Remittances Review
September, 2023
Volume: 8, No: 3, pp.795- 806
ISSN: 2059-6588(Print) | ISSN 2059-6596(Online)

Received : 10 July 2023, Accepted: 28 September 2023
DOI: https://doi.org/10.33282/rr.vx9i.84

Failure Handling for Real-time Games running at Edge Computing Environment using offloading

Nimra Aslam[1] Muhammad Mudassar* [1], Muhammad Altaf [1]

[1]Computer Science Department, COMSATS University Islamabad Vehari Campus, Vehari, Pakistan
* Corresponding author muhammad.mudassar@cuivehari.edu.pk

Abstract

Online video games are executed on edge devices that are limited in power, battery, and computation constraints. So, it is necessary to offload the tasks of these games to some other devices as the games cannot bear latency. Offloading a task to the cloud is the default choice but increases the latency and cost. To overcome these factors, a system based on the heuristic algorithm and fuzzy logic (FL) methodology is proposed for resource measurement at an edge node which offloads the tasks of games to other available edge devices in the edge network. A FL system will estimate parameters like battery, power, and memory of the device and generates the output showing whether to offload or keep the game on the device, the heuristic algorithm will find the best device for offloading by comparing the parameters of the available devices. Based on current research, the proposed system is applicable and it can be a remarkable addition to the research community and industry in the era of IoT games executing in an edge computing environment.

Index Terms—Edge Computing, offloading, fault tolerance.

Remittances Review
September, 2023
Volume: 8, No: 3, pp.795- 806
ISSN: 2059-6588(Print) | ISSN 2059-6596(Online)

I. INTRODUCTION

With the growth of Information Technology, the number of IoT devices is increasing every day. The Internet of Things (IoT) is a network of physically addressable objects that may interact and communicate with one another using the Internet as their common platform. Sensors, actuators, cloud services, protocols, and layers are just a few of the components that make up IoT architecture. The architecture of IoT for gaming consists of three layers as shown in Figure 1. Different levels of processing, sensing, and actuation capabilities are available for these things. [1].

There are various applications of IoT. Smart cities, smart rooms, smart agriculture systems, etc. are being used to facilitate human beings. It is estimated that by the end of 2030, the number of IoT devices will increase to 29.4 billion [2]. Applications running on IoT devices require communication technologies as well as the availability of a reliable and high-speed internet connection. It includes Augmented Reality (AR), Online Video games, and Video Streaming [3]. The IoT devices are limited in processing, storage, and battery constraints, it is necessary to offload the tasks of applications to some other nearby devices (edge devices), fog devices, or cloud devices for the smooth execution of tasks.

IoT devices having inadequate battery, power, and computation resources, execute their applications on cloud and fog. When an application is being executed on these devices, it is checked and measured periodically for battery, memory, and storage constraint. If anyone of the constraint is running out, the tasks of the application are shifted to the cloud or fog. If it requires high computation, then it is offloaded to the cloud. But if the device is losing power, the tasks are sent to fog instead of the cloud [4]. For example, if a person wearing a smartwatch is running, all the sensors sense the data and send it to the cloud or fog. The operations are performed on these sides and the results are sent back to the devices.

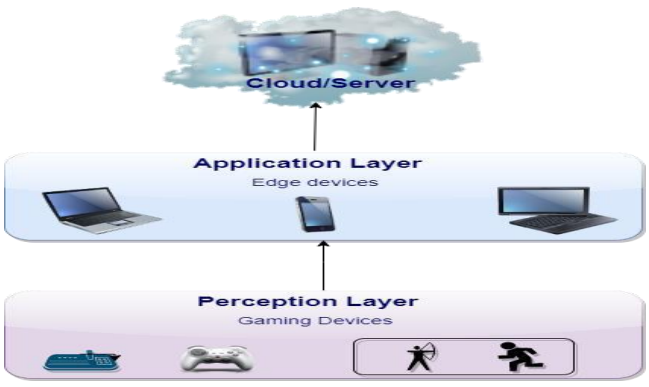

Figure 1: Games and IoT

Cloud computing is considered the most promising technique to tackle the massive growth of IoT devices in processing, storage, and resource constraints as it provides pay-as-you-go and on-demand resources. IoT devices that are limited in battery and computation constraints, offload their tasks to the cloud. However, resources in cloud computing are centrally located at a distance from the normal IoT devices that generate significant amount of data needs to be processed and delivered in real-time [5]. Offloading tasks to the cloud are not feasible for applications that are concerned with low latency because offloading tasks to the cloud devices increases the latency [6]. Moreover, it also increases the cost because the cloud has to maintain a large data center to provide the resources for computation and storage [7]. Therefore, tasks that require low latency are offloaded to nearby edge devices.

Fog computing is the solution to increase latency in cloud computing. It brings the cloud services near the edge devices. The IoT devices that require low latency, offload their tasks to fog. It serves as a filter between cloud and edge devices. Offloading is only the most important data to the cloud after being filtered at the edge. Latency and bandwidth problems associated with cloud computing are addressed using fog computing by bringing cloud services closer to edge devices [8]. The cost is increased because the task management design is sophisticated even though it improves network efficiency and reduces latency. Cloud and fog computing's drawbacks are overcome via edge computing. Edge computing provides an enormous number of devices that are very close to the IoT devices to decrease latency. In edge computing, large files are not uploaded/downloaded. Moreover, the pre-execution of tasks is not done, thus the overall performance and delay time of the applications are decreased [9], [10] the tasks are offloaded to the other edge devices based on some parameters and these parameters are measured using fuzzy logics.

The Fuzzy Logic (FL) provides reasoning which resembles how our brains operate more closely. It aims to simplify difficult problems to a level that is understandable. This helps to characterize the system's uncertainty and imprecision so that the imprecise information can be defined more logically and understandably [11]. FL rules are designed based on some parameters e.g., battery, memory, processing, etc. Then linguistic variables are assigned to these rules. The resources are measured based on these variables and offloading decisions are made. For their research work the fuzzy logic is used to detect the node contention or the bottleneck to decide about the offload. This architecture of the fuzzy logic based system is provided in Figure 2.

