National College of Ireland

EP-MPCHS: Edge Server based cloudlet offloading using Multi-Core and Parallel Heap Structures

MSc Research Project
MSc Cloud Computing

Rajkumar Nagulsamy
Student ID: x22102817

School of Computing
National College of Ireland

Supervisor: Punit Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rajkumar Nagulsamy
Student ID:	X22102817
Programme:	MSc Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Punit Gupta
Submission Due Date:	16/09/2024
Project Title:	EP-MPCHS: Edge Server based cloudlet offloading using Multi-Core and Parallel Heap Structures
Word Count:	7038
Page Count	16

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rajkumar Nagulsamy

Date: 12th August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

EP-MPCHS: Edge Server based cloudlet offloading using Multi-Core and Parallel Heap Structures

Rajkumar Nagulsamy
x22102817

Abstract

The increasing computational demands of mobile applications, like image caption generators and Google Lens, result in higher memory and RAM usage. Thus, to offboard the computational workload of these applications cloud-based edge server frameworks has been in demand lately. In the age of cloud-based computing, the study aims to create a hybridized cloudlet placement algorithm that caters to reducing latency, increasing bandwidth, reducing network flow pressure, and optimizing the edge server resources. The primary condition of the utilized placement algorithm is to prioritize the cloudlets enabling the reduction of latency, increase of bandwidth, and CPU utilization.

This study proposes Edge Priority Placement using Multi-Core and Parallel Heap Structures (EP-MCPHS) utilizing the min heap prioritization technique to deduce the placement of each cloudlet. This incorporates the priority queue-based resource allocation system which ensures that the optimal process is selected to the minimum available edge server allowing the reduction of resource utilization, increased latency, decreased network flow, and enhanced bandwidth. The algorithm also reinforces the technique with a multi-latent parallel head provisioning or Ph.C. with parallel processing allowing reduced process starvation for the non-processing cloudlets. The EP-MCPHS reduces the end time for cloudlet processing by 27.36%, increases bandwidth by 71.27%, and data flow by 24.81%.

The study incorporates the SimPy framework for simulation testing using the EdgeSimPy framework for edge server simulation. The study compares the results achieved by the architecture with the Min-Max fairness algorithm. It provides statistical testing across time intervals to showcase the ability of the placement algorithm even with increasing workload.

1 Introduction

In today's landscape of Cloud Computing, mobile technologies have started utilizing applications with high computational needs. Hence, task–resource management has become a crucial research domain for mobile resource offloading. Throughout the last ten years, cloud computing has emerged as one of the most appealing alternatives for hosting applications via the Internet infrastructure. Computing in the cloud is based on the use of centralized computing resources located in data centers. This practice alleviates the burden of infrastructure administration and enables service providers to provide cloud services at reasonable costs for both people and organizations.

Cloud Computing's role as a catalyst for digital transformation has led to mobile technologies having a multitude of use cases in recent times, including AR (Augmented Reality), designing mobile applications, cross-platform gaming, and so on. These scenarios continue to raise the standard in terms of how people utilize software programmes in their day-to-day lives. With the emergence of these software solutions, it has become apparent that the concentration of cloud computing to data centers around a small number of global large-scale infrastructures (like Google, Azure, and Amazon) has resulted in the setup facing latency issues due to geographical distance from end devices. This distance causes reaction times that are incompatible with the real-time needs of the applications in question.

Therefore, the use of edge computing and its seminal role in offloading mobile application task computations is unavoidable. Cloud-native applications like Amazon Web Services or AWS and Google Cloud Platform or GCP for task offloading require high latency and bandwidth and are constrained to the unforeseen case of service unavailability. The usage of Cloudlet or a small-scale computationally viable resource center available at the edge of the network peripheral zone is the more plausible option. The VM–based cloudlet offloading proposed

in this study takes advantage of small-scale resource management procedures, rather than high-scale centralized cloud computing data center environments like AWS or GCP.
The distributed computation-based system leverages resources by creating a highly available cloudlet processing application using edge-scale devices by offloading in real-time. The procedure highlights the enhanced high computing tasks to be split in computational parallel pipelines which can be performed across edge devices as a form of D2D or Device to Device cloudlet offloading, and the final output can be computed in a centralized fashion at the remote cloud highly available offloading technique. The availability of data sources nearby is the main characteristic of the edge-based cloudlet computing mechanism or ECM technique. ECM facilitates the diversion of tasks between high-scale system units to AWS or GCP and edge-based offloading options. ECM infrastructure is capable of reducing carbon emissions from high computation needs of the task/resource allocation for mobile units as well as saving cost and decreasing latency for wide-scale VM infrastructure.
The ECM mechanism is capable of reducing the cost and latency extensively, however, it consists of two important factors

A. **Placement**: The inability to place the tasks in the best-suited edge computing device during D2D offloading, this is crucial as without appropriate placement the tasks could witness high latency and a generic decrease in bandwidth. Moreover, the proper utilization of ECM devices is ensured by appropriate placement algorithms
B. **Migration**: Further with the low-scale computational issues, starvation from the processes aligned across for offloading is a normalized phenomenon to reduce the inherent cloudlet or process starvation. Thus, a need for continuous migration is essential.

The presented study utilizes the Edge SimPy Simulation network for the technique utilization and formulation of placement and migration algorithms. The study therefore proposes three things:

A. The formulation of a priority-based muti-core heap structure to prioritize the cloudlets based on task/resource allocation allows reduction of latency and increased bandwidth
B. An objective function towards the use of both the cloudlets and edge server where the most optimal task or cloudlet is allocated to the edge server which can just match the requirements thus resource utilization is maximized.
C. The need to introduce parallel computation using threads and multi-latent systems to reduce process starvation in collaboration with the migration algorithm.

2 Literature Review

MEC or Mobile Edge Computing technique about cloudlet offloading offers to overcome issues of high computational needs by mobile applications and enhance the storage for low latency and quick response application-centric designs. However, the real-time offloading poses a challenge towards placement and migration of systems cloudlet across edge resources. To overcome this challenge, various research has advocated the use of deep learning algorithms to improve the offloading process in MEC networks. The solutions can be broadly categorized into 2 paradigms Intelligent Offloading mechanism discussed in section 2.1 and Task or Cloudlet offloading using Deep Reinforcement Learning in section 2.3

2.1 Mobile Devices and its Limitations

The primary challenge to real-time offloading MEC as discussed by Alghamdi et al. (2021) is the computation deficit within mobile networks for offloading MEC. Cardellini et al. (2016) propose a game theory solution to offload D2D cloudlets with central cloud and MEC architecture. Patel et al. (2015) emphasize network congestion and the use of MEC as a key technocratic use towards 5G systems as it allows faster network flow and could lead to network drops reducing congestion. Liu et al (2018) and Rego et al. (2019) discuss the inability to place the jobs in the most suitable edge computing device during D2D offloading, this is critical as without adequate placement the tasks might observe excessive latency and an overall drop in bandwidth. Moreover, the effective usage of ECM devices is assured by suitable placement algorithms. The low computational capabilities lead to fewer resources for utilization to fulfill each process which in turn creates process starvation leading to high latency, high data transfer, and low bandwidth. Due to low-scale computing resources and process starvation caused by the offloading of the cloudlet process. Thus, a requirement for ongoing migration is needed.
The necessity of transforming data with the help of computation requires high-end server with cloud computing capabilities. The research discussed above showcases the absence of High Computation and storage capacity solutions to offload cloudlet tasks (RG1).

