Efficient Data Management in Cloud Fog Computing environment with tangential Data Control System

Kalka Dubey[1], Aman Raj[1], Abhishekh Kumar Singh[1], Satyam Kumar Tripathi[1], Mohit Kumar[2], Sukhpal Singh Gill[3]

[1]Department of Computer Science and Engineering, RGIPT Amethi, India
[2]Department of Information Technology, Dr B R Ambedkar NIT Jalandhar, India
[3]School of Electronic Engineering and Computer Science, Queen Mary University of London, UK

Abstract: The widespread adoption of Internet of Things (IoT) devices has spurred a surge in data-driven applications, presenting challenges in processing, storing, and managing IoT data within fog and cloud computing environments. To address these issues, we propose the "Foggy Weather Architecture with Tangential Data Control System." This architecture utilizes fog nodes as intermediaries, employing a Tangential Data Control System to supervise and filter IoT data, allowing only changed information to enter fog nodes. Additionally, our approach includes an algorithm that dynamically determines whether to use fog or cloud for data processing, prioritizing cloud processing during rapid data alterations and opting for fog computing during stable data periods. This integrated system optimizes resource usage, enhances scalability, and improves responsiveness for IoT applications within cloud environments.

Keywords: Data Control System; Tangential; Fog Computing, Fog to cloud computing(F2C), Quality of service

1. Introduction

The evolution of technology has led to a significant transformation in how we harness and process data, notably through the advent of Cloud Computing. Cloud Computing, with its centralized servers, has been pivotal in handling vast amounts of data and enabling various services accessible over the internet. However, the exponential growth of Internet of Things (IoT) devices has posed challenges to the conventional cloud-based approach.

The proliferation of IoT devices, encompassing an array from smart sensors to connected appliances and vehicles, has brought about a paradigm shift in data generation and utilization. These devices continuously generate extensive data streams, empowering data-driven decision- making and elevating our quality of life. Yet, managing and processing this avalanche of data has become a daunting task.

The conventional method of directly transmitting all IoT data to centralized cloud servers for storage and analysis is proving to be less practical over time. This approach strains the cloud infrastructure, raises concerns about data privacy, latency, and escalates operational costs associated with transmitting and storing vast volumes of data.

To counter these challenges, Fog Computing has emerged as a promising paradigm. Fog Computing extends the capabilities of cloud computing to the network's edge, enabling data processing and storage at proximate fog nodes. These nodes act as intermediaries between IoT devices and the cloud, alleviating issues related to data transmission, latency, and privacy.

In this context, our research introduces the **"Foggy Weather Architecture with Tangential Data Control System."** This innovative system draws inspiration from the concept of a tangent touching a circle at a single point. Similarly, our Tangential Data Control System supervises and directs data flow before dispatching it to the fog or cloud as deemed appropriate. Similar to how version control systems pass only the modified data, our Tangential Data Control System is designed to supervise and manage data flow, allowing only relevant and modified data to be transmitted to the fog or cloud. It also focuses on when to use fog or cloud according to the input data. The extensive discussion of this Data Control System is a central focus of this research paper, outlining its mechanisms and efficiencies in detail, contributing to the optimization of data handling. This system strategically curtails unnecessary data transmission, ensuring resource conservation and reduced operational overhead for both fog nodes and the cloud.

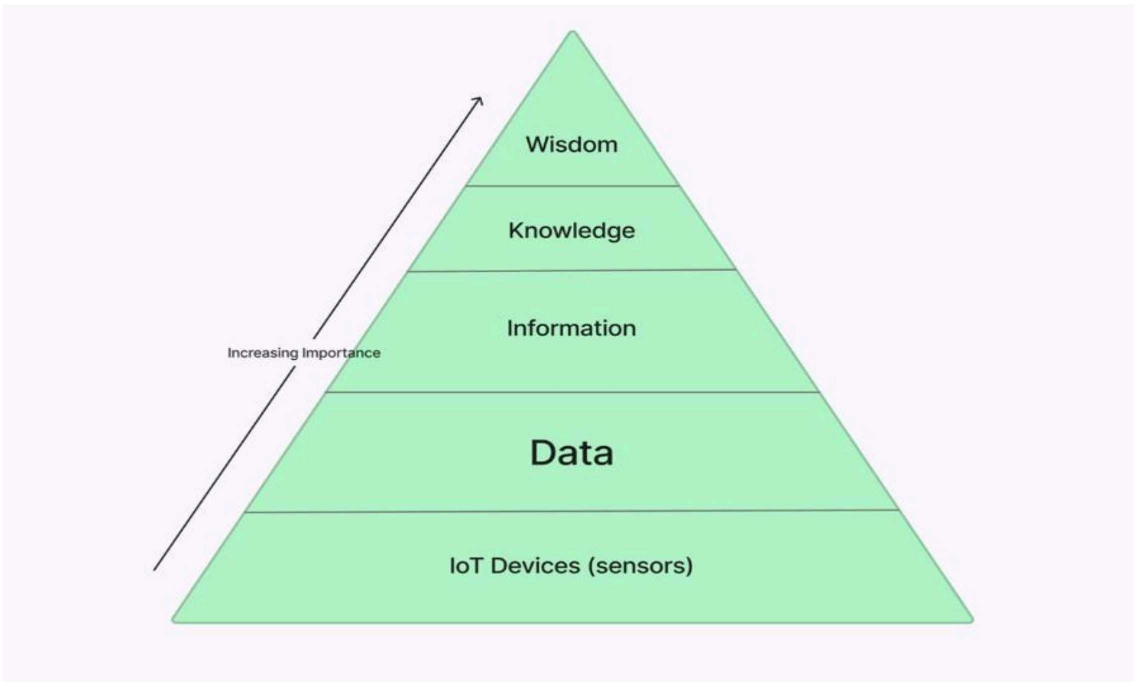


Figure 1

The term Foggy weather coined above means that there is introduction of Fog nodes in between the IoT devices and the Cloud. **The proposed framework is shown in figure 2.** This architectural framework aims to optimize the flow of IoT data, ensuring real-time insights for users while enabling in-depth analysis, which we term "Knowledge" and "WISDOM" (figure1) respectively [1].

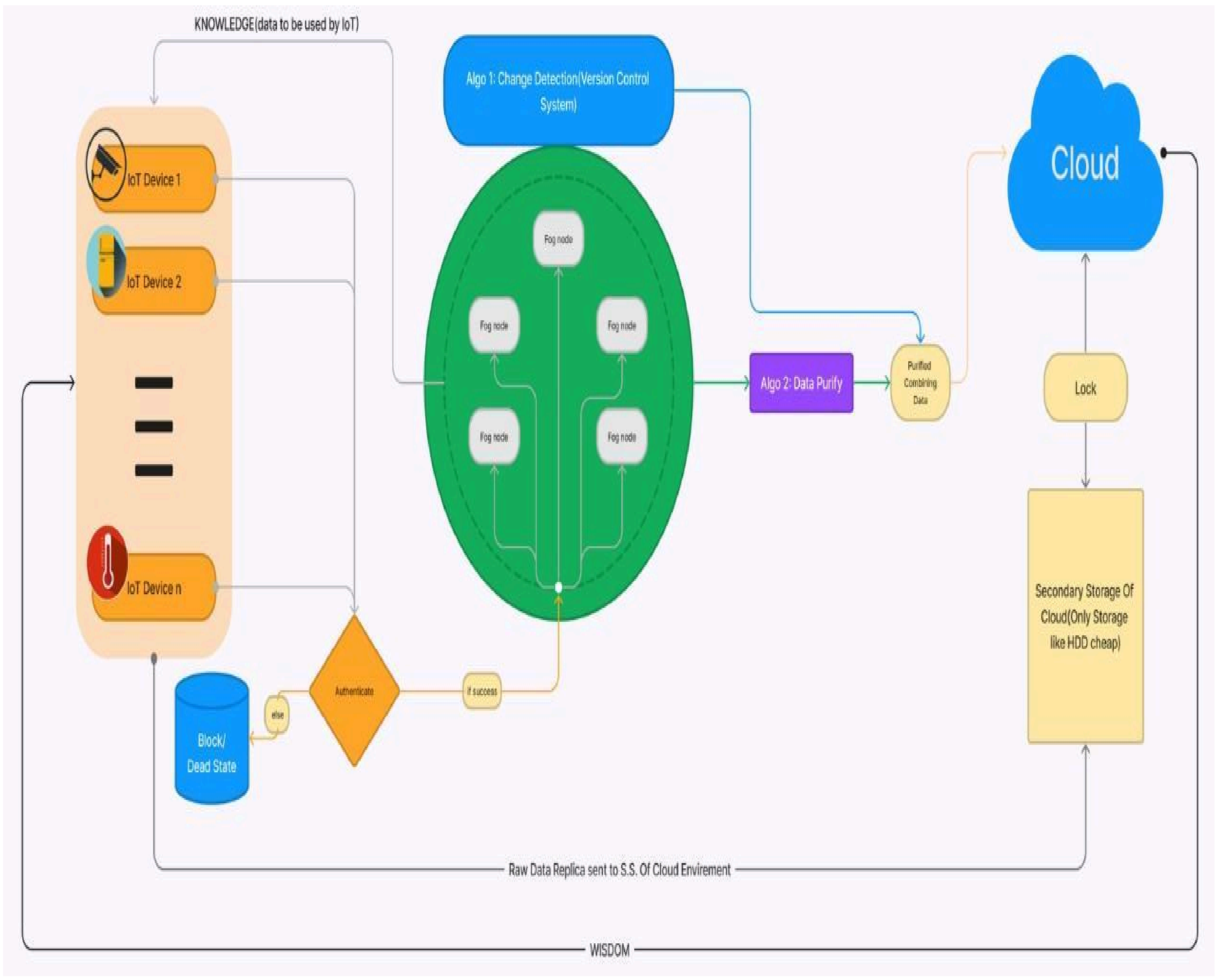

KNOWLEDGE(data to be used by IoT)
Algo 1: Change Detection(Version Control System)
Cloud
IoT Device 1
IoT Device 2
IoT Device n
Fog node
Fog node
Fog node
Fog node
Fog node
Algo 2: Data Purify
Purified Combining Data
Lock
Secondary Storage Of Cloud(Only Storage like HDD cheap)
Authenticate
if success
else
Block/ Dead State
Raw Data Replica sent to S.S. Of Cloud Envirement
WISDOM

Figure 2

The challenges we aim to overcome can be summarized as follows:

A) **Overloaded Cloud Infrastructure/Introduction of Fog nodes:** The increasing influx of IoT data to cloud servers can lead to overloading, resulting in latency issues and escalating operational costs. This research endeavours to alleviate this burden.

B) **Redundant Data Transmission/Data Control System:** Sending unchanged or redundant data to fog nodes and the cloud is inefficient and consumes network bandwidth unnecessarily. Our architecture introduces a solution to filter out such data.