Remittances Review
September, 2023
Volume: 8, No: 3, pp.795- 806
ISSN: 2059-6588(Print) | ISSN 2059-6596(Online)

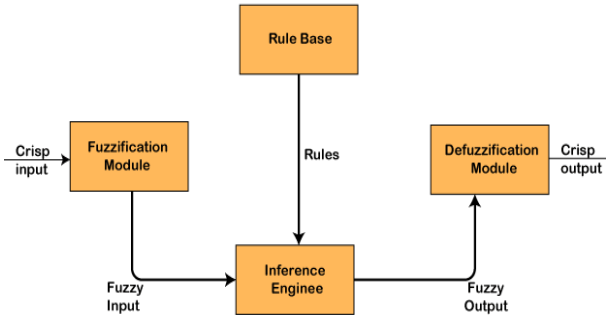

Figure 2: Architecture of fuzzy logic system [26]

Video games are one of the most complicated applications because of the real-time constraints and high processing demands. They require much computation and processing as they display the streaming video in a real-time manner while mobile or edge devices do have not much computation and processing power to execute high-intensity video games efficiently. Cloud gaming, which executes the entire game on specialized cloud servers, is one approach to this problem. The video is encoded and transmitted to the device by the servers after the game has finished rendering. [12]. But this increases the latency and the costs will be too high.

The proposed approach aims to offload the tasks of video games to high-computation edge devices. So, to decrease the delay time and for efficient utilization of the edge nodes, the tasks of the application (Online video game) are divided into a finite number of sets and are offloaded to the nearby high-computation edge devices. The tasks of video games are dependent. As it is an NP-hard problem, dependent tasks are offloaded using a heuristic approach [13]. The offloading is done based on some parameters (RAM, battery of mobile devices, CPU utilization, and mobility). The proposed approach adopts the fuzzy logic method to take decisions, on where to offload the tasks. This fuzzy logic works on the parameter mentioned above. Then these tasks are offloaded to the nodes based on the mentioned algorithm. Moreover, an architecture that includes the required components will be proposed.

To reduce the latency, a heuristic algorithm and fuzzy logic-based system is proposed. The fuzzy logic system will check the device parameters and checks if there is need of offloading. If offloading is required then the heuristic algorithm will check out the best device for offloading based on some parameters and then the game will be offloaded to that device.

II. Literature Review

These exist literature related to offloading, fault tolerance, edge computing environment and game execution. These previous research works assumes an application's tasks are independent from each other to reduce the dependencies and simplify offloading mechanism. But at real there exist dependencies between different tasks of and application. The predecessor has to complete its processing before the next dependent task. To handle these dependencies some authors used the call gap between different modules of the application which indicates the tasks that are dependent need some time to wait [14]. Scheduling for the applications having interdependencies between tasks is a tough one [15], [16].

Authors in [17] used priority queues where all of the tasks are completed orderly. The work in [18] proposes designing a cost-efficient system for offloading different tasks of multiple user based environment. This type of environment is common in games. The system in [18] also proposed to use heuristic offloading algorithm to reduce the cost making it cost effective for the constraint based environment. After the first task next offloading of tasks is based on the remaining energy of the relative node making the system to choose from high-cost devices to low-cost devices in term of energy reducing the system costs and ensuring energy requirement and completion time requirements.

Edge-cloud offloading is common and the independent task offloading is proposed by [19]. They targeted to reduce the time by using the fuzzy logic to schedule the task based on the device test using the fuzzy rules. Their task scheduling algorithm compares the capacity of the resource upto some thresh-hold values and decide to offload task once the threshold meets. The simulated results show that independent tasks were offloaded successfully. However, there exist lack of studies that have some mechanism to offload the tasks of the video games. The processing environment of games is somewhat different from the normal applications as these are normally online and multiplayers can also involve making the task dependencies different from normal application. It is highly required to model such scenario as the games are getting more famous. Currently game playing and developing is having high trends in the computing world.

A. Offloading to Fog Devices

Unlike edge cloud offloading, the fog have resources and can afford offloading to these devices and they can execute the task on behalf of the cloud. The research work [20] have discussed the fog nodes are small but have computation resources and some IoT device can offload tasks to the fog node. Down-link Non-Orthogonal Multiple Access (NOMA) is applied for efficient offloading. Using their methodology, a single IoT device can offload to some or multiple fog devices. The work in [21] considers uplink NOMA to offload to an edge device that is not considered previously, however they have not considered processing limitation of the nodes as the assumed unlimited processing which is not possible at real-time. Additionally, they care the profit of transferring a single task leading this to calculate the profit that is short term.

Authors in [24], presented a method to offload that improves the functionality of games using the Unity 3D environment. The game objects are offloaded to the edge server, game object frames that are rendered on the server based on some heuristic algorithm to play the game. These frames can be transfer back to the mobile. Their focus is onl

Volume: 8, No: 3, pp.795- 806
ISSN: 2059-6588(Print) | ISSN 2059-6596(Online)

Table 1 : Comparative study

Paper	Algorithm	Approach	Latency based offloading	Device heterogeneity	Cloud/ Edge/ Fog	Limitations
[18]	Heuristic Algorithm	Relative energy consumption	No	Yes	Edge to Cloud	Only focuses on reducing the system cost.
[19]	Fuzzy logic algorithm	Fuzzy logic system	Yes	No	Edge to Cloud	Only focuses on independent tasks.
[20]	Lyapunov optimization -based algorithm	Downlink NOZMA	No	Yes	Edge to Fog	Only focuses on offloading to fog devices.
[21]	Lagrange dual methodbased algorithm	Uplink NOMA	No	-	Edge to Fog	Only focuses on offloading a single task to limited computation devices.
[23]	Creates Virtual Machine	Library & Compiler (NDK)	No	No	Cloud	Increased latency
[24]	Heuristic algorithm	RPC and GVSP	No	Yes	Cloud	Focuses just increasing game capacity, latency is not handled.

to run the game the latency handling is not perfumed. Authors in [25] have discussed the latency reduction. They used common algorithms. The results show that DPSO is better task migration algorithm and more appropriate low latency applications like games.