2.2 Intelligent Offloading Mechanism

Xu et al. (2018) propose a deep neural mix of convolution and recurrent neural networks to create a hybrid Deep Reinforcement algorithm for an Intelligent offloading system. The CRNN or Convolutional Recurrent neural network for offloading process in MEC Networks. The proposed techniques offer a 25% lower energy use case in comparison to traditional offloading algorithms.

Yu et al. (2022) address the delay in computing resources and high latency constraints for Mobile Edge Computing or MEC. The paper proposes a Duel Deep Q network to optimize the scheduling of resource allocation for a multiuser, multicluster, and multi-tenant environment. The paper proposes an energy-efficient, loss-specific optimization algorithm that reduces energy use by 11.54% and 20.83% respectively for D2D and Generic Cloud scenarios. Further Vaiswan et al. (2016) propose a solution for the binary system increases latency and decreased bandwidth problem. The study discusses the pitfalls of the existing Monte Carlo distribution method, the study proposes Stochastic Enumeration using Decision tree-based cost optimization offering an Intelligent pursuit to effective sampling and placement of jobs based on computing resource parameters.

The above research showcases a significant research gap towards the concerns of congestion packet loss, process starvation, transport security, and latency threats (RG2).

2.3 Cloudlet offloading to Edge Cloud Networks using Deep Reinforcement Learning

Xuzehu et al.(2021) propose an intelligent deep reinforcement learning method for resource allocation towards cloudlet offloading for device-to-device (D2D). This study investigates the resource allocation issue that occurs during the offloading process. The MEC-oriented real-time energy-aware offloading system when considering the battery capacity of the data center. The paper showcases energy savings, close to 10^4 X 10^4 Joules. The study is nuanced over the MATLAB platform for the execution of the technique, providing a Deep Weighted parameter for values 0.2, 0.4, 0.6, 0.8, and 1.0. The study offers an offloading strategy for optimization, it takes the shortest latency and the cheapest computing cost as the optimization aim and then applies Q-learning to solve the problem.

Similarly, Ullah et al. (2023) studied edge computing for offloading from centralized cloud servers to D2D devices. It proposes a DRL or Deep Reinforcement Learning algorithm for offloading cloudlets based on available computation resources and device location. The study uses the Markov Decision Process to model the problem of latency and bandwidth in the underlying centralized cloud server system. The Study proposed DDQEC a Q learning-based Edge computing resource allocation algorithm using DRL.

The study by Mourita Mozib et al. (2023) works forth to increase the computation capabilities for mobile applications, high bandwidth and low latency. The paper identifies That MEC offloading requires optimization of offloading decisions in real time scenario. To achieve the desired goal the paper leverages the attribute of function from computation from edge computing networks. The distributed deep learning-based system for real-time offloading in MEC networks has tremendous potential to improve performance, scalability, and efficiency. The technique achieves a reduced energy utility of up to 40% and a 60% increase in latency across different simulations.

The critical question arriving from the discussion of the above paper is an apparent need to create a priority-based system capable of reducing network flow, enhancing bandwidth, and reducing latency. The current research uses a simplistic process to offload cloudlets leading to process starvation, and high latency thus requiring high Computation and storage capacity solutions to offload cloudlet tasks (RG1).

Research Gaps:

RG1: Need for High Computation and storage capacity solution to offload cloudlet tasks from Mobile applications that have low processing power and battery life to support such high computational need.

RG2: MEC network works extensively on wireless devices, thus congestion packet loss, transport security, and latency threats are quite common in this domain. Thus, centralized cloud systems often tend to cause significant obstruction to service or the quality of offerings. Thus, the use of Edge Computing for cloudlet remediation is essential.

RG3: In the context of RQ1 and RQ2, it's important to formulate a placement and migration algorithm capable of effectively reducing energy need, latency, and increasing bandwidth on ECM processes.

3 Methodology

The methodology describes the simulation pipeline followed throughout the paper. It delves into the use case of various physical layer attributes in section 3.1. In section 3.2 we describe the use case and ability of the Placement and Migration algorithm, Section 3.3 describes the proposed hybrid min heap queue placement creator. Figure 1 describes the high-level overview of the functionalities of the proposed Edge SimPy simulation. It consists of a variety of steps from the availability of simulation 1 and simulation 2. Which is passed onto the Simulator with the initializer path. It creates monitoring components such as the Edge Server, Network Switch, Base Station, and User data. The figure further showcases the placement and migration algorithms with experimental arbitrary parameters a and b. Where a can take values between 1, 2, 3 and b converges over 100, 200, 3. In the final stage monitoring analyses various data and metrics across network flow.

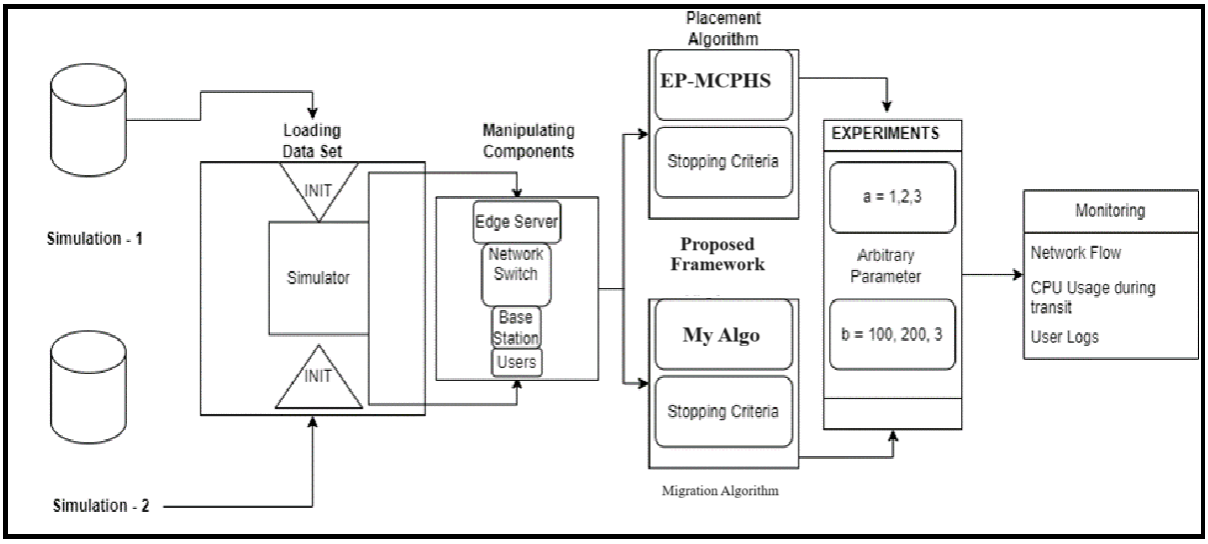

Figure 1 High-level diagram

3.1 Physical Layer and Attributes

The physical layer pertains abstraction necessary for maintaining users and resources defined below. Each component is provided with a piece of unique geospatial information which helps in locating and creating a 2D map of the same. Especially attributes like edge server and base station.