C) **Resource Conservation (When and when NOT to use cloud)/Algorithm:** The fog nodes play a critical role as data intermediaries; their efficient use is vital to the overall system performance. Our architecture ensures that fog nodes operate optimally, extending the longevity of these resources.

The rest of the paper is organized in six sections. Section 2 presented the recent relevant work and motivation and significance of real-world application. We introduce in section 3 the proposed system model. Section 4 described the problem statement and formation. Section 5 deals with the proposed algorithm. In section 6, we validate the effectiveness of proposed work against other exiting work through computer-based simulation. We conclude our paper with the conclusive statement and the future aspect in the section 7.

2. Related Work

In recent research, advancements in data deduplication for fog computing have focused on adaptive and dynamic deduplication algorithms tailored for edge devices. Data deduplication is a technique used in computer science and information technology to reduce the amount of storage space needed by eliminating duplicate copies of data. These innovations aim to optimize resource utilization while considering the constraints of edge computing environments, such as limited processing power and energy.

Additionally, developments in data control systems have explored techniques for efficient differential data transmission, ensuring secure and reliable data synchronization among fog nodes. These advancements aim to minimize overhead while maintaining data consistency and integrity.

Koo et al. [2] introduced a novel method where a fog storage system operates on ciphertext C received from an end user. Using a Deduplication (Dedup) algorithm, this system checks for duplicates by comparing C with another stored ciphertext C'. If a duplicate is identified, the fog storage replaces the previous ciphertext with the newly received one for the same content, acknowledging the end user as the data owner (termed server-side deduplication). In cases where no duplicate is found locally, the fog storage initiates periodic or on-demand duplicate checks by sending only the last two parameters of the received ciphertext to the central cloud storage. The central cloud, employing the same Dedup algorithm, responds to the fog's query. If the result indicates unique data not stored in the cloud (resulting in a 'False' response), the fog forwards the first component, denoted as c1, to the cloud. The cloud then stores this component along with existing parameters in the format C = (c1, c2, c3), executing client-side deduplication. Following the deduplication process, regardless of the outcome, the fog storage system removes the ciphertext from its local storage. This collaborative approach between the fog storage and the central cloud optimizes storage use while ensuring data integrity and ownership. They have also stated that fog nodes can perform deduplication and provide data outsourcing services to data owners faster than the ones in the central cloud architecture. In contrast to centralized cloud storage, local fog devices situated closer to users, albeit having restricted storage capacities, exhibit quicker deduplication capabilities and can offer data outsourcing services more rapidly. These devices have temporary storage due to their limitations but can efficiently perform deduplication processes, providing faster services directly to data owners. Simultaneously, the centralized cloud, operating at a distant location, can effectively optimize storage resources on a global scale by receiving and managing unique data transmitted from various fog devices. This decentralized setup enables faster deduplication services at the edge, closer to users, due to reduced latency and quicker access to data. Fog devices, despite their limited storage, can swiftly process and eliminate duplicates, catering to data owners' needs promptly. Meanwhile, the central cloud, benefiting from receiving only unique data, optimizes its storage utilization by maintaining a consolidated repository of distinct information obtained from multiple fog devices distributed across different locations. This collaborative approach between fog devices and the central cloud allows for a more efficient and streamlined data management process, benefiting both the users and the cloud infrastructure.

Torabi et al. [3] introduced a comprehensive data life cycle model tailored for the IoT-fog-cloud environment. This model delineates distinct stages: data collection and command execution occurring at the device layer, data pre-processing at the fog layer, and basic analysis and collection feedback at the cloud layer. A significant aspect addressed in this model is the data replica placement problem, which aims to optimize the placement of delay-sensitive data across multiple replica nodes. The primary objective is to mitigate access delay and minimize network bandwidth utilization. Unlike traditional cloud computing where data is centralized, in this context, data is distributed among various replica nodes. Strategies for determining the number of replica nodes encompass full replication and partial replication approaches. The storage of data on fog devices for subsequent processing necessitates the implementation of cache management techniques. These techniques are crucial for minimizing the storage space occupied in end devices' caches while ensuring real-time response capabilities. The process of storing data involves optimizing cache usage to enable efficient data processing on fog devices without overwhelming their storage capacities. This balance ensures timely and effective data processing within the fog layer of the IoT-fog-cloud ecosystem.

Sinaeepourfard et al. [6] introduced a pioneering approach to data management within smart cities, focusing on a distributed hierarchical F2C (Fog to Cloud) system architecture. Their study delved into the data acquisition block, specifically examining the scenario in the city of Barcelona, estimating a generation of approximately 8GB of data daily from fundamental public sensors. Their research highlighted the efficacy of data aggregation within the data management process. Data aggregation encompasses various processing tasks such as gathering, reducing, and mixing data. The authors emphasized the importance of techniques like redundant data elimination and compression in managing data. Redundant data elimination, a simple yet effective solution, significantly reduced duplicated data, while compression, facilitated by data accumulation and delayed transmission, offered additional opportunities to reduce data volume. The study concluded that real-time data access was notably faster within the fog layer compared to centralized architectures. Additionally, the hierarchical F2C system notably reduced backbone network loads by enabling local access and utilization of data. They also highlighted the ease of applying various aggregation techniques to further diminish network data volume. The authors proposed adjusting data transmission frequency during periods of low traffic as a strategy to optimize network usage. Notably, through their model's exploration of aggregation techniques like redundant data elimination and data compression, the study showcased a substantial reduction in data volume. Specifically, they achieved a 75% reduction in data volume through redundant data elimination, and when coupled with data compression techniques, achieved an additional reduction rate of up to 78%.

Naas et al. [5] introduced iFogStore as a solution aimed at latency reduction within fog computing. Their approach takes into account the diverse features and heterogeneous nature of fog devices along with their respective locations. iFogStore primarily leverages storage and retrieval capabilities to effectively minimize latency. One of its key advantages lies in its ability to capitalize on data sharing among consumers who might shift locations or utilize devices with varying infrastructural capacities. Their research recommends a storage management architecture designed specifically for fog computing. This architecture follows a three-tier structure, facilitating real-time decision-making processes. By considering the characteristics of fog devices, their heterogeneity, and geographical distribution, iFogStore aims to optimize latency by utilizing storage and retrieval functionalities effectively. Moreover, the emphasis on data sharing among users with diverse infrastructural capabilities contributes to the overall efficiency of the proposed solution.

Kleinrock et al. [7] explained the M/M/1 model from various queuing theories. This model provides insights into the expected queue length and delays in the system. In fog computing, if the local fog nodes have a short queue and low delays, it might be favourable to process tasks locally. Here, customers arrive according to a Poisson process, are served one at a time by a single server, and the service times are exponentially distributed. This model is particularly useful for analysing systems where customers arrive randomly, are served on a first-come, first- served basis, and the service times are exponentially distributed. It provides insights into metrics such as the average number of customers in the system, the average time a customer spends in the system, and the utilization of the server.

Harrison et al. [8] did the stationary analysis of M/M/1 queuing model and found this model is considered stable only if $\lambda < \mu$. If, on average, arrivals happen faster than service completions the queue will grow indefinitely long and the system will not have a stationary distribution. The stationary distribution is the limiting distribution for large values of t. Various performance measures can be computed explicitly for the M/M/1 queue. We write $\rho = \lambda/\mu$ for the utilization of the buffer and require $\rho < 1$ for the queue to be stable. ρ represents the average proportion of time which the server is occupied.

Guillemin et al. [9] analysed Average number of customers in the system, they found that the number of customers in the system is geometrically distributed with parameter $1 - \rho$. Thus, the average number of customers in the system is $\rho/ (1 - \rho)$ and the variance of number of customers in the system is $\rho/ (1 - \rho)^2$. This result holds for any work conserving service regime, such as processor sharing.

Little et al. [10] proposed a queuing law known as Little's law which states that the long-term average number L of customers in a stationary system is equal to the long-term average effective arrival rate λ multiplied by the average time W that a customer spends in the system. Expressed algebraically the law is $L = \rho * W$. The relationship is not influenced by the arrival process distribution, the service distribution, the service order, or practically anything else. In most queuing systems, service time is the bottleneck that creates the queue. The result applies to any system, and particularly, it applies to systems within systems. For example in a bank branch, the customer line might be one subsystem, and each of the tellers another subsystem, and Little's result could be applied to each one, as well as the whole thing. The only requirements are that the system be stable and non-pre-emptive, this rules out transition states such as initial startup or shutdown. In some cases, it is possible not only to mathematically relate the average number in the system to the average wait but even to relate the entire probability distribution (and moments) of the number in the system to the wait.

3. Problem formation

The aim of this section is to improve the significant QoS parameters that directly impact the performance of the system. The problem is formulated mathematically to optimize the objective functions that accomplished the demand of end user as well as service provider without violating the sla. Let us consider a cloud datacenter that contains m number of heterogeneous physical machines, denoted as PM = $\{PM_1, PM_2, PM_3 \quad PM_m\}$, where k number of container can be deployed over each physical machine depends upon the availability of resources, represented as C=$\{C_1, C_2$ $C_3 \quad C_k\}$. Various types of services and library function can be installed over the PM to execute the microservices within defined time and cost. End users submitted n request in the form of microservices, denoted as MS= $\{MS_1, MS_2, MS_3 \quad MS_n\}$, where a microservice can communicate with other one and work independently to process the application. Each microservice MS_i required the resources from the PM for the completion of services, and represented in the form of tuples as $\{R^{CPU}$, R^{MM}, R^{BC}, $R^{Bw}\}$, Where R^{CPU} is required CPU, R^{MM} is main memory required, R^{BC} is budget constraint and R^{Bw} is required bandwidth. The proposed approach searches the best container to deploy the microservice and fulfill the resource requirement. Resource capacity of container is defined in equation 1-2

$$R^{CPU}_{i} = \eta * \boldsymbol{\rho} \qquad (1)$$

Where η denotes the number of core and $\boldsymbol{\rho}$ is size of or processing power of each core.

$$R^{MM}_{i} = NB_{MM} * \text{sizeof}(NB_{MM}) \qquad (2)$$

Where NB_{MM} represents the required main memory blocks and sizeof (NB_{MM}) is each block size. The time required to execute each microservices over cloud resources is defined in equation 3

$$ET_{MS_i} = \frac{SMS_i}{i} \qquad (3)$$

Where ET_{MS_i} is the execution time of microservice, SMS_i is the size of microservice. The service cost for the microservice (SCO_{MS_i}) based upon CPU and memory usage is calculated by equation 4 & 5

$$SCO^{CPU}_{MS_i} = R^{CPU}_{i} * \frac{ET_{MS_i} * SCO^{MS_i}_{CPU_{usage}}}{f_1} \qquad (4)$$

$$SCO^{MM}_{MS_i} = R^{MM}_{i} * \frac{ET_{MS_i} * SCO^{MS_i}_{MM_{usage}}}{f_2} \qquad (5)$$