B. Centralized and Distributed Task Offloading

Offloading decision can be processed in centralized or in distributed for edge computing environment. For centralized offloading [27] [28] a multiuser task offloading strategy is proposed by authors in [27], and centralized job offloading is advised in [28] ro reduce overall energy consumption the solution is based on deep-learning. However, the user satisfaction is not considered for the former approaches. Industrial IoT-Edge-Cloud computing scenarios multi-hop cooperative computation offloading is proposed for the case of distributed offloading by Hong et al. [29]. Wang et al. [30] used deep-learning to study the offloading problem and resource allocation. In general the recent research unable to cover effects of offloading within the edge computing environment and between the edge nodes themselves.

Resource allocation and offloading for three-way round-robin game with many uses and edge nodes is proposed by [31]. Their design involves a service-oriented resource allocation method for the edge computing environment. Gu et al. [32] suggested student project allocation game strategy based on matching game. These research papers take into account the issue related to resource allocation between users and edge nodes but some offloading mechanism is missing.

Table 3.1 represents the limitations that we have found in our study. The authors in [18] have used the heuristic algorithm and relative energy consumption approach for offloading. But they have made offloading from edge to cloud with the aim to reduce the system cost but offloading to the cloud actually increases the cost. offloading from edge to fog has been done in [20] however they do not focus on reducing the latency. The creation of a virtual machine using libraries and compiler (NDK) for offloading is proposed in [23], which is created on the cloud and ultimately increases latency as well as cost. We have proposed the offloading between edge devices. that will not only decrease the latency but will reduce the cost for the systems where cost is a major factor.

III. METHODOLOGY

A. The Fuzzy Logic System

The input given to the fuzzy logic system is CPU utilization, memory utilization, and battery usage. Using the Python library psutil, the values are taken directly from the system and given to the fuzzy logic system as input, and a fuzzy value for offloading is generated.

In fuzzy logic-based task offloading, data analysis is typically performed using a combination of methods and tools to handle fuzzy sets and decision-making processes. Here are some commonly used methods and tools for data analysis in fuzzy logic-based task offloading:

1. Fuzzy Sets and Membership Functions: these are the basic concepts in fuzzy logic. It represents and quantify linguistic variables and their degrees of membership in a fuzzy set. Membership functions define the shape and

characteristics of fuzzy sets, enabling the modeling of uncertain and imprecise data.

2. Fuzzy Logic Systems: These are used to process fuzzy rules and make decisions based on fuzzy logic. They consist of fuzzification, rule evaluation,and defuzzification stages. Fuzzification changes crisp input data to fuzzy sets, a membership function is used. Rule evaluation applies fuzzy rules to the fuzzy inputs and produces fuzzy outputs. Defuzzification changes back the fuzzy output to crisp for the decision.

3. Rule-based Reasoning: Rule-based reasoning is a key component of fuzzy logic-based task offloading. It involves defining a set of fuzzy rules that describe the relationship between input variables (such as CPU utilization, memory utilization and battery) and the offloading decision status (Offload, Keep). These rules are based on expert knowledge and domain-specific considerations. The rules are typically expressed using "IF-THEN" statements, where the antecedent (IF part) specifies the conditions, and the consequent (THEN part) determines the offloading decision.

4. Toolkits and Libraries: There are several toolkits and libraries available that provide functionality for working with fuzzy logic and performing data analysis in fuzzy systems. Some popular ones include:

1. Scikit-Fuzzy: A Python library that provides a wide range of fuzzy logic tools, including fuzzy sets, membership functions, fuzzy inference systems, and defuzzification methods.

2. Psutil Python Library: This Python library is used to utilize real-time values of the fuzzy logic input variables like CPU utilization, memory utilization, and battery from the device dynamically.

3. Matplot Library: This Python library is used to generate graphs with different graph styles for visual representation. We used this library for plotting the real-time system stats and comparison graphs.

The datasets consist of the input variables for the fuzzy logic system and devices for the heuristic algorithm. There are 3 input variables used as datasets in this research work. In fuzzy logic-based systems, datasets consist of input variables and corresponding output variables. In the case of task offloading, where fuzzy logic is used, the dataset may include input variables such as CPU utilization, memory utilization, and battery life. These variables provide information about the current state of the system and help determine the offloading decision.

In Table 2, each row represents a specific observation or instance within the dataset. The dataset captures various combinations of input variables and their corresponding achieved results. The columns in the table represent the input variables, including CPU utilization, memory utilization, and battery life. Each value within the table corresponds to the respective variable's measurement or observation for a specific instance.

The dataset encompasses a diverse range of instances, encompassing different system states at various points in time. These instances can be collected through measurements or simulations, enabling the representation of a comprehensive set of system states and their associated values for CPU utilization, memory utilization, and battery life.

Table 2: Dataset for input variables

CPU (%)	Memory (%)	Battery (%)
59	50	20
20	80	68
70	25	23
65	60	90
45	30	100

IV. PROPOSED ARCHITECTURE

The architecture diagram of our work is shown in Figure 3. This architecture is the implementation of the proposed work in which the gaming devices and edge nodes are connected with each other through a central layer which is the edge manager. The edge manager monitors the devices and manages the services of the nodes based on the data collected by these nodes. The detail of the components of the architecture and their interaction is given below;

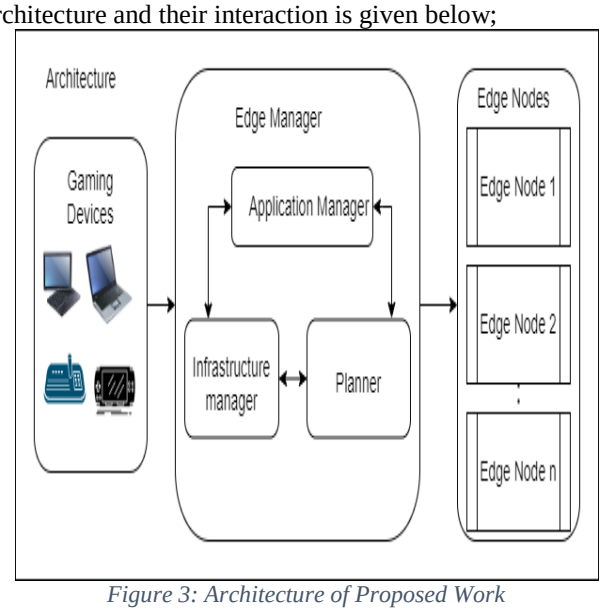

Figure 3: Architecture of Proposed Work

A. Edge Manager

Edge Manager (EM) is a centralized component that is responsible for planning, managing, and deploying the application services in the Edge-Edge system. In order to know the status of system resources (such as available and used), the number of edge devices, the tasks performed by their applications, and the offloading of game tasks, EM communicates with other architectural components. The components of EM are Application Manager, Infrastructure Manager, and Planner. EM is an independent entity and keeps all the edge nodes in its control to manage them.