3.1.1 Base Station

Base Station is an integral and inherent component placed within the EdgeSimPy Simulation orchestration. It acts as a service platform allowing the users and edge servers to communicate via wireless communication using TCP protocol. It is typically expected to cover the entire map arena allowing the users to stay connected at all locations and geospatial coordinates of the mapping. It is responsible for capturing energy / power-based inputs and wireless latency of the orchestration grid. Further, it involves the presence of network switches and edge servers. Network switches are essential for wired connectivity with the users whereas Edge Servers allow cloudlet offloading or hosting abilities for the orchestration to perform smoothly.

3.1.2 Network Switches

Network switches act as a traffic flow originator. It provides weird connectivity for hosting infrastructure components and managing flow. It consists of ports, delays, clocks, bandwidth, etc. The network switch utilizes the placement algorithm. It will use the hybrid proposed algorithm to place the cloudlets on the Edge cloud orchestration and computation servers. It secures and protects the data transfer, network flow, and user requests as per QoS policies.

3.1.3 Edge Servers

Edge servers are scaled orchestration devices available for cloudlet offloading and computational adaptation. Each edge server consists of CPU cores, RAM, Disk space and MIPS performance is responsible for scaling and processing cloudlet offloading and co-hosted application runtime. It allows round-robin processing of each

cloudlet in the given time frames and onboards the highest priority of the cloudlet first. Further to save power consumption which inherently affects the latency of the node, it encompasses with decision to shut down the server to reduce power utilization.

3.1.4 Users

The Users are defined as client machines that offload tasks to the edge servers using the network switches, The users consist of geospatial information, which is the coordinate of the originating request, the user class defines delay in response from the edge servers, starvation of cloudlet, etc. Further, the user access pattern creates a cloudlet to be offloaded to the edge servers. The mobility pattern recognizes random pathways for the users.

3.1.5 Simulation Framework

The study makes use of a framework that is based on Python to construct the simulation cycle. The option other than utilizing EdgeSImPy was developing a new simulation architecture, however, to save time and reduce the amount of computing that is required for testing EdgeSimPy was used. EdgeSimPy offers the benefit of object simulation for base stations, edge servers, users, flows, etc. in addition to the usage of a predetermined simulation dataset. Additionally, it enables specified code samples to compute basic base instances for comparison and additional study.

To achieve the simulation of edge server cloudlet offloading the paper reviews various possible scenarios to compare the algorithms proposed. The paper utilizes a Python-based framework to develop the algorithm rather than creating a new simulation architecture in the interest of time and computation necessary for testing. The simulation framework adopted for this study is EdgeSimPy. It is a Python-based edge server simulation framework architecture that allows the simulation to be aided with pre-existing simulation data sets consisting of base stations, network switches, edge servers, and users. The simulation aids in the monitoring of data flow, resource utilization, bandwidth of the edge servers, and other parameters for comparison of the placement algorithm. EdgeSimPy which is an abstract class over EdgeSimPy is a generic tool based on Python for simulation of different parameters. It is generally used for resource utilization simulation and EdgeSimPy is a specific use case of EdgeSimPy for edge server and cloudlet offloading scenarios.

Each of the components within the simulation setup i.e. the base stations, network switches, and edge servers are connected where the user request is collected at the base station. The user requests and processes are then sent to the edge servers via the network switches. All three components are tightly coupled i.e. if one of the network switches is down the workload can be taken via the other switches available, this is a part of the fault-tolerant system. Similarly for the Edge Servers as well. However, the central joint of the system is the base station from where the switches are connected to the edge server, considering a fault tolerance scenario if the base station is done then the overall system performance will be severely affected. However, in a real-world scenario, the base stations are geo stations thus if a single node is down the traffic can be shared to a remote geo station.

3.2 Continuous Integration of Resource Allocation

CIRA or Continuous integration of resource allocation ensures that the processes are optimally offloaded onto the edge servers. The DVD and cloudlet offloading are synchronized for the cloudlet to utilize maximum resources thus ensuring the absence of underutilization of resources. CIRA is also responsible for reducing starvation for processes within an edge server orchestration setup. The main cause of resource unavailability in edge server simulation is either the high-priority processes that monopolize processing power or the movement of high-resource-utilization processes within the flow, resulting in increasing flow rates and subsequent process starvation.

The suggested approach establishes a multithreaded architecture that guarantees the execution of processes in a round-robin manner, with a priority cycle directly proportionate to computing requirements, hence reducing the flow rate while more sophisticated processes are being executed. The processes in the queue are either partly completed by multi-threading or, if not, their priority is elevated in sequential order based on time elapsed.

3.2.1 Network Architecture

The Network architecture utilizes the Physical Layer and Attributes namely, Base Station, Network Switches, Edge Servers, Users, and Simulation Framework to create a flow architecture diagram. It allows two workflows each consisting of services or cloudlets for offloading. Figure 2 describes the architectural setup of the complete orchestration. It connects the ES instances i.e. the edge servers, S1, S2, S3 i.e. the services using the network switches (L1, L2, L3,… L8). The CIRA is divided into two segments namely Placement and Migration Algorithms.