Where $SCO^{MSi}_{CPU_{usage}}$ denotes the container cost in per unit time interval (f_1) based upon the utilization of CPU, and $SCO^{MSi}_{MM_{usage}}$ denotes the container cost based upon the utilization of main memory in per unit time interval (f_2). Total service cost is represented by the equation 6

$$SCO_{Total} = SCO^{CPU}_{MSi} + SCO^{MM}_{MSi} \qquad (6)$$

After calculating the execution time of microservices, we find the deployment time of each container (DT_{C_i}) over physical machine for the services with the help of applications and its components (microservices) by equation 7. Suppose an application (A_{app}) consists of α microservices and it required k container for executing in a time period t, deployment cost of container over PM is represented by DCO^{PM}_{i}

$$DCO_{C_i} = \frac{DT_{C} * DCO^{PM}}{f_3} \qquad (7)$$

If DT_{C_i}=0, then total time taken to execute the applications is calculated by the equation 8

$$COST_{A_{app}} = \sum_{i=1}^{\alpha} SCO_i + \sum_{i=1}^{k} DCO_{C_i} \qquad (8)$$

After deploying the container over the PMs, need to monitor the workload continuously to avoid the condition of over and underutilization. Hence, workload cannot be allocated more than the capacity of physical machines resources especially CPU and memory. Current workload over the physical machine can be calculated using the equation 9

$$Workload_j(CPU, MM) = \gamma * \sum_{i=1}^{k} R_i^{CPU}(t) + (1-\gamma) * \sum_{i=1}^{k} R_i^{MM}(t) \quad (9)$$

Where range of CPU and memory utilization are between 0 to 100 and γ is constant with range [0, 1]. The utilization of cloud resources can be found with the help of equation 9.

$$ResU_j = \frac{Workload_j(CPU,MM)}{Workload_j^{max}(CPU,MM)} * 100 \quad (10)$$

Where $Workload_j^{max}(CPU, MM)$ represents the maximum utilization of cloud resources. Our objective is to minimize the time and cost ($COST_{A_{app}}$), while maximizing the utilization of cloud resources ($ResU_j$) without any sla violation like deadline, budget etc. Considering the equation no. 8 & 10 to fulfill the objectives,

$$\text{Min } COST_{A_{app}} \quad (11)$$

$$\text{Max } ResU_j \quad (12)$$

Subject to: $COST_{A_{app}}$ BC_{Aapp} $<= R$

$$ResU_j <= 100 \quad (14)$$

$$\sum_{i=1}^{k} ResU_i <= ResU_j \quad (15)$$

$$Workload_j(CPU, MM) <= Workload_j^{max}(CPU, MM) \quad (16)$$

$$x_{i,n} \in \{0, 1\} ; \quad i \in C_k; \quad (17)$$

$$\sum_{i=1}^{k} x_{i,j}; \quad i \in C_k; \quad j \in PM_m \quad (18)$$

The above-mentioned constraints describe that the cost of application for the microservices can not be more than the defined budget, and utilization of cloud resource can not be more than 100% for any resource. Workload over the physical machine cannot be beyond the limit as shown in equation 16. Separate container is allocated for each microservice and several containers can run over a single physical machine based upon the configuration as defined in constraint 17-18.

4. System model

In this section, we discussed the proposed data control system model for the IoT-Fog-Cloud environment or in short, the foggy weather environment as discussed earlier in the introduction. The steps in which the data control system will work are as follows:

3.1 Authentication and Authorization**:**

To initiate our architecture, we address the critical issues of authentication and authorization. IoT devices generate copious data, necessitating the validation of users. An authentication system verifies the legitimacy of incoming data, permitting access exclusively to subscribed and authorized users while rendering non-subscribed users inactive (block or dead state).

3.2 Initial Data Transmission (t=0):

At the inception (t=0), IoT devices transmit their data to fog nodes as reference data. This initial data transfer establishes a baseline for subsequent comparisons and modifications, facilitating the detection of variations in incoming data.

(C) Tangential Data Control System:

Post-t=0, the data from IoT devices undergoes the tangential Data Control System. This pivotal component scrutinizes incoming data, identifying alterations relative to the reference data. Only the modified or altered data is permitted to enter fog nodes, thereby preventing the overload of the fog computing infrastructure by excluding redundant or non-essential data. Moreover, when the data is changing frequently then instead of overloading the fog nodes it sends the data to the cloud. For this it uses an M/M/1 queuing model for deciding when to use fog and when to use cloud for giving the output. Hence both cloud and fog are saved at the same time from getting unnecessary overload and division of labor concept works best here.

(D) Fog Computing and Real-Time Processing:

The processed and filtered data resides within the fog nodes, which maintain their storage buffer. Following computation, the output is transmitted promptly to IoT devices, ensuring real- time data processing and feedback. This seamless mechanism ensures the delivery of **"Knowledge"** - timely, actionable insights.

(E) Cloud Integration and In-Depth Analysis:

As data accumulates within the fog's storage buffer and reaches a predetermined capacity, it is automatically forwarded to the cloud for further processing and storage is emptied (DELETED) from the fog. Here, data is processed and stored in larger units, offering the potential for comprehensive and advanced insights, referred to as **"WISDOM."** Users can access this data after a specific duration (weekly, monthly etc.), thereby gaining a deeper understanding of the information and its implications.

(F) Architectural Benefits:

Our architecture provides several notable advantages:

- Efficient Data Handling: By filtering redundant and non-essential data at the edge, our architecture conserves network resources and computational capacity.

- Real-Time Insights: Users receive immediate feedback and insights from the fog nodes, empowering them to make informed decisions promptly.

- In-Depth Analysis: The cloud's processing capabilities enable advanced data analysis, providing users with a profound understanding of the data over time.

- Scalability: Our architecture accommodates an expanding array of IoT devices while maintaining operational efficiency, rendering it highly scalable.

In order to enhance Quality of Service (QoS) for users, integrating fog nodes between the IoT and Cloud environment is essential. However, relying solely on fog computing has limitations due to constraints in computational power and storage capacity. We believe that if fog nodes exclusively handle all tasks, it essentially replicates a cloud-based infrastructure. Therefore, to optimize QoS effectively, our approach focuses on maximizing the efficiency of fog computing. By doing so, we aim to achieve optimal performance from both fog and cloud environments without overburdening them. This optimization strategy not only improves QoS but also conserves energy by balancing workload distribution intelligently between fog and cloud resources.

We have examined two scenarios regarding data generated by IoT/edge devices.

Case 1: The first scenario involves data that exhibits infrequent alterations, such as a temperature sensor in a room. Here, the data remains relatively stable most of the time, with occasional minor variations or periods of constancy. To handle such data efficiently, we've proposed an XOR-based data encoding algorithm. This algorithm operates by transmitting only the modified data, adjusting it with the reference data initially passed at t=0, thereby optimizing data transfer and storage.

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Case 2: In the second scenario, we consider data that rapidly fluctuates within shorter time intervals. For instance, when the data changes frequently in a short span. In response to this, we've introduced an M/M/1 queuing model. This model serves as a decision-making tool, determining whether to route the data directly to the cloud or through the fog infrastructure based on the data input. If the data exhibits rapid variations, indicating quick changes, the model directs this data to the cloud, thereby alleviating the burden on the fog infrastructure and ensuring efficient handling of dynamic data streams

The following equations are used for the modelling:

SYSTEM PARAMETERS

1. Arrival_rate : No of process arriving per second
2. Service Rate, μ = 1/ Service Time
3. Service time , S= Number of Total tasks / Processing Time.
4. Utilization, ρ = Arrival Time / Service rate
5. Average Waiting Time , W= $\rho / (\mu - \lambda)$.
6. Queue Length, L= utilization * average waiting

   time L = $\rho * W$

7. Maximum queue length threshold. : Maximum tasks that can be served in the queue

Processing Time: Can be found by measuring the time it takes for a fog node to process a set of tasks.

TASK PARAMETERS:

1. Cost: The amount of money required to acquire or maintain a product or service.
2. Communication Band: The range or capacity of frequencies that can be used for transmitting data.

3. Baud Rate: The speed at which data is transmitted over a communication channel, typically measured in symbols per second.
4. Signal Level: The strength or intensity of a signal, indicating how well it can be detected or received.
5. Latency: The time delay between the initiation and the actual occurrence of an event, often used in the context of data transmission to indicate delay.

Weighted Score = w1 × Cost + w2 × Communication Band + w3 × Baud Rate + w4 × Signal Level - w5 × Latency

5. Proposed Algorithm

This section comprises two segments addressing distinct data scenarios. The initial part details the implementation of the delta encoding algorithm utilizing an XOR-based approach. This algorithm is tailored for situations where the input data exhibits minimal variability or remains relatively constant for extended periods. It focuses on efficiently transmitting only the altered data by employing a reference point established at the onset (t=0), optimizing data transfer and storage in instances of limited data variability.

The latter segment discusses the integration of the M/M/1 queuing model. This model serves a pivotal role in determining the optimal routing of data streams towards the cloud infrastructure. Specifically designed for scenarios where data variability occurs rapidly and frequently, this queuing model dynamically evaluates the data input. When rapid variations are detected, signifying swift changes in the data, the model efficiently directs this dynamic data flow directly to the cloud. This strategic routing mechanism aims to alleviate strain on the fog infrastructure while ensuring timely and effective handling of swiftly changing data.

5.1 XOR based delta encoding algorithm

The algorithm operates through two primary phases: initialization at t = 0 and the main algorithm at t > 0. During the initialization phase, we establish two crucial components: the reference dataset (D_reference) and the new data packet (D_new), both meticulously represented as arrays of binary data. Subsequently, in the main algorithm phase, we employ the XOR operation to diligently calculate the delta (D_delta) between the new data packet and the reference dataset.

A fundamental checkpoint in the algorithm pertains to the determination of the non-empty status of D_delta. This evaluation is a pivotal step that effectively distinguishes whether changes have occurred in comparison to the reference data. If D_delta is found to be dissimilar to an array of all zeros, it indicates the presence of data alterations, which necessitates immediate action. In such cases, the reference dataset is swiftly updated to reflect the new data, facilitating a state-of-the-art mechanism for data synchronization. Concurrently, the data delta, in its modified state, is dispatched to the fog node for further computational processes, ensuring data integrity and accuracy.