1) Application Manager

It manages all the game applications that run on an edge device or mobile device. It keeps a record of all the game requirements including memory, CPU Utilization, latency constraints, etc.

2) Infrastructure Manager

It is key component of all the physical components of the Edge-Edge system. Like processors, IoT devices, and network devices for all the edge nodes. It provides the detail of the utilization of all the physical components of the EM.

3) Planner

This component mainly monitors the game tasks, detect the tasks failures due to the shortage of resources and make the offloading plans according to the need. The task offloading method that is being discussed in this study operates on this component and sends its output to the EM for execution.

B. Fuzzy Logic System

This module will determine the appropriate place to offload the gaming tasks by collecting information related to offloading tasks and CPU utilization. The description of the process of the fuzzy logic system is as follows:

1) Fuzzy Input Variables

Here for the fuzzy system the necessary inputs are defined. The required inputs are edge node CPU utilization, the battery of the edge device, and the memory of the edge node. We represented all these variables as linguistic variables: Low, Medium, and High as shown in Figure 4. These categories show the dynamic shift in the characteristics of the Edge-Edge architecture and applications' offloaded responsibilities.

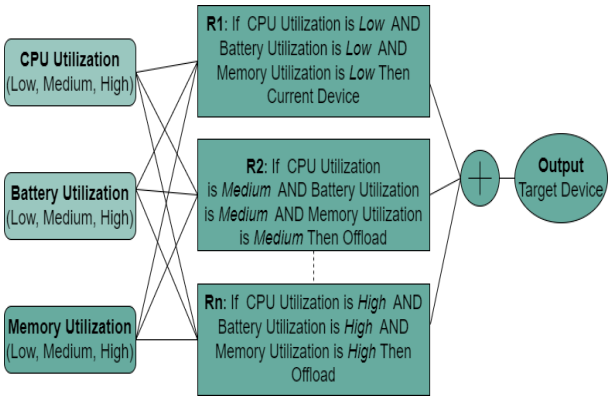

Figure 4: Proposed Fuzzy Logic System

a) CPU Utilization

This parameter represents the current utilization level of the CPU of the edge device on which the game is being executed. By this parameter, we can know about the capacity of that node. If it is highly utilized, then the tasks of the game needed to be offloaded to another edge device, having maximum resources to execute those tasks.

b) Battery Usage

This parameter represents the current battery level of the edge device on which the game is being executed. By this parameter, we can know about the remaining battery time of that node. If the battery of the device is running out, then the tasks of the game will be shifted to some other device with a high battery level so that the game tasks can be executed efficiently to reduce the latency.

c) Memory Utilization

This parameter refers to the current storage level of the edge device on which the game is being executed. By this parameter, we can know about the remaining memory of that node. If the storage capacity of the device is running out, then the tasks of the game need to be shifted to some other device with a higher storage capacity so that the latency can be reduced and the game tasks can be executed efficiently.

2) Fuzzification

In this stage, all the required values will be given as numeric input to the fuzzifier, then all these values will be assigned to the related linguistic variable in membership functions (e.g., Low, Medium, High), after that the fuzzy variables of all the parameters will be combined and evaluated in the fuzzy rules-base. This will produce the output known as the de-fuzzification.

a) Fuzzy Membership Functions

For each fuzzy input variable, the linguistic variable is quantified using the fuzzy membership function. In this work, we have used three functions. CPU Utilization, Battery Usage, and Memory Utilization have three variables (Low, Medium, and High). The membership function for CPU Utilization is shown in Figure 5. The values (0,15,35) show the Low linguistic variable, the Medium linguistic variable is shown by values (25,45,65), and the values (55,80,100) represent the High linguistic variable for CPU Utilization. The membership function for Battery Usage is shown in Figure 6. The values (0,15,35) show the Low linguistic variable, the Medium linguistic variable is shown by values (25,45,65), and the values (55,80,100) represent the High linguistic variable for Battery usage. The membership function Memory Utilization is shown in Figure 7. The values (0,15,35) show the Low linguistic variable, the Medium linguistic variable is shown by values (25,45,65), and the values (55,80,100) represent the High linguistic variable for Memory Utilization.