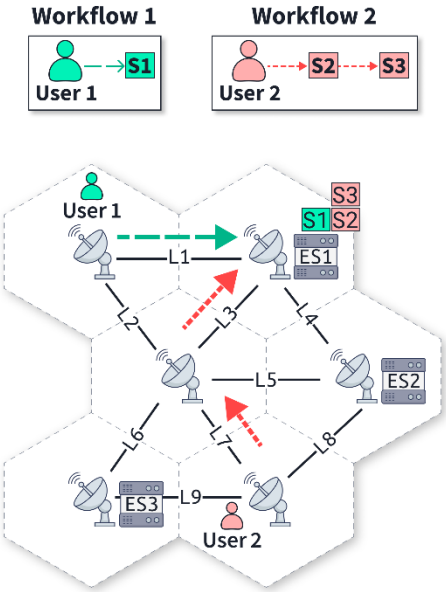

Figure 2 Network Architecture Diagram

3.2.2 Placement Algorithm

The EdgeSimPy framework is built with the resource allocation system mechanism consisting of multiple units. The placement algorithm is the rudimentary backbone of the resource allocation unit as it is responsible for placing the cloudlets from the respective user requests to the edge servers with the help of the network switches in transit. It is essential in optimizing the performance and resource utilization of the edge servers and orchestration as a whole. The placement algorithm is based on Cloudlets described in Equation i, Edge Servers in Equation ii, R_{ci} defines the resource of services i, C_{ej} defines the capacity of edge servers, $L_{ci,ej}$ defines the latency in between.

$$C = \{C_1, C_2, C_3\}$$

Equation i

$$E = \{E_1, E_2, E_3\}$$

Equation ii

$$\sum_{j=1}^{m} R_{ci} * x_{ij} < C_{ej}$$

Equation iii

$$\sum_{j=1}^{m} x_{ij} = 1 \ \forall i \in \{1,2,3, \ldots . n\}$$

Equation iv

Equation iii defines the minimum capacity needed for each service defined by X_{ij}. Equation iv defines that each service must be placed only on one edge server to prevent any redundant tasks.

3.2.3 Migration Algorithm

The migration algorithm allows continuous integration of Equation iii and Equation iv by re-evaluating starvation parameter and combines the possibility of task computation based on the dynamic mappings available and re-

balances the load on each edge server, in simpler terms it can be viewed as a rebalancing agent and the placement algorithm can be demonstrated as a load balancing unit. Placement algorithm is necessary for load distribution in the inception of task allocation however the migration technique is a constant running service that rebalances as and when needed.

The Migration algorithm ensures CICP or continuous integration and continuous processing for the cloudlets offloaded to the edge servers. It does so by optimizing the time benefits. The Migration algorithm is an optimization technique that tries to minimize the Equation v.

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \left(L_{ci,ej} + M_{ci,ej} + \propto T_{ci} \right) * y_{ij}$$

Equation v

Equation v describes the optimization function with $M_{si,ej}$ or the migration cost of moving Cloudlet i to Edge Server J, $\propto T_{ci}$ is the starvation parameter.

$$\sum_{j=1}^{m} y_{ij} = 1 \ \forall i \ \in \{ 1,2,3, \ldots . n \}$$

Equation vi

Equation vi describes the global cloudlet offloading parameter which should always be 1, as each cloudlet must be placed at an edge server.

$$\alpha T_{ci}$$

Equation vii

Equation vii defines that within the EdgeSimPy simulation mechanism, the migration strategies allow seamless transition of cloudlets between edge servers which is essential for service continuity while enabling continuous integration and continuous processing

3.3 Edge Priority Placement using Multi-Core and Parallel Heap Structures (EP-MCPHS)

EP-MCHPS or Edge Priority Placement using Multi-Core and Parallel Heap Structures combines multi-latent processing to benefit from the degeneracy minimum optimization unit of the heap function. The algorithm is divided into 3 segments Ph.C. or Parallel heap creation, Provisioning, and Ph.C. combination with parallel processing.

3.3.1 Parallel heap creation (Ph.C.) for Priority-based Resource allocation

A priority queue is utilized as a provisioning mechanism for the minimum optimizing which implements a minimum heap to allow the least possible resource to be allocated to the cloudlet ensuring the least wastage of resources. The Ph.C. algorithm uses Equation viii. The Heap used is a min heap for the edge server section based on the cloudlet computational resource needs.

$$Heap\ H_i = Heapify\ (\{ \left(\eta_{CPU,Memory,Disk} \right) \ | e_j \ \in ES\}) \quad \forall i \ \in \{ 1,2,3, \ldots . n \}$$

Equation viii

Equation viii describes the heap function responsible for the priority-based distinction of resource allocation. The function notes the minimum required edge server that can successfully offload the cloudlet. This allows the edge servers to be optimized, the resource utilization to be maximized and the migration flow rate of data to be low as the edge servers would be working at maximum capacity compared to the Min-Max fairness algorithm. Thus, the Priority-based allocation reduces latency, and data flow rate and maximizes resource optimization and bandwidth it also caters to creating a high operational edge server unit that can cater to on-demand service requests based on resource priority task allocation.

3.3.2 Provisioning

The provisioning algorithm verifies the ability of an edge server to cater to the cloudlet that is being sent. It compares the $\eta(x, y) the$ function is given by Equation ix

$$\eta(x, y) = \{1, \quad if\ (x.\varsigma > y.\varsigma \quad and \quad x.\mu > y.\mu \quad and \quad x.\delta > y.\delta)\ 0, \quad else$$

Equation ix

Which in turn creates the $Prov(e_j, c_i)the$ function is given by Equation x. These functions help compare the parameters given by the CPU or ç, RAM or μ, and disk space δ.

$$Prov(e_j, c_i) = \{1, \ if \ \ \eta(e_j, c_i) > 0\ 0, \qquad else$$

Equation x

3.3.3 Ph.C. combination with multi-core parallel processing

Parallel processing allows the Ph.C. to compute faster and more optimally. Equation xi showcases the parallel threads set available.

$$T = \{T, T_2, T_3\}$$

Equation xi

Equation xii showcases how each thread operates on a different cloudlet where C_k is a partition of Cloudlets C

$$C = \bigcup_{k=1}^{P} C_k$$

Equation xii

The time complexity of Ph.C. combination with parallel processing is given by Equation xiii

$$T_p = O\left(\frac{m \ . \ n}{P}\right)$$

Equation xiii

The proposed algorithm optimizes the placement algorithm by reducing the starvation of unused resources by using a Min heap with resource optimization. Further on it encompasses a parallel processing domain for the reduction of time computation enabling fast retrieval of cloudlets into edge computing processing via degeneracy pipelines.

The multi-core parallel heap structure divides the unit of resources into smaller units and aims to fulfill as many as possible cloudlets in a single unit of time whereas the heap emphasizes allowing a resource-based priority entry to the pipeline. It ensures that the process starvation of the cloudlet is reduced as it also has the migration technique to remap the starving cloudlets. The multi-core parallel processing caters to priority-based resource allocation as well as an automatic reduction in the starvation of cloudlets.

The proposed EP-MCPHS promises to reduce latency, and time spent to start processing and enhance faster traffic flow based on the ability of a heap-based processing speeding system.