Start

- Initialization at t = 0:

```
  - D_reference = [0, 0, 0, ..., 0] (An array of zeros representing the initial reference dataset)
  - D_new = [A1, A2, A3, ..., An] (An array representing the new data packet, where Ai is a binary digit)

- Main Algorithm at t > 0:
  1) Calculate the delta:
    - D_delta = XOR (D_new, D_reference)

  2) Check if D_delta is non-empty:
      - If D_delta ≠ [0, 0, 0, ..., 0] (An array of zeros):
-         Update the reference dataset:
        - D_reference = D_new

   3) Send D_delta to the fog node for computations.
end
```

5.2 M/M/1 Queuing model with dynamic adjustment.

Algorithm Steps:
Algorithm consists of 5 stages. Stage 1 includes calculating utilization, average waiting time and task parameters.
Second stage includes Checking Data complexity, if data is found to be complex, then offload it to the cloud, else step forward for local processing.
Third includes calculating Weighted score by considering the task parameters of the process that includes:
Cost : Higher cost is more preferable, , so positive weight. (w1 x Cost)
Communication Band: Higher bandwidth is generally preferable, , so positive weight (+w2 x Communication Band)
Baud Rate: Higher baud rate is generally preferable, so positive weight. (+w3 × Baud Rate)
Signal Level: Higher signal level is generally preferable, so positive weight (+w4 × Signal Level)
Latency: Lower latency is generally preferable, so negative weight (-w5 × Latency)
If Calculated score is Greater than Threshold Score, then proceed for local processing else offload to cloud.
Stage 4 includes Finding Queue Length, if queue length is less than Threshold Queue length && , then this task can be added to the queue for local processing, else offload it to cloud.
Then final stage consists of Dynamic adjustment which includes recalculating system Parameters i.e. Utilization & Average Waiting Time.

ALGORITHMS

MAIN ALGORITHM FOR CLOUD FOG OFFLOADING procedure

```
CLOUD_FOG_OFFLOADING_ALGO(CFOA)
    Calculate Utilization ➡️ ρ
    Average Waiting Time ➡️ Wq
    Calculate Task Personalized Parameters
        Task_Param = [CT, CmBW, BR, SGL, L]
    Check Data Complexity
        if Data is complex then Offload to cloud
        else Calculate Weighted_Score (Task_Param)
            if Weighted Score > Threshold Score then
                Check Queue Length and update queue
                for each task in Queue
                    if task is completed then
                        Remove it from queue
                        Update
                        Queue_Length
                        If Queue_Length>0 then
                        Queue_Length--
                    if Queue_Length < Threshold_Queue_Length && Average
                    waiting time < Threshold waiting time then
                        Add the Process to the queue.
                        Update Queue_Length
                            Queue_Length++
                        Process Locally on Fog
                    else Offload to cloud
                end for
            else Offload to Cloud
    Recalculate Utilization
    Recalculate Waiting Time
    Back to Procedure
end procedure
```

SUBROUTINE FUNCTIONS

```
procedure WEIGHTED_SCORE( array Task_Param)
    Assign the weight to each parameters (w1, w2, w3, w4, w5)
    Calculate Weighted Score
    $WSc = w1×CT + w2×CmBW + w3×BR + w4×SGL - w5×L$
    RETURN WSc
end procedure
```

```
procedure DATA_COMPLEXITY
    Check Data type ➡️ dTyp
        If dTyp includes [ videos, bulk images, audio, bulk unstructured data]
            RETURN Complex
        else RETURN Not Complex
end procedure
```

5.3 Proposed Cloud Framework for processing the task

Figure shows the primary components of the proposed architecture for the microservices application in the cloud environment. The proposed framework is divided among the following components.

5.3.1 End User Devices: It provides a user interface for cloud users to send the microservices application through various end users’ devices.

5.3.2 Workload analyzer: Clustering and a ranking system are two modules of the workload analyzer. The clustering component divides end-user jobs into clusters based on their resource requirements. It is used to find tasks with comparable resource requirements. On the other hand, a ranking methodology provides a numerical weight to every task, known as rank. Its QoS requirements determine a task's rank. Tasks are ranked to determine their priority so that resources can be allocated appropriately.

5.3.3 Resource pool: Multiple firms provide cloud computing services, which might be diverse and geographically dispersed. The resource pool connects resource providers who are spread across the globe. It also aids in centralized resource supply and administration by storing resource details such as resource type, amount of resources, geographical position, etc.

5.3.4 Monitoring: It collects consumption statistics for every VM from the appropriate PM's hypervisor. Efficient monitoring aids in making judgments on how to increase the system's behavior, energy efficiency, and other attributes. A monitoring agent operates in a PM's hypervisor and gathers QoS measurements regularly, which is kept in a QoS metric database for later analysis. This article employed low-level measurements like CPU, memory, and network bandwidth utilization to avoid the substantial cost.

5.3.5 Resource scheduling: A resource schedule is a list of activities and resources on a timeline. In other terms, it evaluates when a task should begin or end based on the work's length, QoS requirements, and available resources. At certain times, shared resources are available, and activities are scheduled during those times.

5.3.6 SLA management: The SLA management module keeps track of each user's Service Level Objectives (SLOs) and completion history. A service level agreement (SLA) is a established agreement between the service providers and the users. It includes several service performance measures as well as the SLOs that go with them.

5.3.7 Resource Allocation: The suggested FSWOA, which is a merger of Fine-tuned Sunflower Optimization (FSFO) and Whale Optimization Algorithms (WOA), adopts both algorithms' parametric properties to improve the resources allocation method's efficiency. By evaluating the minimum fitness function, the suggested approach efficiently assigns the work to a Virtual machine. To ensure efficient resource allocation, the suggested FSWOA employs the humpback whales hunting technique and foraging behavior, as well as the unusual behavior of sunflowers.

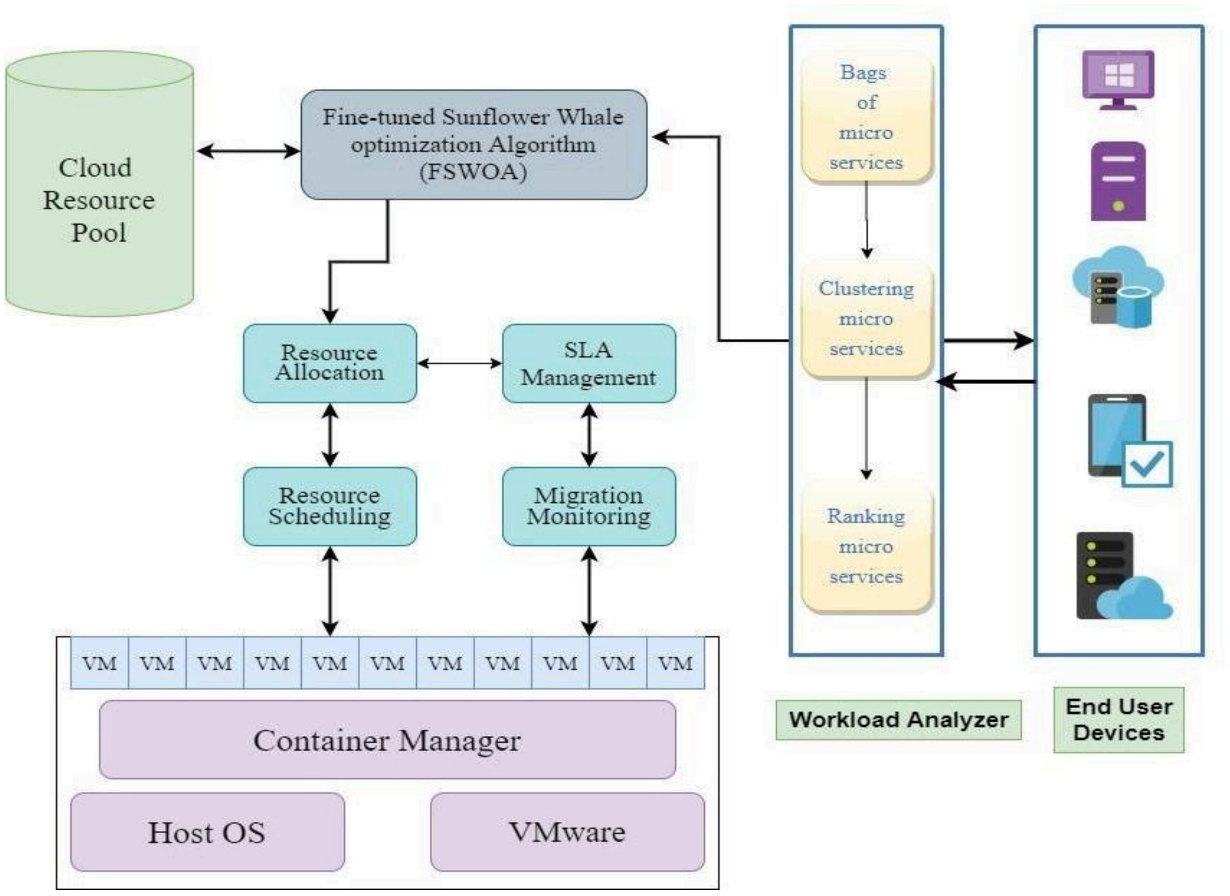


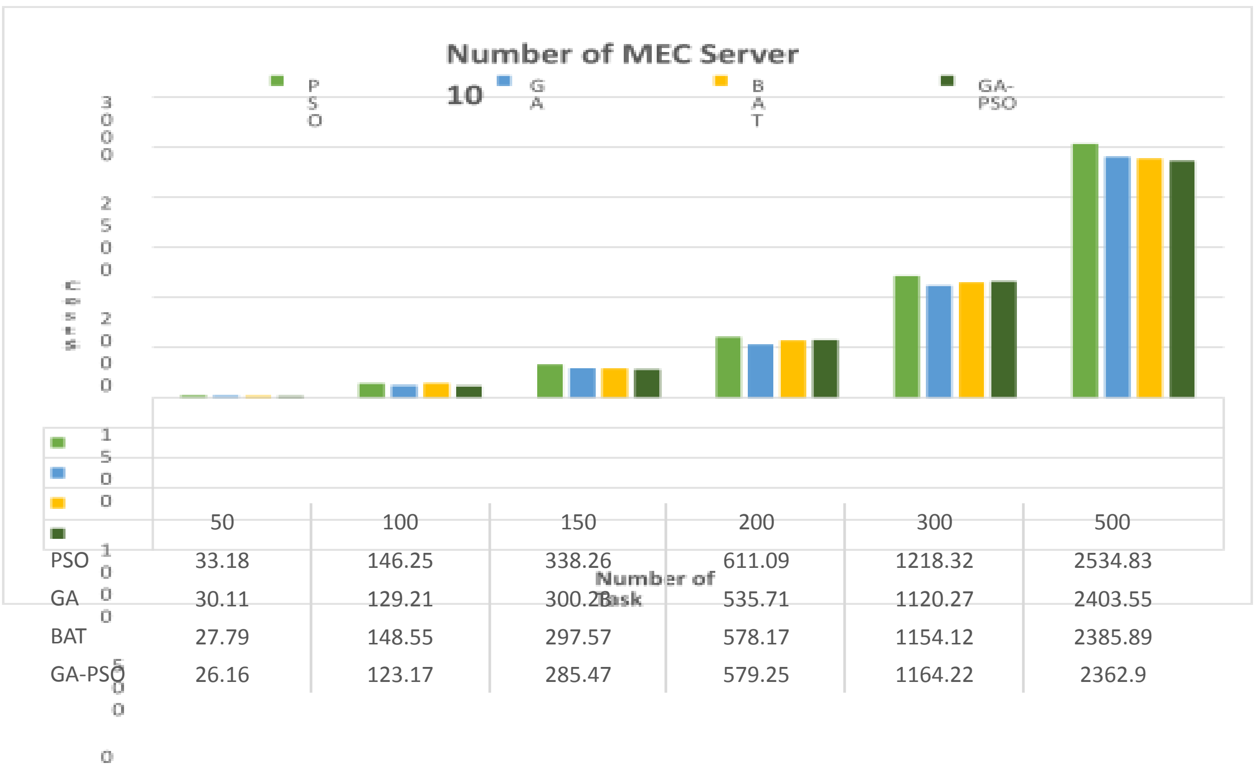
Figure 2 Proposed Framework for microservices in cloud environment