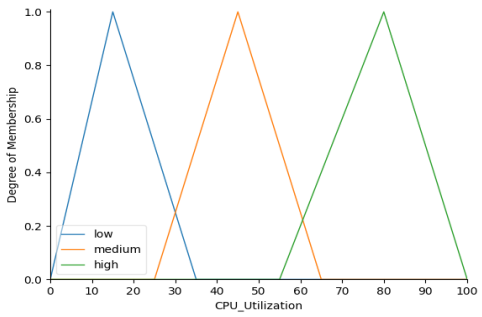

Figure 5:Membership Function for CPU Utilization

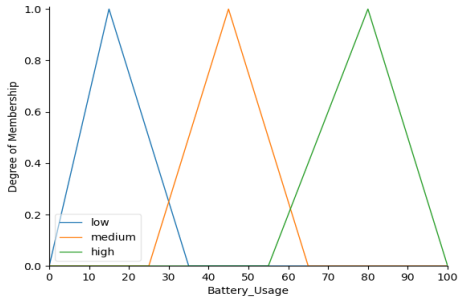

Figure 6: Membership Function for Battery Usage

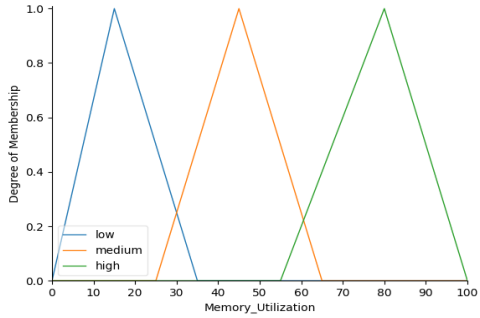

Figure 7: Membership Function for Memory Utilization

b) Fuzzy Rules-Base

It is a set of fuzzy rules that are similar to the way humans think. It is the collection of simple If-Then conditions that cover all the possible system behavior and conditions. These rules are very important and critical as they define the overall system performance. For example, if the CPU Utilization is Low, Battery Usage is Low, and Memory Utilization is Low too, then there is no need to offload and the tasks will be executed on that current device and the output will be Keep. Similarly, if the CPU Utilization is High, Battery Usage is High, and Memory Utilization is High too then the tasks will be offloaded to some other device and the output will be Offload. These outputs will be used at the defuzzification stage. Some examples of fuzzy rules-base are given in Table 5.1. The main aim of the work is to reduce the latency so that games can run efficiently.

Table 2: Fuzzy Rules

	Input Variables			Output
Rule	CPU Utilization	Memory Utilization	Battery Usage	Output Decision
1	L	L	L	Keep
2	L	L	M	Keep
3	L	L	H	Keep
4	L	M	L	Offload
5	L	M	M	Keep
6	L	M	H	Keep
7	L	H	L	Offload
8	L	H	M	Offload
9	L	H	H	Offload
10	M	L	L	Offload
11	M	L	M	Keep
12	M	L	H	Keep
13	M	M	L	offload
14	M	M	M	Keep
15	M	M	H	Keep
16	M	H	L	Offload
17	M	H	M	Offload
18	M	H	H	Offload
19	H	L	L	Offload
20	H	L	M	Offload
21	H	L	H	Offload
22	H	M	L	Offload
23	H	M	M	Offload
24	H	M	H	Offload
25	H	H	L	Offload
26	H	H	M	Offload
27	H	H	H	Offload

3) Defuzzification

Defuzzification is the process of turning the inference engine's fuzzy output into a clear output that may be used for control or decision-making. A fuzzy set that depicts the degree of membership of the output variable in each of its linguistic words is the output of the inference engine in fuzzy logic. The output variable Fuzzy Value has two fuzzy membership functions, Keep and Offload. The membership function for Fuzzy value is shown in Figure 8. The values (0,40,80) show the Keep linguistic variable and the values (80,95,100) represent the Offload linguistic variable for Fuzzy Value.

4) Defuzzification Techniques

Defuzzification can be accomplished using a number of techniques, including the centroid approach, mean of maximum (MOM) method, and smallest of maximum (SOM) method, among others. The choice of method relies on the particular application and each method has advantages and cons of its own.

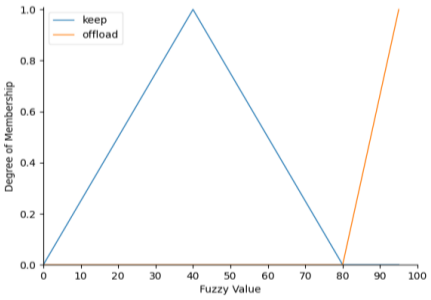

Figure 8: Defuzzifiction function

a) Centroid Approach

One of the most popular defuzzification techniques is the centroid approach. In this method, the crisp output is the

center of gravity for the set of fuzzy outputs. By averaging the values of the output variable over the range of its discourse universe, the center of gravity is determined. The weights represent the levels of the output variable's membership in each of its linguistic phrases. In our work, we have used the Centroid method for defuzzification.

b) MOM Approach

Another prominent technique for defuzzification is the MOM approach. The crisp result for this method is the average of the output variable's maximum values across the gamut of its discursive universe. This method can be helpful when the output variable's membership function contains numerous peaks.

c) SOM Approach

The SOM method is an easy defuzzification technique that chooses the crisp output as the value representing the highest degree of membership in the fuzzy output set. This approach is helpful when the output variable has a single peak in its membership function.

C. Heuristic Algorithm System

Heuristic algorithm is a technique or approach that provides a practical solution to a problem by using experience, intuition, or rules of thumb rather than guaranteeing an optimal or perfect solution. Heuristics are often employed when finding an optimal solution is computationally expensive or not feasible. In our work, the heuristic algorithm system is used to find the best device out of all the available devices based on some parameters like battery, memory and CPU utilization etc. it will check the parameters (CPU Utilization, Battery Usage and Memory Usage) of all the available devices and will compare them to find the best device with maximum value of all the parameters to offload the game tasks to. After finding the best device, offloading will take place.

D. Proposed Algorithms

We have provided two algorithms for offloading purpose. **Algorithm 1** will keep continuous track of the current device (on which the game is being executed) for the parameters (CPU_Utilization, Battery_Usage, and Memory_Usage). These parameters will be given to the Fuzzy Logic System (FLS) as fuzzy input. The FLS will continuously calculate the crisp value i.e., Fuzzy value *F* for these fuzzy inputs as shown in Figure 9. Then the calculated fuzzy value will be compared with the threshold value that is 80. If the fuzzy value will be greater than or equal to the threshold value then the Fuzzy Output *FO* will be *Offload*, otherwise there will be the output of *Keep*.

