beginning of the project, I was required to read research papers and learn about the basics of cloud and edge computing. As this topic was foreign to me outside of common knowledge, it was hard to grasp onto key features to differentiate a good cloud computing solution from a bad cloud computing solution.

Another complication was the complexity of a centralised versus a decentralised cloud computing solution. Although it seems straightforward on paper, the actual design and concept of a decentralised solution is much more complicated. There were times where I found myself designing a decentralised cloud computing solution only for it to be a logically centralised cloud computing solution.

The implementation of the centralised solution in eclipse proved to have many challenges. One main challenge was the management of Java's handling of arrays and classes. Java is always pass-by-value [11]. However, when dealing with variables holding objects, the object's handle is being passed. Although the value passed is the object handle, we are essentially dealing with pass-by-reference. This was a large setback and I had to find workarounds to avoid this situation, especially since I was dealing with concatenated ArrayList variables.

As mentioned above, formulating a decentralised solution was much more complicated than expected. The decentralised solution characteristic, a system must be loosely coupled, was the biggest difficulty to overcome. Theorising concepts and algorithms to enable the different master orchestrators to generate the same task schedule table was complicated. I had to factor in the considerations when deciding the winning task and had to consider for master orchestrators that were out of range, which required many iterations of broadcast relaying to achieve standardising across all the task schedule tables of the many master orchestrators.

5.3. Recommendations/Future Work

With regards to future improvements, upload and download latencies can be properly formulated into the code. This will display a more accurate result in the simulation's calculations. The centralised system can also complete the implementation the three functions that have been previously returned default values. This will bring the centralised system closer to the intended theoretical work.

The simulation code can also be upgraded by implementing scripts which will generate a random scalable list of tasks into the configuration xml files. This will help scale the simulation and allow for realistic environment numbers. It can also implement Google cluster-usage traces which contains real world task data, and the simulation will be able to provide results that are closer to real world situations. [12]

Moving forward, the decentralised policies can be coded and implemented into the current code where tests can be applied to evaluate the feasibility and effectiveness of the new decentralised system compared to its centralised counterpart. The evaluation results can also be compared to other decentralised solutions to determine its efficacy before any actual real-world implementations.

Another potential project is to migrate the simulation system into CloudSim Plus, which is a state-of-the-art framework for cloud computing simulation [13]. The CloudSim Plus framework can be run as a maven project on eclipse and it provides a more accurate environment for cloud simulations. This could potentially improve the integrity and quality of the project which can also possibly open new doors for the proposed decentralised algorithm, be it being fine-tuned to improve performance, or being used as a cloud computing solution baseline by a major cooperation.

6. References

[1] J. Fan, X. Wei, T. Wang, T. Lan, and S. Subramaniam, "Deadline-Aware Task Scheduling in a Tiered IoT Infrastructure," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 4-8 Dec. 2017 2017, pp. 1-7, doi: 10.1109/GLOCOM.2017.8255037.

[2] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online Task Dispatching and Scheduling with Bandwidth Constraint in Edge Computing," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 29 April-2 May 2019 2019, pp. 2287-2295, doi: 10.1109/INFOCOM.2019.8737577.

[3] G. Castellano, F. Esposito, and F. Risso, "A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 29 April-2 May 2019 2019, pp. 2548-2556, doi: 10.1109/INFOCOM.2019.8737532.

[4] H. Tan, Z. Han, X. Y. Li, and F. C. M. Lau, "Online job dispatching and scheduling in edge-clouds," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 1-4 May 2017 2017, pp. 1-9, doi: 10.1109/INFOCOM.2017.8057116.

[5] R. Begam, H. Moradi, W. Wang, and D. Zhu, "Flexible VM Provisioning for Time-Sensitive Applications with Multiple Execution Options," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2-7 July 2018 2018, pp. 122-129, doi: 10.1109/CLOUD.2018.00023.

[6] T. T. Vu, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "QoS-Aware Fog Computing Resource Allocation Using Feasibility-Finding Benders Decomposition," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 9-13 Dec. 2019 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9013527.

[7] "Advantages of Java." https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java (accessed 12-October, 2022).

[8] "Eclipse Foundation." https://www.eclipse.org/ (accessed 12-October, 2022).

[9] G. Coulouris, J. Dollimore, and T. Kindberg, "Distributed Systems: Concepts and Design Edition 3," ed: Addison-Wesley.–2001.–779 p, 2001.

[10] M. Van Steen and A. Tanenbaum, "Distributed systems principles and paradigms," *Network*, vol. 2, p. 28, 2002.