6. Simulation and Result Analysis

To validate the performance of the proposed framework, extensive simulations are performed that are based on the collection of distributed fog nodes and a centralized cloud computing model. The communication time and cost between the fog nodes and the cloud are the main parameters for this research work. Furthermore, the last job completion time is also known as makespan, and the quantity of the data transferred is also considered for evaluation. The above parameters analysis aims to show the effectiveness of the proposed algorithm

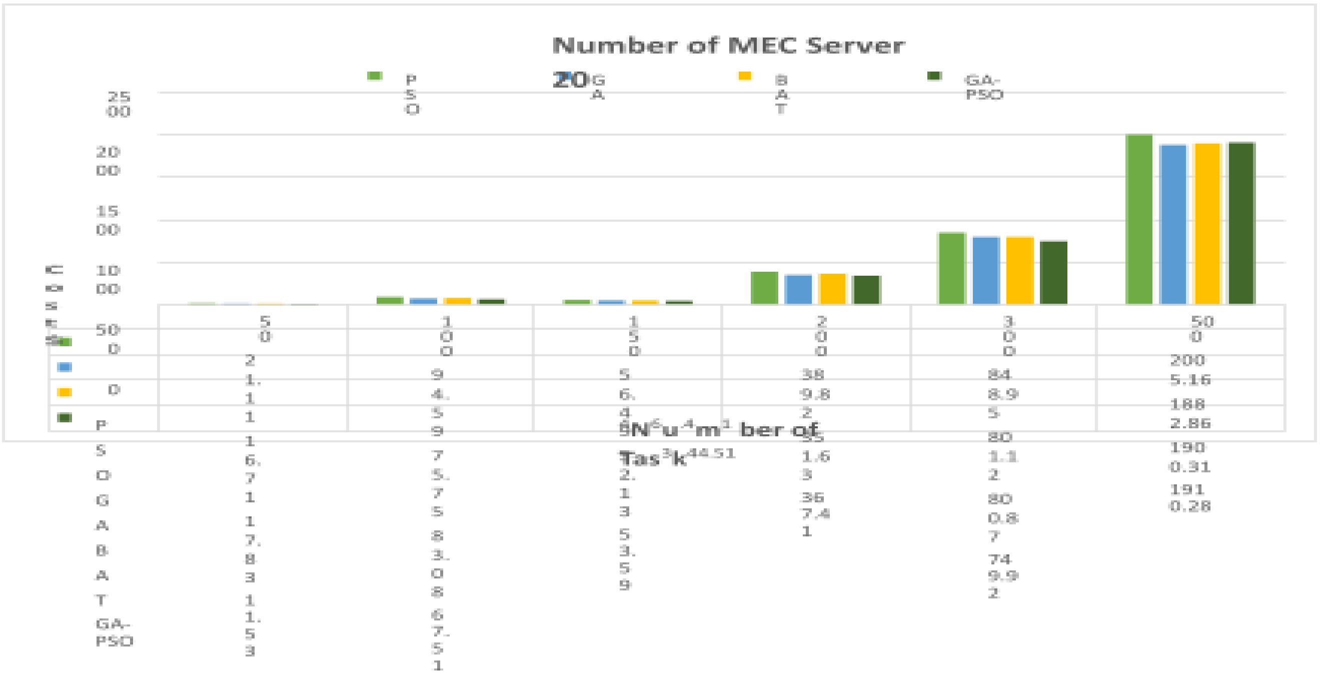
1. Total Computation Cost

Cost is one of the most significant parameters for evaluating the performance of proposed research work in terms of cloud service providers and cloud users. The total cost can be defined as the amount a cloud user pays to the cloud services provider for using the services of the cloud model. The presented research work considers the Amazon Elastic Compute Cloud (EC2) model for calculating the cost parameter. The evaluate the performance of the proposed algorithm for total cost parameters, we performed the four simulations set by varying the number of MEC servers from 10, 20, 30, and 40 against the task sets 50, 100, 150, 200, 300, and 500. In the first simulation set, the number of MEC servers is fixed to 10, and varies the task from 50 to 500. Figure 1 shows the comparative analysis of three states of art algorithms: PSO, GA, and BAT with the proposed GA- PSO algorithm. The proposed algorithm has the least cost compared to the other three algorithms. The proposed algorithm reduces the overall cost by an average of 5.2%, 7.8 %, and 12.5% to BAT, GA, and PSO respectively. In order to test the consistency of our proposed algorithm, we checked the performance of our algorithm with 20 MEC servers with the same task set. Figure 2 demonstrates the comparative result analysis of total computation cost with the varying number of tasks. It is concluded from the result that the proposed algorithm reduces the total computation cost by an average of 3.6%, 6.2%, 7.1%, and 11.9% for BAT, GA, and PSO respectively. Similar to MEC servers 10 and 20, we test over experiments for 30 and 40 MEC servers with the same number of tasks set to check the upper bound of our proposed algorithm. Figures 3 and 4 demonstrate the total computation cost comparative analysis with the 30 MEC servers and 40 MEC servers respectively. The outcome of all four simulation experiments confirms that the proposed algorithm better than the other three state of art algorithms.



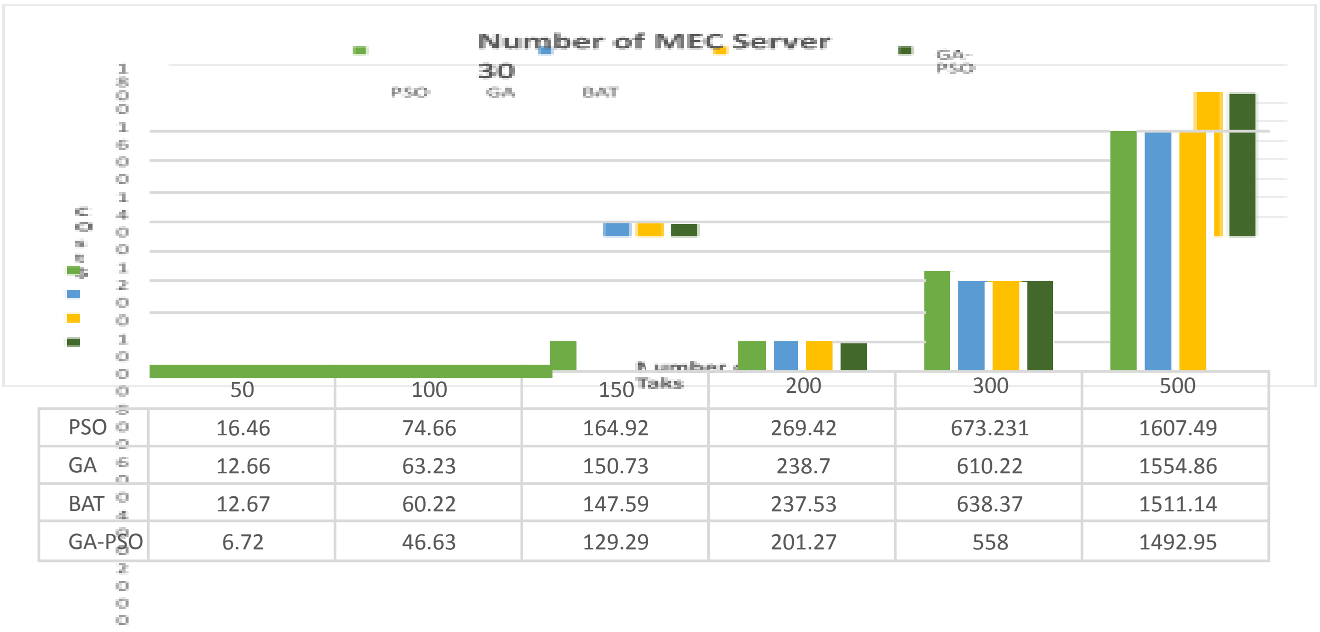
	50	100	150	200	300	500
🟩 PSO	33.18	146.25	338.26	611.09	1218.32	2534.83
🟦 GA	30.11	129.21	300.28	535.71	1120.27	2403.55
🟨 BAT	27.79	148.55	297.57	578.17	1154.12	2385.89
🟩 GA-PSO	26.16	123.17	285.47	579.25	1164.22	2362.9

Figure 1: Total computation cost vs number of task comparison for MEC Server 10



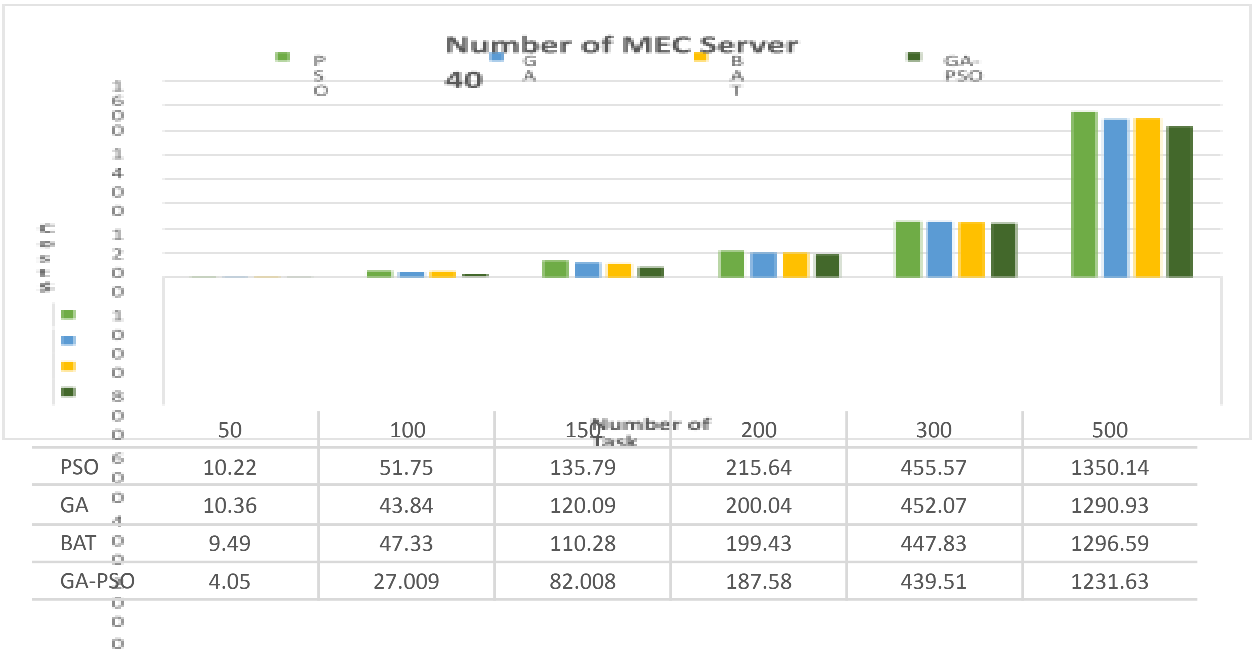
	50	100	150	200	300	500
🟩 PSO	21.11	94.59	56.45	389.82	848.95	2005.16
🟦 GA	16.71	75.75	44.51	351.63	801.12	1882.86
🟨 BAT	17.83	83.08	52.13	367.41	800.87	1900.31
🟩 GA-PSO	11.53	67.51	53.59	[illegible]	749.92	1910.28