Algorithm 1 EP-MCPHS

```
class EdgeServerWrapper:
  def __init__(self, edge_server):
    self.edge_server = edge_server

  def __lt__(self, other):
    # Compare edge servers based on their combined available resources (CPU, memory, disk)
    return (self.edge_server.cpu, self.edge_server.memory, self.edge_server.disk) < \
        (other.edge_server.cpu, other.edge_server.memory, other.edge_server.disk)

def provision_service(service, edge_server_heap):
  while edge_server_heap:
    try:
      edge_server_wrapper = heapq.heappop(edge_server_heap)
      edge_server = edge_server_wrapper.edge_server

      if edge_server.has_capacity_to_host(service=service):
        service.provision(target_server=edge_server)
        logging.info(f"Service {service.id} provisioned on edge server {edge_server.id}")
        return True
    except Exception as e:
      logging.error(f"Error provisioning service {service.id} on edge server {edge_server.id}: {e}")
  logging.warning(f"No suitable edge server found for service {service.id}")
  return False
```

```
def my_algorithm(parameters):
    services = [service for service in Service.all() if service.server is None and not service.being_provisioned]
    edge_servers = [EdgeServerWrapper(edge_server) for edge_server in EdgeServer.all()]
    heapq.heapify(edge_servers)

    with ThreadPoolExecutor(max_workers=10) as executor:
        futures = [executor.submit(provision_service, service, edge_servers) for service in services]

        for future in futures:
            try:
                future.result()
            except Exception as e:
                logging.error(f"Error in provisioning process: {e}")
```

Algorithm 1 EP-MCPHS showcases the algorithmic design of our placement algorithm

3.4 Testing and Evaluation Framework

The study incorporates the comparative analysis between the proposed Edge Priority Placement using Multi-Core and Parallel Heap Structures (EP-MCPHS) and the Min-Max Fairness algorithm. The Min-max fairness algorithm or MMFA was proposed in the base paper version of EdgeSimPy architecture. The study performs a deep analysis of the bandwidth, latency, resource utilization, and data flow rate across both techniques. The study further compares the mean and standard deviation under statistical analysis to showcase the evaluation of the EP-MCPHS architecture to the baseline Min-Max Fairness algorithm. The study provides a entailed discussion of how the reduction in data transfer and enhanced latency is the output of the incorporated minheap-based task priority system, It also compares the results in the upcoming section under the bar plots and statistical analysis cycles. It also creates a simulation parameter comparison for intervals of 5, 10, 15, and 20-second cycles showcasing the sturdity of the architecture across increasing time ranges and upcoming cloudlets.

4 Results and Discussion

4.1 Evaluation Metrics

In this study, we primarily compare the results among Start, End, Actual bandwidth and Data transfer let's discuss in detail the meaning and utilization of these.

Start and End time showcases the actual lapse of time to process the cloudlets. The edge server tries to minimize the difference between these as much as possible. The actual bandwidth refers to the potential to cater to more services which should be as high as possible. This also showcases the low latency of the edge servers. Finally, the data flow is a paramount effect of having the most optimal cloudlet being placed allowing for low data flow, accommodating high bandwidth and low latency for the algorithm.

4.2 Edge Servers

Table 1 Edge Server details showcase the instances, coordinates, CPU demand, RAM demand, disk demand, and services. These will be used for the below-mentioned simulations

Table 1 Edge Server details

Time Step	Instance ID	Coordinates	CPU Demand	RAM Demand	Disk Demand	Services
0	1	[0, 0]	0	0	0	[]
0	2	[0, 2]	0	0	0	[]
0	3	[6, 0]	0	0	0	[]
0	4	[1, 3]	0	0	0	[]
0	5	[7, 1]	1	1024	1017	[]
0	6	[6, 2]	0	0	0	[]
1	1	[0, 0]	6	12288	82	[]
1	2	[0, 2]	0	0	0	[]
1	3	[6, 0]	0	0	0	[]

1	4	[1, 3]	0	0	0	[]
1	5	[7, 1]	1	1024	1017	[]
1	6	[6, 2]	0	0	0	[]
2	1	[0, 0]	6	12288	82	[1, 2]
2	2	[0, 2]	0	0	0	[]
2	3	[6, 0]	0	0	0	[]
2	4	[1, 3]	0	0	0	[]
2	5	[7, 1]	1	1024	1017	[]
2	6	[6, 2]	0	0	0	[]
3	1	[0, 0]	6	12288	82	[1, 2]
3	2	[0, 2]	0	0	0	[]
3	3	[6, 0]	0	0	0	[]
3	4	[1, 3]	0	0	0	[]
3	5	[7, 1]	1	1024	1017	[]
3	6	[6, 2]	0	0	0	[]
4	1	[0, 0]	6	12288	82	[1, 2]
4	2	[0, 2]	0	0	0	[]
4	3	[6, 0]	0	0	0	[]
4	4	[1, 3]	0	0	0	[]
4	5	[7, 1]	1	1024	1017	[]
4	6	[6, 2]	0	0	0	[]
5	1	[0, 0]	6	12288	82	[1, 2]
5	2	[0, 2]	0	0	0	[]
5	3	[6, 0]	0	0	0	[]
5	4	[1, 3]	0	0	0	[]
5	5	[7, 1]	1	1024	1017	[]
5	6	[6, 2]	0	0	0	[]
6	1	[0, 0]	6	12288	82	[1, 2]
6	2	[0, 2]	0	0	0	[]
6	3	[6, 0]	0	0	0	[]
6	4	[1, 3]	0	0	0	[]
6	5	[7, 1]	1	1024	1017	[]
6	6	[6, 2]	0	0	0	[]
7	1	[0, 0]	6	12288	82	[1, 2]
7	2	[0, 2]	0	0	0	[]
7	3	[6, 0]	0	0	0	[]
7	4	[1, 3]	0	0	0	[]
7	5	[7, 1]	1	1024	1017	[]
7	6	[6, 2]	0	0	0	[]
8	1	[0, 0]	6	12288	82	[1, 2, 3, 4, 5, 6]
8	2	[0, 2]	0	0	0	[]
8	3	[6, 0]	0	0	0	[]
8	4	[1, 3]	0	0	0	[]
8	5	[7, 1]	1	1024	1017	[]
8	6	[6, 2]	0	0	0	[]

4.3 Simulation Results

The section showcases the results attained via the comparison of simulation dataset 2. Table 2 User Data with the statistical measures of count, mean, standard deviation, minimum, 25% quartile, 50% quartile, 75% quartile, and maximum, these are calculated across time steps and instance ID.

Table 2 User Data

Statistical Measure	Time Step	Instance ID
count	54	54
mean	4	3.5

std	2.60623	1.72386
min	0	1
25%	2	2
50%	4	3.5
75%	6	5
max	8	6

The Table 3 Network Flow using first come first serve showcases the statistical measures across time stamp, instance id, start, end time, source, target, bandwidth, and data transfer rate.