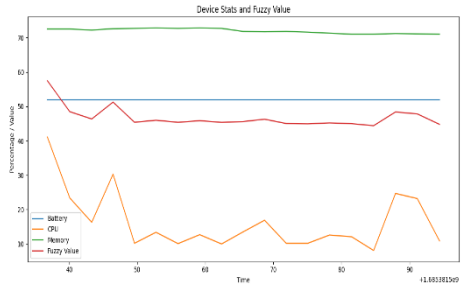

Figure 9: Continuous Device Stats and Fuzzy Value

Algorithm 1: Fuzzy Logic System (FLS) for Offloading

INPUT: Current Device D with its parameters $D_{CPU\_Utilization}$, $D_{Battery\_Usage}$ and $D_{Memory\_Utilization}$

OUTPUT: Fuzzy value F for offloading of Current Device D *and Fuzzy Output FO*

```
for Current Device D do
    while (Game is not completed) do
        Read Device D parameters
            D_CPU_Utilization, D_Battery_Usage and D_Memory_Utilization
        F = FuzzyLogicSystem (D_CPU_Utilization, D_Battery_Usage
            and D_Memory_Utilization);
        if F ≤ Threshold then
                FO = Keep
        else
            if F > Threshold then
                FO = Offload
            end if
        end if
    end while loop
end for
```

Algorithm 1 will take the device parameters such as CPU, memory and battery as input variables. Its output will be fuzzy value. It will continuously read the device parameters until the game is completed and gives these parameters to the FLS. The system will generate a crisp value i.e., fuzzy value against theses parameters that will be used by the **Algorithm 2** for offloading decision.

Algorithm 2: Heuristic Approach for choosing Best Device

INPUT: Devices D_i with their parameters $D_{CPU\_Utilization}$, $D_{Battery\_Usage}$ and $D_{Memory\_Utilization}$

OUTPUT: Select best device for offloading D_i

```
for all Devices in D_i do
    Read Devices D_i parameters
        D_CPU_Utilization, D_Battery_Usage and D_Memory_Utilization
    D_i = HeusisticSystem (D_1, D_2,..... D_n);
        if FO = Keep then
            Allocate Game on Current Device D
            else
            Offload Game to Best Device D_i
        end if
end for
```

The Algorithm 2 will be executed based on the heuristic approach to find the best device based on the parameters (CPU_Utilization, Battery_Usage and Memory_Usage). These parameters will be used as input. The algorithm will

continuously read the parameters of the available devices and compare them as well as choose the best device for offloading. Then offloading will take place to that device based on FO which is calculated using Algorithm 1. If FO will be Keep, the game will be executed on the current device. If the FO will be Offload, then the offloading will take place to the Best Device.

A. *Fuzzy Rules Graphs*

At first the results of the fuzzy rules are provided. Here we have shown only one case of offload decision and one case of keep decision. Other rule outputs were also tested and graphs were generated. The figure 10 provides the output of each component against Fuzzy Rule # 16.

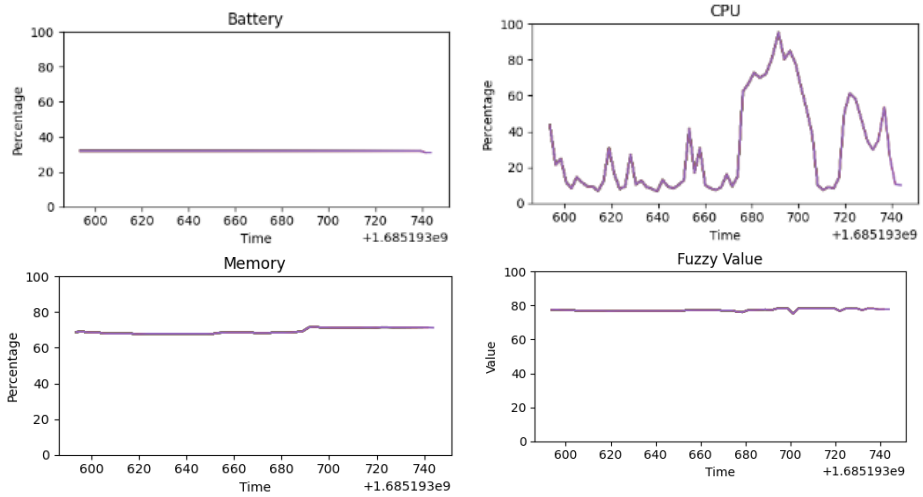

Figure 10: Output of Battery, CPU, Memory and Fuzzy value against rule number 16 to offload decision

V. RESULTS AND DISCUSSION

Real-time games are concerned with the timely execution of tasks in order to reduce latency. The existing system takes much time for executing the game tasks. The proposed system has achieved the efficient results as compared to the existing system.

if:Battery='Low' && CPU='Medium' && Memory='High' ➔ Fuzzy='Offload'

The figure shows that when the Battery of Current device is low, CPU utilization is medium and Memory utilization is high then Fuzzy value shows the value of 80 that is equal to the threshold value which means to offload.

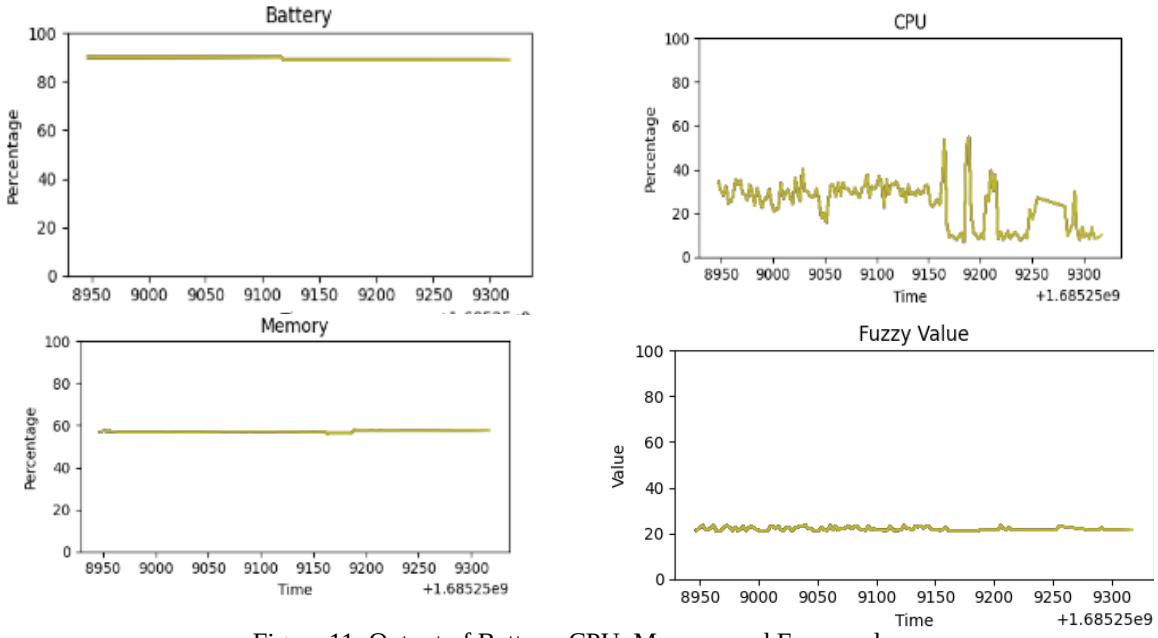

Figure 11: Output of Battery, CPU, Memory and Fuzzy value against rule number 6 to keep decision