Figure 2: Total computation cost vs number of task comparison for MEC Server 20



	50	100	150	200	300	500
PSO	16.46	74.66	164.92	269.42	673.231	1607.49
GA	12.66	63.23	150.73	238.7	610.22	1554.86
BAT	12.67	60.22	147.59	237.53	638.37	1511.14
GA-PSO	6.72	46.63	129.29	201.27	558	1492.95

Figure 3: Total computation cost vs number of task comparison for MEC Server 30



	50	100	150	200	300	500
PSO	10.22	51.75	135.79	215.64	455.57	1350.14
GA	10.36	43.84	120.09	200.04	452.07	1290.93
BAT	9.49	47.33	110.28	199.43	447.83	1296.59
GA-PSO	4.05	27.009	82.008	187.58	439.51	1231.63

Figure 4: Total computation cost vs number of task comparison for MEC Server 40

2. Energy Consumption

In the initial stage of simulation, we plan to analysis the energy consumption parameters with all tasks on different virtual machines based on a series of tasks. However, with such experiments, we did not reach to the upper bound of energy consumption parameters. As shown in Figure 5, we observed that the proposed algorithm saves energy with an average of 9.4 KW/H, 5.6 KW/H, and 13.4 KW/H compared to BAT, GA, and PSO respectively. One more interesting observation we found with the comparison results that with the increment of number, the energy saving amount is also increases. To check the energy consumption with higher MEC servers, we performed the simulation experiments on 20, 30, and 40. Figure 6 demonstrates the result of energy consumption with the same set of tasks against the 20 MEC servers. The comparative result shows that the proposed algorithm reduces the energy consumption rate with an average of 8.9 KW/H, 6.1 KW/H, and 12.5 KW/H compared to BAT, GA, and PSO respectively. The similar result we found in the Figures 7, and 8 for the MEC servers 30 and 40 respectively.

Figure 6 : Energy consumption vs number of t

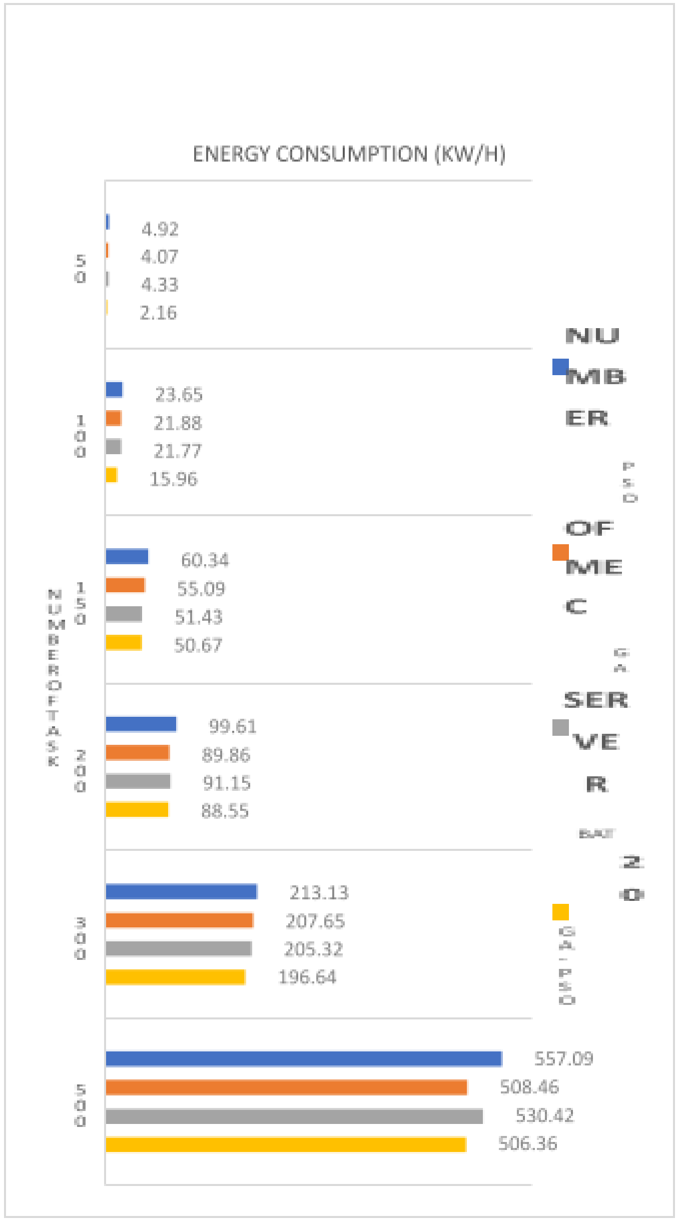
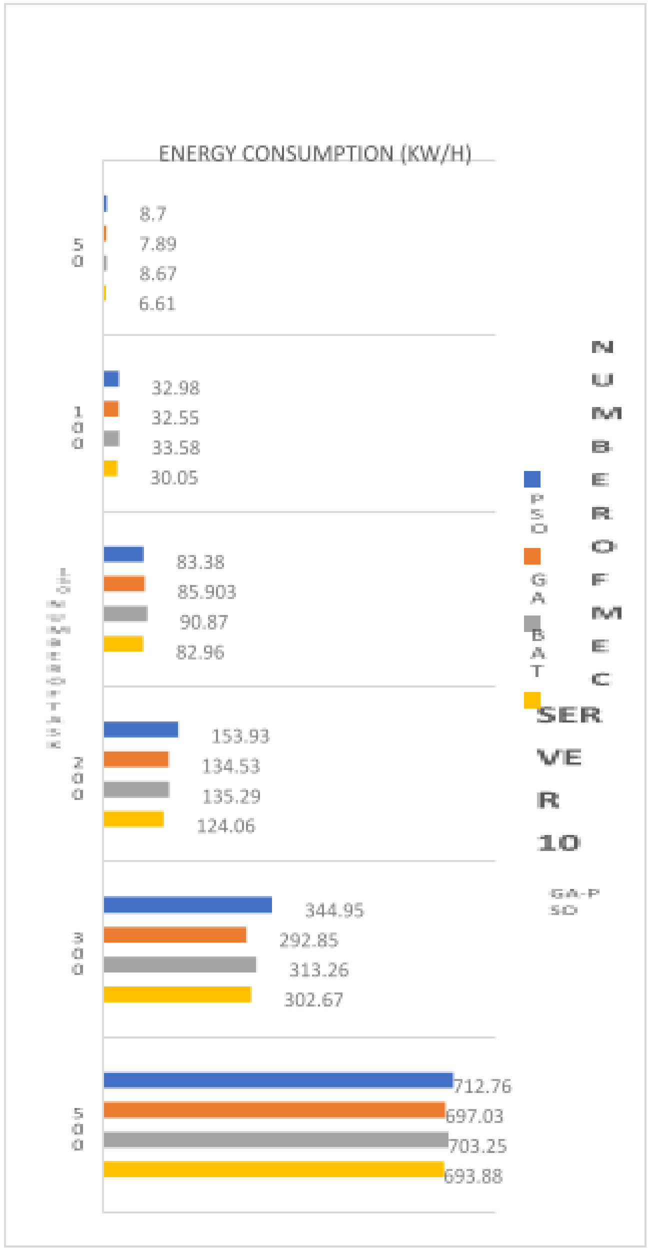


Figure 5 : Energy consumption vs number of t



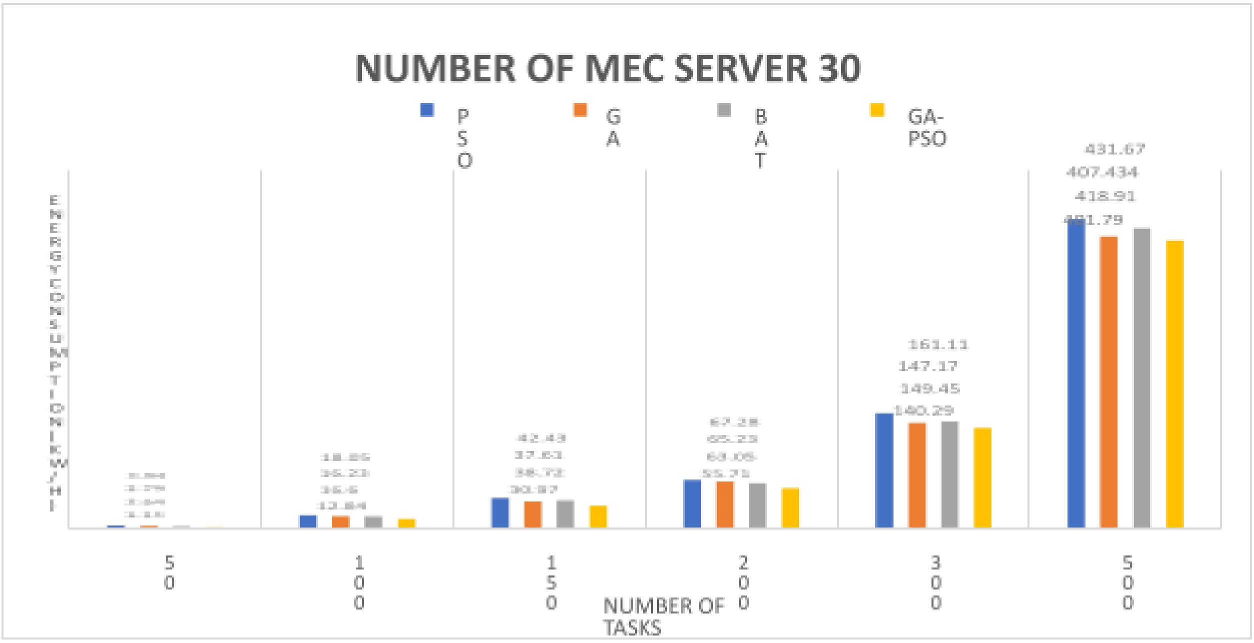


Figure 7: Energy consumption vs number of task comparison result for MEC Server 30