Table 3 Network Flow using Max-Min Fairness Algorithm

Statistical Measure	**Time Step**	**Instance ID**	**Start**	**End**	**Source**	**Target**	**Actual Bandwidth**	**Data to Transfer**
count	31.000000	31.000000	31.000000	15.000000	31.0	31.0	31.000000	31.000000
mean	4.612903	2.451613	1.225806	3.600000	5.0	1.0	3.811828	6.532258
std	2.275631	1.120676	0.425024	2.898275	0.0	0.0	1.511640	8.107522
min	1.000000	1.000000	1.000000	1.000000	5.0	1.0	2.000000	0.000000
25%	3.000000	1.500000	1.000000	1.000000	5.0	1.0	2.041667	0.000000
50%	5.000000	2.000000	1.000000	1.000000	5.0	1.0	4.166667	1.875000
75%	6.500000	3.000000	1.000000	6.500000	5.0	1.0	4.687500	12.875000
max	8.000000	4.000000	2.000000	7.000000	5.0	1.0	6.250000	23.750

Similarly in Table 4 EP-MCPHS Network Flow showcases the network route changes in bandwidth, timestamp, and data transfer.

Table 4 EP-MCPHS Network Flow

	Time Step	**Instance ID**	**Start**	**End**	**Source**	**Target**	**Actual Bandwidth**	**Data to Transfer**
count	42.000000	42.000000	42.0	26.000000	42.0	42.000000	42.000000	42.000000
mean	3.500000	4.000000	1.0	2.615385	5.0	2.714286	6.428571	4.916667
std	1.728527	2.024243	0.0	1.626700	0.0	1.686181	3.857320	7.548959
min	1.000000	1.000000	1.0	1.000000	5.0	1.000000	2.000000	0.000000
25%	2.000000	2.000000	1.0	1.000000	5.0	1.000000	2.000000	0.000000
50%	3.500000	4.000000	1.0	3.000000	5.0	2.000000	6.250000	0.000000
75%	5.000000	6.000000	1.0	4.000000	5.0	4.000000	6.250000	10.000000
max	6.000000	7.000000	1.0	5.000000	5.0	6.000000	12.500000	23.750000

Table 5 Comparison of EP-MCPHS and Max-Min Fairness

Algorithm	Statistical Analysis	Start	End	Actual Bandwidth	Data to Transfer
EP-MCPHS	**mean**	1.0	2.615385	6.428571	4.916667
	std	0.0	1.626700	3.857320	7.548959
	25%	1.0	1.000000	2.000000	0.000000
	50%	1.0	3.000000	6.250000	0.000000
	75%	1.0	4.000000	6.250000	10.000000
Max-Min Fairness Algorithm	**mean**	1.225806	3.600000	3.811828	6.532258
	std	0.425024	2.898275	1.511640	8.107522
	25%	1.0	1.000000	2.041667	0.000000
	50%	1.0	3.000000	4.166667	1.875000
	75%	1.0	4.000000	4.687500	12.875000

Table 5 Comparison of EP-MCPHS and Max-Min Fairness algorithm showcases that the proposed hybridized placement algorithm showcases lower start time statistics across mean, standard deviation, 25%, 50%, and 75% scores. Similarly, the end time is reduced to 2.615 in comparison to 3.6 seconds earlier. Further, the placement algorithm has a standard deviation of 0 meaning most algorithms are placed in the first unit of time during simulation. The utility of Edge Priority Placement using the Multi-Core and Parallel Heap Structures (EP-MCPHS) algorithm is the increase in bandwidth from 3.811828 in MAX-MIN FAIRNESS ALGORITHM to 6.428571. The algorithm also reduces the inherent data transfer rate as the most optimal task in case resources are running on the top edge server, henceforth smaller tasks need to be migrated. This is showcased by the parameter 4.91 mean in comparison to MAX-MIN FAIRNESS ALGORITHM 6.53, as well as the standard deviation of 7.54 during EP-MCPHS 8.10.

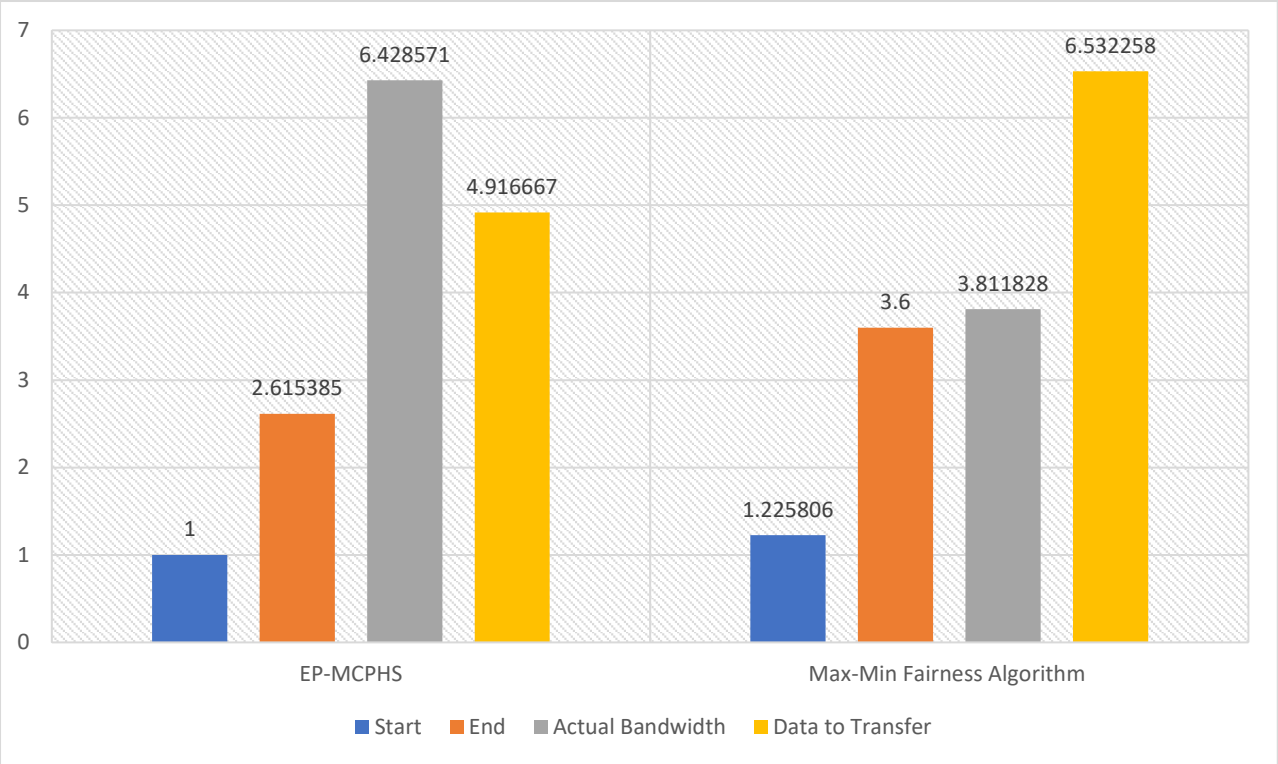

Figure 3 Comparison between EP-MCHS and Max-Min Fairness Algorithm

Figure 3 Comparison between EP-MCHS and Max-Min Fairness algorithm showcases the start, end, actual bandwidth, and data to transfer for the two algorithms. The figure also shows improvement in the mean of start, end, and data transfer wherein increase of data to transfer from 6.53 to 4.91 on EP-MCHS.