Figure 11 depicts the Fuzzy Rule # 6 which is about to keep.

if: Battery='High' && CPU='Low' && Memory='Medium' ➔ Fuzzy='Keep'

The figure 11 shows that when the Battery of Current device is high, CPU utilization is low and Memory utilization is medium then Fuzzy value shows the value of 20 that is much less than the threshold value which means to keep i.e., there is no need of offloading.

B. Fuzzy based offloading versus no offloading

Latency is the network delay. We have compared the task offloaded to edge and without offloading. For the later case the application will be executed on Cloud once the task is failed. Delay occurs due to data transmission time over the network. Latency factors at cloud levels are high as expected as shown in figure 12.

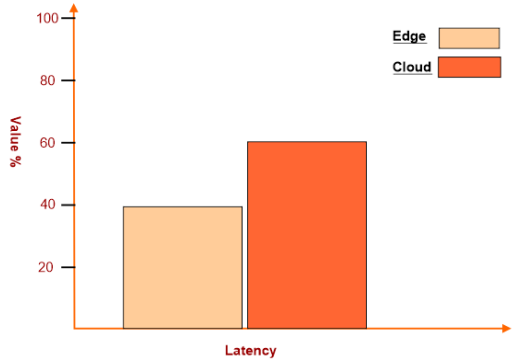

Figure 12: system latency when offloading is used versus no offloading

The second parameter is the bandwidth comparison. For no offloading case a high bandwidth is consumed at using the n cloud for the failed tasks, where for offloading active scenario very low bandwidth is used as the tasks are offloaded to nearby edge device. Figure 13 shows the bandwidth use for both scenarios. This shows that the proposed system is efficient in latency and bandwidth both.

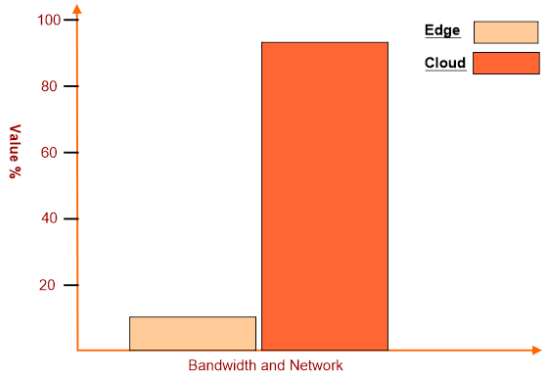

Figure 13: Bandwidth use comparison

The experiments for game task offloading were performed on the existing system that is a simple system without having any sort of offloading technique or any backup and our proposed system having heuristic based offloading. We have calculated the performance of the proposed system on the basis of task completion. In each setup we have executed different number of game tasks starting from single task to 5 task based game. And against each execution the execution time is compared. This result is presented in the figure 14.

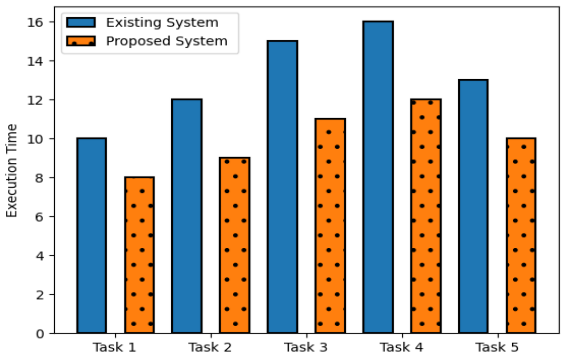

Figure 14: working with number of game tasks

The results show that offloading between edge devices reduces the overall latency as well as makes the system efficiency as compared to offloading to the cloud.

VI. CONCLUSION

Video games are getting famous day by day attracting the business community to invest in the computer based games. This require that a game should execute on the system that is available and provide low latency. As cloud have high latency edge computing can provide resources enough to execute games successfully. At same these edge devices are limited in power, battery, and computation constraints. So, sometime require to offload the tasks of these games to some other edge devices available in the vicinity as the games cannot bear latency. Offloading a task to the cloud is the default choice but increases the latency and cost. Hence a heuristic based offloading system is proposed that uses fuzzy logic at its base to decide when to make and offload decision. The results show that a game tasks can be executed in the ege computing environment successfully. This will help to increase the response time and the efficiency will also be increased. At the same time the edge nodes are resource limited nodes so the bottleneck can be detected successfully using lightweight fuzzy logic system. And once the bottleneck exists offloading to some resource free edge node can be performed. This can be handled efficiently as normally the redundant smart devices available in the vicinity can act as the edge device when required. This will reduce the overall latency and ensure the system availability as well.
In the future, more parameters like Network bandwidth, Task priority, and Latency delay can be considered to make the system more efficient. Moreover, the proposed work can be applied on the different fields i.e., in Real-time health systems etc. More resources like Graphical Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) can be considered for the games that require intensive computation.

REFERENCES

[1] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, "A Review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT)", in 2015 Internet

Technologies and Applications (ITA), pp. 219– 224, Sep. 2015, DOI: 10.1109/ITechA.2015.7317398.

[2] Vailshery, L.S. (2022) IOT connected devices worldwide 2019-2030, Statista. Available at: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (Accessed: November 6, 2022).

[3] Posey, B., Shea, S. and Wigmore, I. (2021) What is fog computing? - definition from Iotagenda, IoT Agenda. TechTarget. Available at: https://www.techtarget.com/iotagenda/definition/fog-computing-fogging (Accessed: November 7, 2022).