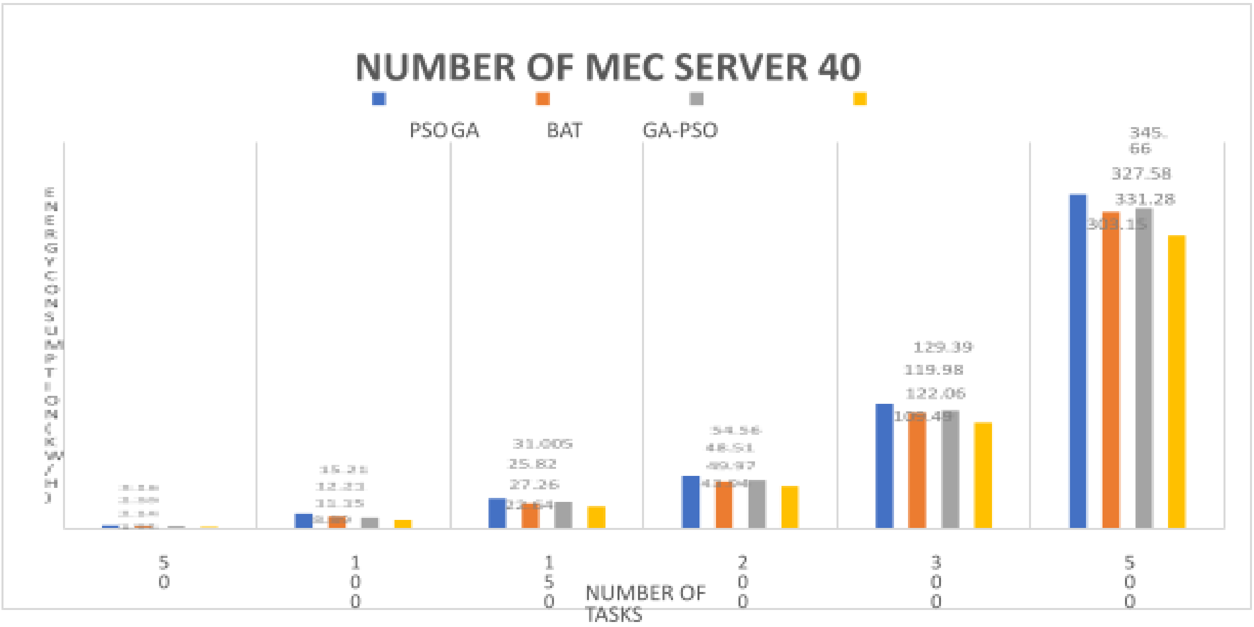
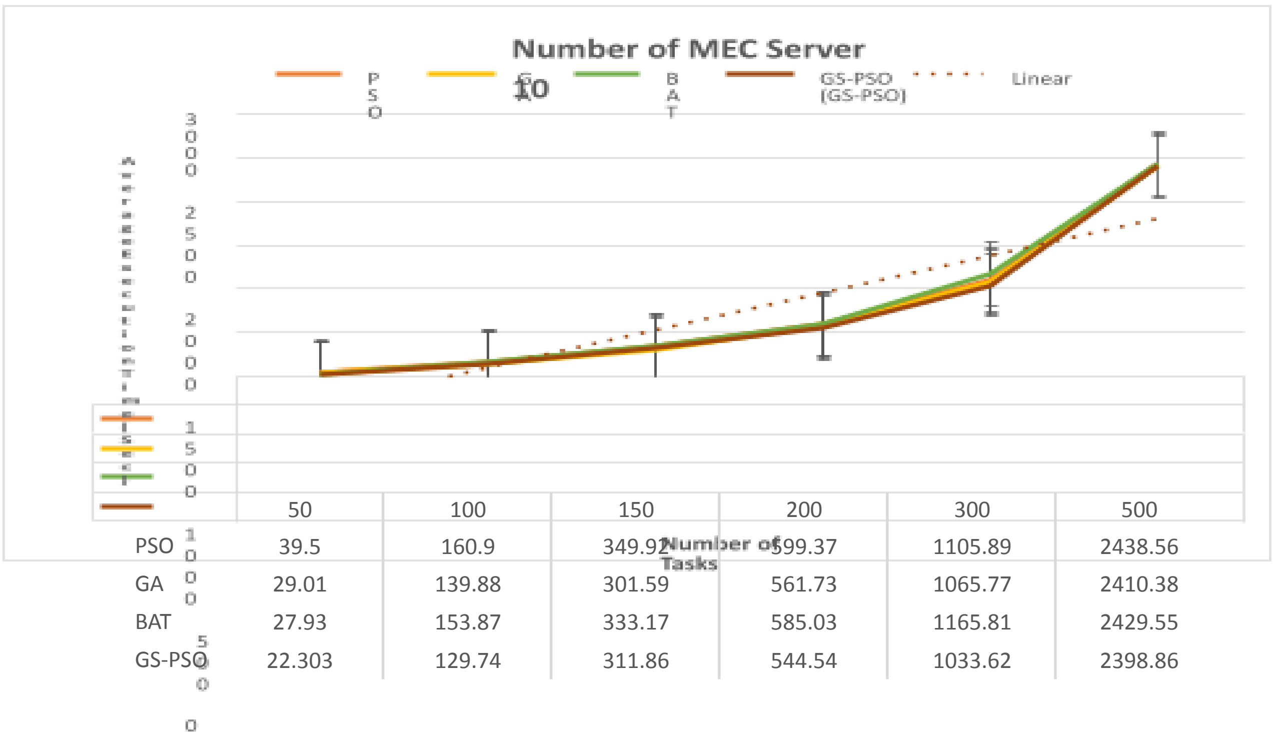


Figure 8: Energy consumption vs number of task comparison result for MEC Server 40

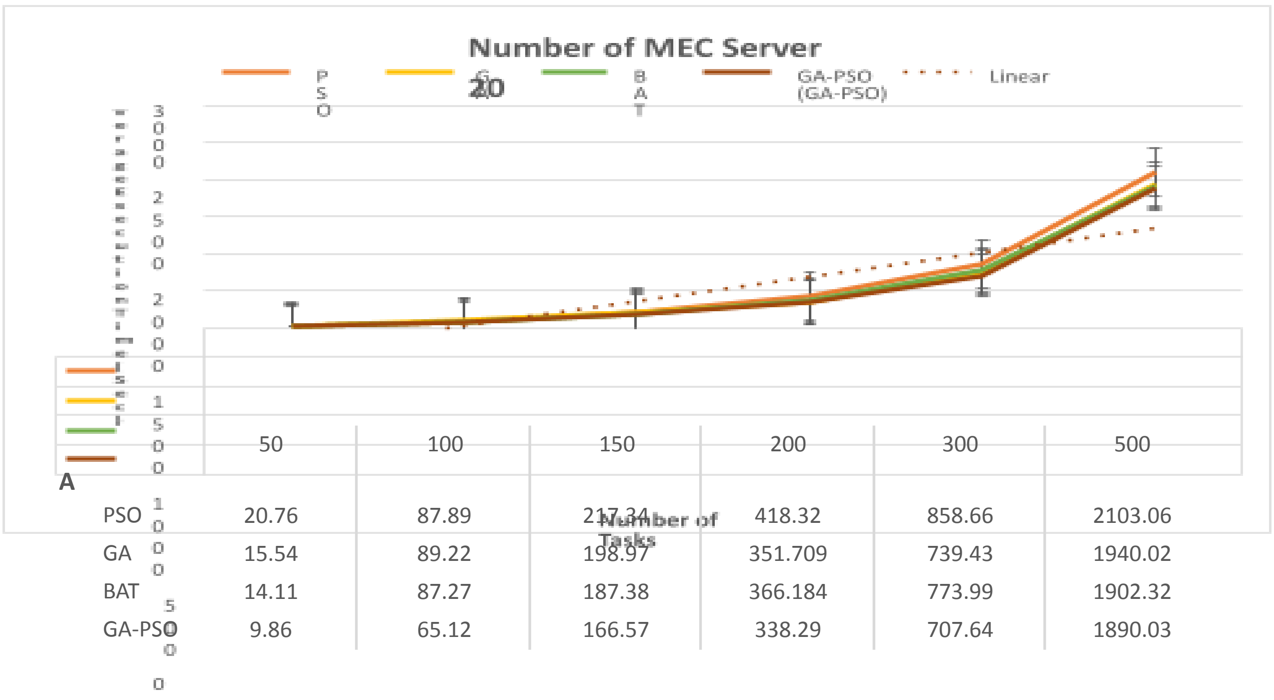
3. Average Processing Time

Average Processing Time parameters play a very significant role in making a strong analysis for the proposed algorithm against the other three state of art algorithms. Figure 9-12 shown the average processing time parameter comparative analysis of the proposed algorithm with BAT, GA, and PSO with the 10, 20, 30, and 40 MEC servers respectively. The simulation result proves that the our proposed algorithm performs significantly better than the other three algorithms.