Table 6 Simulation Analysis across Interval Time

interval	Statistical	Start	End	Actual Bandwidth	Data to Transfer
5	**Mean**	1.0	2.615385	6.428571	4.916667
	Standard Deviation	0.0	1.626700	3.857320	7.548959
10	**Mean**	1.00087	2.615	6.428578	4.916667
	Standard Deviation	0.0715	1.62686	3.857320	7.548959
15	**Mean**	1.000081	2.6185	6.428561	4.916667
	Standard Deviation	0.00006	1.62579	3.857320	7.548959
20	**Mean**	1.000094	2.6159	6.428577	4.916667
	Standard Deviation	0.00051	1.626709	3.857320	7.548959

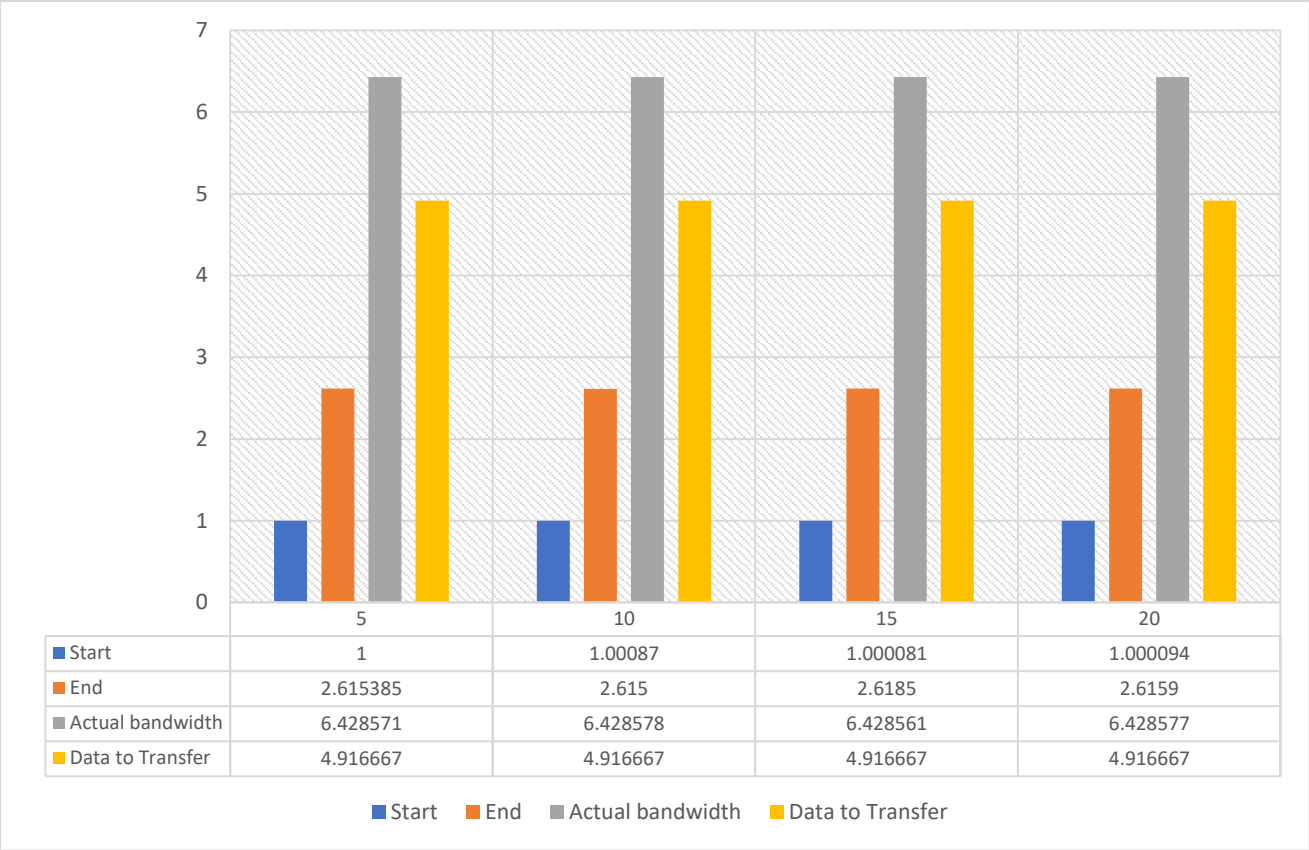

Figure 4 Simulation Analysis across interval time

Table 6 Simulation Analysis across interval time and Figure 4 showcases the meager change in the start, end, bandwidth, and data to transfer values within the framework of simulation across time intervals

5 Conclusion

The concept of edge computing is gaining popularity as a paradigm that may provide applications with minimal latency by moving processing from typical cloud data centers to the periphery of the network. To ensure that Edge Computing goes from being a promise to a reality, it is necessary to create effective approaches for resource management. This is because the expectations about the potential advantages of Edge Computing are growing significantly. The need for today's day and age mobile applications to be onboarded with applications needing, high computation, low latency, high bandwidth, etc. This leads to a scenario formation where in the fast age computation is needed, a general cloud server with high latency cannot cater to the needs of such high demand. Thus, to mitigate this edge computing resources are selected. In this study, the proposition of Edge Priority Placement using Multi-Core and Parallel Heap Structures (EP-MCPHS) allows the use of minimum heap with multi-processing techniques to reduce the workload from the unutilized edge servers, enhance resource utilization, reduce latency, and increase bandwidth to cater service request. This follows when the effective flow rate of the network drops as per our experiments in the section Simulation Results. This is because the current effective resources are allocated to the most optimal process leading to low data transfer in the network, thus further leading to a low latency and high bandwidth model. The final comparison with the first come first serve model architecture and the reduction of time interval-based cross-validation showcases the superiority of the modeling architecture of EP-MCPHS over other current state-of-the-art techniques for placement of cloudlet in edge servers. The algorithm also enhances the need for fast cloudlet offloading for high-demand mobile applications thus catering to the DVD or device-to-device offloading with cloudlet-to-edge server mapping as well. It increases the bandwidth from 3.811828 in the Max-Min Fairness algorithm to 6.428571. The algorithm also reduces the inherent data transfer rate to 4.91 for EP-MCPHS in comparison to MAX-MIN FAIRNESS ALGORITHM 6.53.

The domain implications and use cases of this study can be envisioned as:

A. Mobile applications and cloudlet drivers, allowing high-scale computations to be completed without a direct computing engine or cloud computing offset.

B. IOT-based home application, remote drivers, Car navigation, and operation devices. The IOT device can be placed in geographically sparse areas where in cloud computing latency would be very high. However, with edge computing and faster task onboarding the solution can empirically be placed in the local vicinity to fulfill the requested reduced latency.
C. Small-scale cloud offloading for simple single-mode servers and processes to be offloaded to edge servers.
D. Autonomous small-scale bot computation offloading, using which without a heavy server the bots can function with the cloudlet computation being distributed across edge servers.
E. High computational need of traffic camera facial recognition system can be fulfilled by synchronous edge computing allowing faster detection without localised control servers.