[4] Shekhar S, Gokhale A (2017) Dynamic resource management across cloud-edge resources for performance-sensitive applications. 2017 17th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID). IEEE, Madrid. pp 707–710

[5] Alrazgan, M. (2022) Internet of medical things and edge computing for improving healthcare in Smart Cities, Mathematical Problems in Engineering. Hindawi. Available at: https://www.hindawi.com/journals/mpe/2022/5776954/ (Accessed: November 7, 2022).

[6] Sahni Y, Cao J, Zhang S, Yang L (2017) Edge mesh: A new paradigm to enable distributed intelligence in the internet of things. IEEE access 5:16441–16458

[7] Aazam, M., Zeadally, S. and Flushing, E.F. (2021) Task Offloading in edge computing for machine learning-based Smart Healthcare, Computer Networks. Elsevier. Available at: https://www.sciencedirect.com/science/article/pii/S1389128621001298 (Accessed: November 7, 2022).

[8] Ashraf, M. et al. (2022) "Distributed application execution in Fog computing: A taxonomy, challenges, and future directions," Journal of King Saud University - Computer and Information Sciences, 34(7), pp. 3887–3909. Available at: https://doi.org/10.1016/j.jksuci.2022.05.002.

[9] Cong P, Zhou J, Li L, Cao K, Wei T, Li K (2020) A survey of hierarchical energy optimization for mobile edge computing: A perspective from end devices to the cloud. ACM Comput Surv (CSUR) 53(2):1–44

[10] Elgendy IA, Zhang W, Tian Y-C, Li K (2019) Resource allocation and computation offloading with data security for mobile edge computing. Futur Gener Comput Syst 100:531–541

[11] Hájek P (2013) Metamathematics of Fuzzy Logic, Vol. 4. Springer, Springer Netherlands

[12] F. Messaoudi, A. Ksentini and P. Bertin, "Toward a Mobile Gaming Based-Computation Offloading," 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1-7, doi: 10.1109/ICC.2018.8422518.

[13] Fan, Y., Zhai, L. and Wang, H. (2019) "Cost-efficient dependent task offloading for Multi users," IEEE Access, 7, pp. 115843–115856. Available at: https://doi.org/10.1109/access.2019.2936208.

[14] Y. Zhang, H. Liu, L. Jiao, and X. Fu, ''To offload or not to offload: An efficient code partition algorithm for mobile cloud computing,'' in Proc. IEEE 1st Int. Conf. Cloud Netw. (CLOUDNET), Nov. 2012, pp. 80–86.

[15] W. Zhang, Y. Wen, and D. O. Wu, ''Energy-efficient scheduling policy for collaborative execution in mobile cloud computing,'' in Proc. IEEE Conf. Conf. Comput. Commun. (INFOCOM), Apr. 2013, pp. 190–194.

[16] M. Jia, J. Cao, and L. Yang, ''Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing,'' in Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS), Apr./May 2014, pp. 352–357.

[17] M.-A. H. Abdel-Jabbar, I. Kacem, and S. Martin, ''Unrelated parallel machines with precedence constraints: Application to cloud computing,'' in Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet), Oct. 2014, pp. 438–442.

[18] Fan, Y., Zhai, L. and Wang, H. (2019) "Cost-efficient dependent task offloading for Multiusers," IEEE Access, 7, pp. 115843–115856. Available at: https://doi.org/10.1109/access.2019.2936208.

[19] Almutairi, J. and Aldossary, M. (2021) "A novel approach for IOT tasks offloading in edge-cloud environments." Available at: https://doi.org/10.21203/rs.3.rs-281532/v1.

[20] Wei, Z. and Jiang, H. (2018) "Optimal offloading in Fog computing systems with Non-Orthogonal Multiple Access," IEEE Access, 6, pp. 49767–49778. Available at: https://doi.org/10.1109/access.2018.2868894.

[21] F. Wang, J. Xu, and Z. Ding, ''Optimized multiuser computation offloading with multi-antenna NOMA,'' in Proc. IEEE Global Commun. Conf. (GLOBECOM) Workshop, Singapore, Dec. 2017, pp. 1–7.

[22] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), San Francisco, California, USA, June 15-18, 2010, 2010, pp. 49–62.

[23] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012, 2012, pp. 945–953.

[24] F. Messaoudi, A. Ksentini and P. Bertin, "Toward a Mobile Gaming Based-Computation Offloading," 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1-7, doi: 10.1109/ICC.2018.8422518.

[25] Alrazgan, M. (2022) Internet of medical things and edge computing for improving healthcare in Smart Cities, Mathematical Problems in Engineering. Hindawi. Available at: https://www.hindawi.com/journals/mpe/2022/5776954/ (Accessed: November 7, 2022).

[26] An overview of Fuzzy Logic System (no date) Section. Available at: https://www.section.io/engineering-education/an-overview-of-fuzzy-logic-system/ (Accessed: December 5, 2022).

[27] B. Baron, P. Spathis, H. Rivano, M. D. de Amorim, Y. Viniotis, and M. H. Ammar, "Centrally controlled mass data offloading using vehicular traffic," IEEE Transactions on Network and Service Management, vol. 14, no. 2, pp. 401–415, 2017.

[28] F. Jiang, R. Ma, C. Sun, and Z. Gu, "Dueling deep q-network learning based computing offloading scheme for f-ran," in 2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications, 2020, pp. 1–6.

[29] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial iot–edge–cloud computing environments," IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 12, pp. 2759–2774, 2019.

[30] Y. Wang, H. Ge, A. Feng, W. Li, L. Liu, and H. Jiang, "Computation offloading strategy based on deep reinforcement learning in cloud-assisted mobile edge computing," in 2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), 2020, pp. 108–113.

[31] Ma, S.; Guo, S.; Wang, K.; Jia, W.; Guo, M. A cyclic game for service-oriented resource allocation in edge computing. IEEE Trans. Serv. Comput. 2020, 13, 723–734

[32] Gu, Y.; Chang, Z.; Pan, M.; Song, L.; Han, Z. Joint radio and computational resource allocation in IoT fog computing. IEEE Trans. Veh. Technol. 2018, 67, 7475–7484.