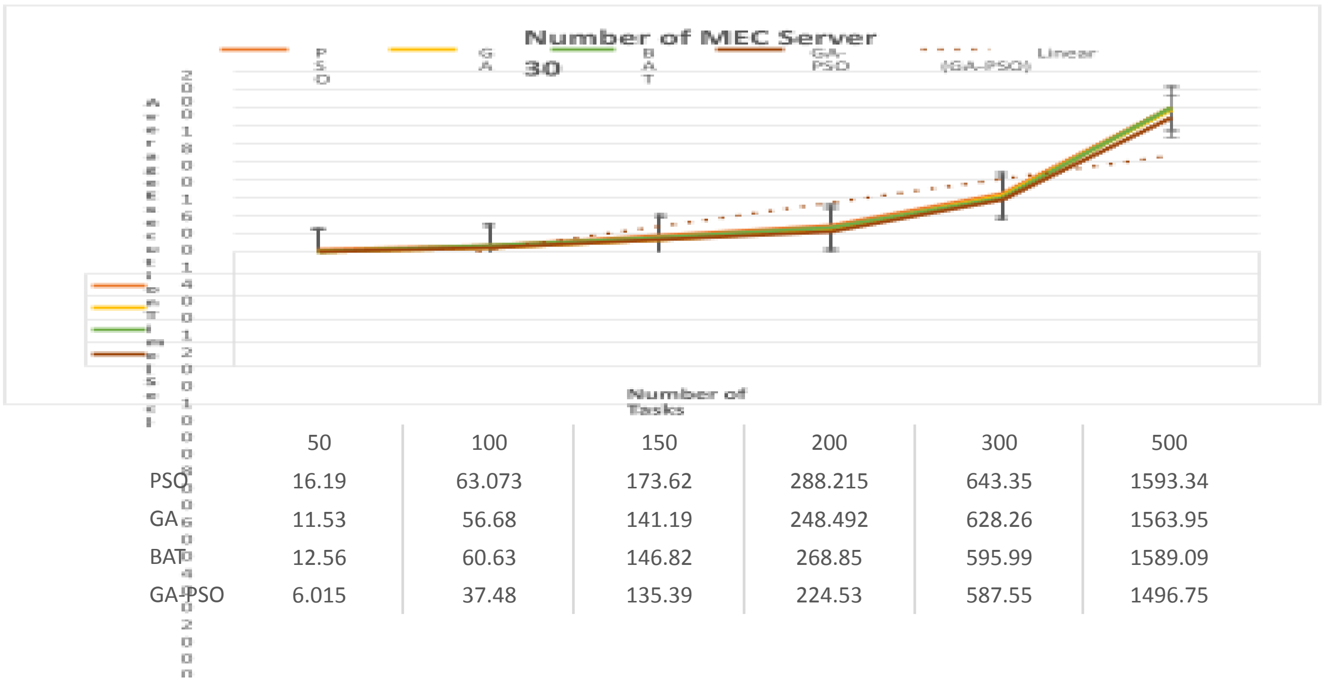
	50	100	150	200	300	500
PSO	39.5	160.9	349.92	599.37	1105.89	2438.56
GA	29.01	139.88	301.59	561.73	1065.77	2410.38
BAT	27.93	153.87	333.17	585.03	1165.81	2429.55
GS-PSO	22.303	129.74	311.86	544.54	1033.62	2398.86

Figure 9: Average processing time vs number of task comparison result for MEC Server 10



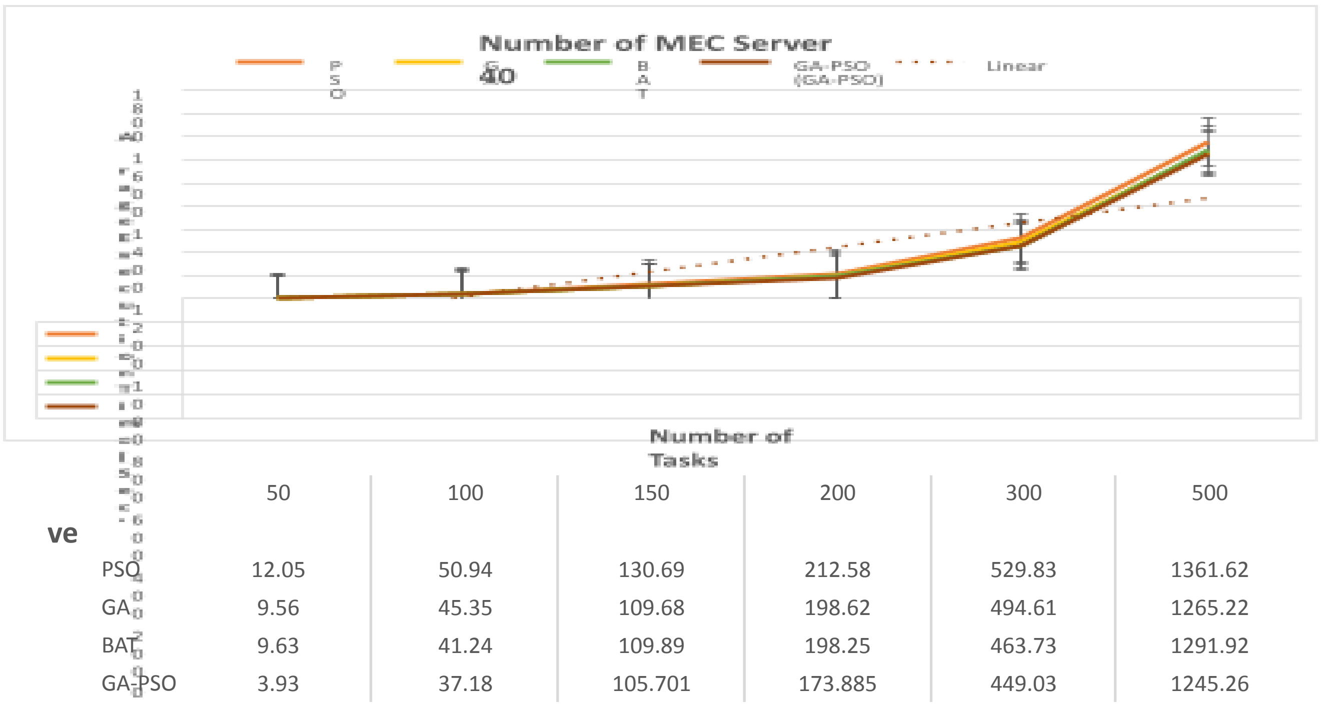
	50	100	150	200	300	500
PSO	20.76	87.89	217.34	418.32	858.66	2103.06
GA	15.54	89.22	198.97	351.709	739.43	1940.02
BAT	14.11	87.27	187.38	366.184	773.99	1902.32
GA-PSO	9.86	65.12	166.57	338.29	707.64	1890.03

Figure 10: Average processing time vs number of task comparison result for MEC Server 20



	50	100	150	200	300	500
PSO	16.19	63.073	173.62	288.215	643.35	1593.34
GA	11.53	56.68	141.19	248.492	628.26	1563.95
BAT	12.56	60.63	146.82	268.85	595.99	1589.09
GA-PSO	6.015	37.48	135.39	224.53	587.55	1496.75

Figure 11: Average processing time vs number of task comparison result for MEC Server 30



	50	100	150	200	300	500
PSO	12.05	50.94	130.69	212.58	529.83	1361.62
GA	9.56	45.35	109.68	198.62	494.61	1265.22
BAT	9.63	41.24	109.89	198.25	463.73	1291.92
GA-PSO	3.93	37.18	105.701	173.885	449.03	1245.26

Figure 12: Average processing time vs number of task comparison result for MEC Server 40

7. Conclusion and Future Work

In this research, we introduced an innovative data control system and proposed two distinct algorithms aimed at augmenting Quality of Service (QoS) for users within the IoT and cloud computing paradigm. Our contributions lay in mitigating cloud overload while efficiently leveraging cloud resources for comprehensive analysis and handling of variable data streams. The implementation of the XOR-based delta encoding algorithm and the **M/M/1 queuing model** demonstrates the potential to optimize data transmission, balancing the load between fog and cloud resources intelligently.

Looking ahead, our future research endeavours will focus on enhancing the effectiveness and robustness of the XOR-based delta encoding algorithm and the M/M/1 queuing model. We aim to refine these algorithms further, making them more comprehensive and adaptive to a wider range of data variability scenarios. Additionally, we encourage fellow researchers in this field to expand upon this work. Collaborative efforts and new perspectives are essential in refining and extending the capabilities of this data control system within the realm of fog computing. We believe that the integration of this data control system with fog computing has the potential to revolutionize the landscape of cloud computing, offering new dimensions for efficient and dynamic data management.

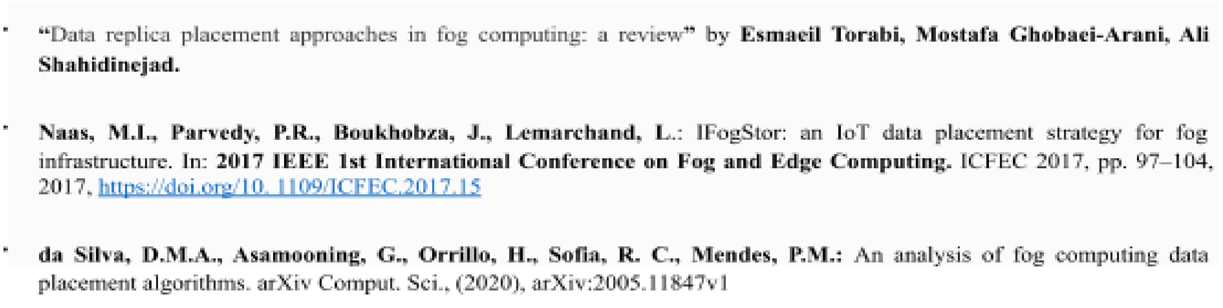
Compliance with ethical standards

Conflict of interest: The authors whose names are given in this article certify that they have no conflict of interest.

References

[1] "Introduction to the IOx Framework" by Todd Baker, Manager of Product Management, IoT Group, Cisco on Thursday, May 22, 11:45 a.m. -12:00 p.m. on the Cisco booth theatre.

[2] "A Hybrid Deduplication for Secure and Efficient Data Outsourcing in Fog Computing" from **2016 IEEE 8th International Conference on Cloud Computing Technology and Science** by **Dongyoung Koo, Youngjoo Shin, Joobeom Yun and Junbeom Hur.**



- "Data replica placement approaches in fog computing: a review" by **Esmaeil Torabi, Mostafa Ghobaei-Arani, Ali Shahidinejad.**
- **Naas, M.I., Parvedy, P.R., Boukhobza, J., Lemarchand, L.:** IFogStor: an IoT data placement strategy for fog infrastructure. In: **2017 IEEE 1st International Conference on Fog and Edge Computing.** ICFEC 2017, pp. 97–104, 2017, https://doi.org/10. 1109/ICFEC.2017.15
- **da Silva, D.M.A., Asamooning, G., Orrillo, H., Sofia, R. C., Mendes, P.M.:** An analysis of fog computing data placement algorithms. arXiv Comput. Sci., (2020), arXiv:2005.11847v1

[6] "A Novel Architecture for Efficient Fog to Cloud Data Management in Smart Cities" from **2017 IEEE 37th**

International Conference on Distributed Computing Systems by **Amir Sinaeepourfard; Jordi Garcia; Xavier**

Masip-Bruin; Eva

Marin-Tordera.

[7] Kleinrock, Leonard (1975). Queueing Systems Volume 1: Theory. p. 77. ISBN 0471491101.

[8] Harrison, Peter; Patel, Naresh M. (1992). Performance Modelling of Communication Networks and Computer Architectures. Addison–Wesley

[9] Guillemin, F.; Boyer, J. (2001). "Analysis of the M/M/1 Queue with Processor Sharing via Spectral Theory" (PDF). Queueing Systems. **39** (4): 377. doi:10.1023/A:1013913827667

[10] Little, J. D. C. (1961). "A Proof for the Queuing Formula: L = λW". Operations Research. 9 (3): 383- 387. doi:10.1287/opre.9.3.383