5.1 Future Scope

The study aims to create a dynamic algorithm capable of managing the load, reducing latency, increasing bandwidth, and decreasing network flow for cloudlet offloading into edge server simulation. The paper uses EdgeSimPy simulation to achieve the nuances described.
The application inherently creates a high demand, low latency, and low flow rate synchronized system which is important for adhering to and helping the existing real-world application. The algorithm not only maps the high resource need cloudlets to the appropriate edge servers, its capable of preventing process starvation and reducing flow rate with minimal computation increase than the baseline techniques used today.
In light of the above points, and considering the use case of the real-world application and high demand need, I envision the proposed system to be not only an enhanced version of the current allocation system but also a novel idea towards high availability computational zone or HACZ creation using edge computing.
The Future scope of this research can be showcased by the enhancement of deep reinforcement learning mechanisms where in a neural simulation for the architecture can be drawn and tested via neural agents. The simulation could allow to adopt to reduce latency and resource utilization. This would also allow for a non-parametric algorithm that could replace the current parametrized setup and create global loss functions for the calculation of algorithm fitness like the studies for feature selection.

6 Reference

[1]. Roman, R., Lopez, J., Mambo, M. (2018). Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. Future Generation Computer Systems, 78, 680–698.
[2]. Shahzadi, S., Iqbal, M., Dagiuklas, T., Qayyum, Z. U. (2017). Multi-access edge computing: open issues, challenges and future perspectives. Journal of Cloud Computing, 6, 1–13.
[3]. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials, 19, 2322–2358.
[4]. M. Satyanarayanan, G. Klas, M. Silva, S. Mangiante, The seminal role of edge-native applications, in: 2019 IEEE International Conference on Edge Computing, IEEE, 2019, pp. 33–40.
[5]. M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vmbased cloudlets in mobile computing, IEEE Pervasive Computing 8 1125 (2009) 14–23.
[6]. H. Zhao, S. Deng, Z. Liu, J. Yin, S. Dustdar, Distributed redundancy scheduling for microservice-based applications at the edge, IEEE Transactions on Services Computing (2020).
[7]. J. Wang, Z. Feng, S. George, R. Iyengar, P. Pillai, M. Satyanarayanan, 1130 Towards scalable edge-native applications, in: ACM/IEEE Symposium on Edge Computing, 2019, pp. 152–165.
[8]. P. Souza, A. Crestani, T. Ferreto, F. Rossi, Latency-aware privacypreserving service migration in federated edges, in: International Conference on Cloud Computing and Services Science, 2022, pp. 288–295. 1135
[9]. P. Souza, T. Ferreto, F. Rossi, R. Calheiros, Location-aware maintenance strategies for edge computing infrastructures, IEEE Communications Letters 26 (2022) 848–852.
[10]. Yu, Z., Xu, X. and Zhou, W. (2022). Task Offloading and Resource Allocation Strategy Based on Deep Learning for Mobile Edge Computing. *Computational Intelligence and Neuroscience*, 2022, pp.1–11. doi:https://doi.org/10.1155/2022/1427219.
[11]. Vaisman, R., Kroese, D.P. and Gertsbakh, I.B. (2016). Improved Sampling Plans for Combinatorial Invariants of Coherent Systems. *IEEE Transactions on Reliability*, 65(1), pp.410–424. doi:https://doi.org/10.1109/tr.2015.2446471.

[12]. Li, X. (2021). A Computing Offloading Resource Allocation Scheme Using Deep Reinforcement Learning in Mobile Edge Computing Systems. *Journal of Grid Computing*, 19(3). doi:https://doi.org/10.1007/s10723-021-09568-w.
[13]. Ullah, I., Lim, H.-K., Seok, Y.-J. and Han, Y.-H. (2023). Optimizing task offloading and resource allocation in edge-cloud networks: a DRL approach. *Journal of Cloud Computing*, 12(1). doi:https://doi.org/10.1186/s13677-023-00461-3.
[14]. Mourita Mozib (2023). Distributed Deep Learning Based Framework to Optimize Real-Time Offloading in Mobile Edge Computing Networks. *International journal of science and research*, 12(6), pp.1812–1827. doi:https://doi.org/10.21275/sr23603125305.
[15]. Yang, H., Xu, C., Liu, S., & Zhang, J. (2020). A Privacy-Preserving Task Offloading Scheme for Edge Computing in the Internet of Things. IEEE Internet of Things Journal, 7(7), 6124-6136.
[16]. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015), 'Internet of things: A survey on enabling technologies, protocols, and applications', IEEE communications surveys & tutorials 17(4), 2347–2376.
[17]. Alghamdi, I. (2021), Computation offloading in mobile edge computing: an optimal stopping theory approach, PhD thesis, University of Glasgow.
[18]. Cardellini, V., De Nitto Person´e, V., Di Valerio, V., Facchinei, F., Grassi, V., Lo Presti, F. & Piccialli, V. (2016), 'A game-theoretic approach to computation offloading in mobile cloud computing', Mathematical Programming 157, 421–449. Hu, Y. C.,
[19]. Patel, M., Sabella, D., Sprecher, N. & Young, V. (2015), 'Mobile edge computing—a key technology towards 5g', ETSI white paper 11(11), 1–16. Liu, L., Z. Y. . S. L. (2019),
[20]. 'Task offloading and resource al location for mobile edge computing by deep reinforcement learning based on sarsa', Ieee Access . Mao, Y., Zhang, J., Song, S. & Letaief, K. B. (2017),
[21]. 'Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems', IEEE Transactions on Wireless Communications 16(9), 5994–6009. Rego, P. A.,
[22]. Trinta, F. A., Hasan, M. Z., de Souza, J. N. et al. (2019), 'Enhancing offloading systems with smart decisions, adaptive monitoring, and mobility support', Wireless Communications and Mobile Computing 2019.
[23]. Satyanarayanan, M., Bahl, P., Caceres, R. & Davies, N. (2009), 'The case for vm-based cloudlets in mobile computing', IEEE pervasive Computing 8(4), 14–23.
[24]. Wang, S., Zhang, X., Zhang, Y., Wang, L., Yang, J. & Wang, W. (2017), 'A survey on mobile edge networks: Convergence of computing, caching and communications', Ieee Access 5, 6757–6779.
[25]. Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S. & Zhang, Y. (2016), 'Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks', IEEE access 4, 5896–5907.
[26]. Souza, P.S., Ferreto, T. and Calheiros, R.N. (2023). EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies. Future Generation Computer Systems, [online] 148, pp.446–459. doi:https://doi.org/10.1016/j.future.2023.06.013.