[11] "Java Pass by Value." https://www.javatpoint.com/java-pass-by-value (accessed 12-October, 2022).

[12] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, vol. 1, 2011.

[13] "CloudSim Plus." https://cloudsimplus.org/ (accessed 12-October, 2022).

Appendix A: serversList.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<serversList>
    <server>
        <name>Cloud</name>
        <uploadSpeed>0.6</uploadSpeed>
        <downloadSpeed>0.6</downloadSpeed>
        <serverComputingSpeed>0</serverComputingSpeed>
    </server>
    <server>
        <name>Edge 1</name>
        <uploadSpeed>450</uploadSpeed>
        <downloadSpeed>0.1</downloadSpeed>
        <serverComputingSpeed>400</serverComputingSpeed>
    </server>
    <server>
        <name>Edge 2</name>
        <uploadSpeed>300</uploadSpeed>
        <downloadSpeed>0.2</downloadSpeed>
        <serverComputingSpeed>500</serverComputingSpeed>
    </server>
</serversList>
```

Appendix B: serversTaskSeq.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!-- Data size is in KB-->
<!-- Latency & Computing Time are in ms-->

<serversTaskSeq>
    <!-- Cloud -->
    <taskSeq>
        <task>
            <dataToProcess>200</dataToProcess>
            <arrivalTime>0</arrivalTime>
            <deadline>3</deadline>
            <waitTime>0</waitTime>
            <uploadStartTime>0</uploadStartTime>
            <computingStartTime>0.6</computingStartTime>
        </task>
        <task>
            <dataToProcess>500</dataToProcess>
            <arrivalTime>1</arrivalTime>
            <deadline>3</deadline>
            <waitTime>0</waitTime>
            <uploadStartTime>1</uploadStartTime>
            <computingStartTime>1.6</computingStartTime>
        </task>
        <task>
            <dataToProcess>300</dataToProcess>
            <arrivalTime>1.3</arrivalTime>
            <deadline>3</deadline>
            <waitTime>0.3</waitTime>
            <uploadStartTime>1.6</uploadStartTime>
            <computingStartTime>2.2</computingStartTime>
        </task>
    </taskSeq>
```

```
    <!-- Edge 1 -->
    <taskSeq>
        <task>
            <dataToProcess>400</dataToProcess>
            <arrivalTime>0.1</arrivalTime>
            <deadline>3</deadline>
            <waitTime>0</waitTime>
            <uploadStartTime>0.1</uploadStartTime>
            <computingStartTime>0.4</computingStartTime>
        </task>
        <task>
            <dataToProcess>200</dataToProcess>
            <arrivalTime>0.3</arrivalTime>
            <deadline>3</deadline>
            <waitTime>0.1</waitTime>
            <uploadStartTime>0.4</uploadStartTime>
            <computingStartTime>0.7</computingStartTime>
        </task>
    </taskSeq>
    <!-- Edge 2 -->
    <taskSeq>
        <task>
            <dataToProcess>600</dataToProcess>
            <arrivalTime>0.2</arrivalTime>
            <deadline>3</deadline>
            <waitTime>0</waitTime>
            <uploadStartTime>0.2</uploadStartTime>
            <computingStartTime>0.6</computingStartTime>
        </task>
        <task>
            <dataToProcess>300</dataToProcess>
            <arrivalTime>0.6</arrivalTime>
            <deadline>3</deadline>
            <waitTime>0</waitTime>
            <uploadStartTime>0.6</uploadStartTime>
            <computingStartTime>1</computingStartTime>
        </task>
    </taskSeq>
</serversTaskSeq>
```

Appendix C: taskList.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<tasks>
    <task>
        <dataToProcess>300</dataToProcess>
        <arrivalTime>0.4</arrivalTime>
        <deadline>3</deadline>
    </task>
    <task>
        <dataToProcess>600</dataToProcess>
        <arrivalTime>0.3</arrivalTime>
        <deadline>3</deadline>
    </task>
    <task>
        <dataToProcess>800</dataToProcess>
        <arrivalTime>0.2</arrivalTime>
        <deadline>3</deadline>
    </task>
    <task>
        <dataToProcess>500</dataToProcess>
        <arrivalTime>0.1</arrivalTime>
        <deadline>3</deadline>
    </task>
    <task>
        <dataToProcess>300</dataToProcess>
        <arrivalTime>0.3</arrivalTime>
        <deadline>3</deadline>
    </task>
    <task>
        <dataToProcess>700</dataToProcess>
        <arrivalTime>0.5</arrivalTime>
        <deadline>3</deadline>
    </task>
</tasks>
```