Original Article
ISSN (Online): 2582-7472

ShodhKosh: Journal of Visual and Performing Arts
May 2024 5(5), 352–361

GENETIC ALGORITHM VS ANT COLONY OPTIMIZATION FOR OFFLOADING IN MOBILE AUGMENTED REALITY

Chandra Shekhar Gautam[1], Dr. Akhilesh A. Waoo[1]

[1] Department of Computer Science, Faculty of Engineering and Technology, AKS University, Satna (M.P.) India

Corresponding Author

Akhilesh A. Waoo,
akhileshwaoo@gmail.com

DOI
10.29121/shodhkosh.v5.i5.2024.1886

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

ABSTRACT

This study presents a comparative analysis of two prominent optimization techniques, Genetic Algorithm (GA) and Ant Colony Optimization (ACO), for offloading tasks in Mobile Augmented Reality (MAR) environments. MAR applications often require intensive computational resources, leading to performance bottlenecks on resource-constrained mobile devices. Offloading tasks to remote servers can alleviate these constraints, but the selection of appropriate offloading strategies is crucial for efficient execution. GA and ACO have been widely employed in optimization problems, yet their effectiveness in the context of MAR offloading remains unexplored. Through experimentation and performance evaluation, this study aims to provide insights into the comparative effectiveness of GA and ACO for MAR offloading scenarios. The findings of this research can inform the selection of suitable optimization techniques to enhance the performance and resource utilization of MAR applications.

Keywords: Genetic Algorithm, Ant Colony Optimization, Mobile Augmented Reality, Offloading, Optimization Techniques

1. INTRODUCTION

Mobile Augmented Reality (MAR) applications have gained significant popularity due to their ability to seamlessly integrate virtual content with the real world, enhancing user experiences across various domains such as gaming, education, navigation, and commerce. However, the resource constraints of mobile devices pose challenges to the efficient execution of MAR applications, particularly in scenarios involving computationally intensive tasks. Offloading, the process of delegating tasks to remote servers, has emerged as a promising approach to alleviate these constraints and enhance the performance of MAR applications [13].

The selection of offloading strategies plays a crucial role in optimizing the performance of MAR applications. Various optimization techniques have been proposed to address this challenge, among which Genetic Algorithm (GA) and Ant Colony Optimization (ACO) have shown promise in diverse optimization problems [1] (Gautam, 2022). Despite their effectiveness in other domains, their applicability and performance in the context of MAR offloading remain relatively unexplored.

This study aims to fill this gap by conducting a comparative analysis of GA and ACO for offloading tasks in MAR environments [16] (Waoo, 2012). By evaluating the performance of these optimization techniques under different scenarios and workloads, this research seeks to provide insights into their effectiveness and suitability for MAR offloading. Understanding the comparative strengths and weaknesses of GA and ACO can inform the selection of optimal offloading strategies, thereby improving the performance and resource utilization of MAR applications.

In this paper, we present a comprehensive overview of the motivations, objectives, and structure of the study. We begin by discussing the background and significance of MAR offloading, followed by a brief review of related work in the field of optimization techniques for offloading. Subsequently, we outline the research objectives, methodology, and experimental setup employed in this study. Finally, we provide an overview of the paper's organization, highlighting the key sections and contributions. Through this investigation, we aim to contribute valuable insights to the research community and practitioners involved in the development and optimization of MAR applications.

2. ALGORITHM IN MAR OFFLOADING

Mobile Augmented Reality (MAR) offloading involves the process of delegating resource-intensive tasks of an augmented reality application from a mobile device to more powerful remote servers or cloud infrastructure. This offloading is done to improve performance, extend battery life, and enable more complex AR experiences on mobile devices.

2.1 Ant Colony Optimization (ACO)

it is a metaheuristic algorithm inspired by the foraging behavior of ants to find the shortest path between their nest and a food source. It was initially proposed by Marco Dorigo in the early 1990s. ACO has been applied to various combinatorial optimization problems, including the traveling salesman problem (TSP), vehicle routing problem (VRP), and job scheduling.

The calculation of ant pheromone deposition on each edge is a crucial aspect of the Ant Colony Optimization (ACO) algorithm. Here's how it's typically done using the formulas described:

Let's denote:
T_{ij}: Pheromone level on edge (i, j)
$T_{i,j}^{k}$: Amount of eromone deposited by ant k on edge (i,j)
The amount of pheromone deposited on each edge (i,j) by each ant k can be calculated using the following formula:

$$\Delta T_{i,j}^{k} = \frac{1}{L_k}$$

Where:

- L_k is the total length of the solution found by ant k.

Typically, L_k represents the objective function value of the solution found by the ant. For example, in the Traveling Salesman Problem (TSP), L_k would be the total distance traveled by the ant.
After all ants have completed their tours, a global pheromone update is performed. The amount of pheromone deposited globally on edge (i, j) is a combination of the pheromone left by all ants that traversed that edge. It can be calculated using a formula like:

$$\Delta T_{ij} = \sum_{k=1}^{\eta} \Delta T_{i^{j\cdot}}^{k}$$

Where:

- n is the total number of ants that traversed the edge (i,j).
- ΔT_{ij} is the total amount of pheromone deposited on edge (i,j) after all ants have completed their tours.

After the global pheromone update, pheromone evaporation is applied to all edges to prevent stagnation and promote exploration. The amount of pheromone evaporated from each edge can be calculated using a formula like:
$T_{ij} \leftarrow (1\text{-}\rho) \bullet T_{ij}$
Where:

- ρ is the pheromone evaporation rate, typically a value between 0 and 1.

These calculations ensure that edges that are part of shorter solutions receive more pheromone deposits, thus biasing future ants to prefer shorter paths.

2.2 Genetic Algorithm

Genetic Algorithms (GAs) are optimization algorithms inspired by the process of natural selection and genetics. They are used to solve optimization and search problems by mimicking the process of natural evolution. During the simulation, the Genetic Algorithm (GA) will analyze the specified metrics, including network connection/speed, distance from the edge server, and tasks awaiting execution. To execute the GA algorithm, a fitness function must be defined and computed. Here, L denotes latency, while P represents power consumption.

The total power consumption (P_{total}) is calculated based on the device's standby energy consumption (P_{base}), the power consumption per unit of data transmitted (P_{tx}), and the total amount of data transmitted (D). In our experiment, we assign a weight of 1.5 (w_1) to the latency component ($L_{network}$) and a weight of 1 (w_2) to the power consumption component (P_{total}), as we deem latency crucial for Mobile Augmented Reality (MAR) applications.

def evaluate_fitness(w1, w2, $L_{network}$), Ptotal):

```
"""
Evaluate the fitness of a solution based on latency and total power consumption.
  Parameters:
    w1 (float): Weight assigned to latency component.
    w2 (float): Weight assigned to power consumption component.
        L_network (float): Latency of data transmission over the network link.
        P_total (float): Total power consumption.
  Returns:
    float: Fitness value.
  """
  fitness = w1 * L_network + w2 * P_total
  return fitness
```

This function takes the weights w_1 and w_2, the latency $L_{network}$, and the total power consumption P_{total} as input parameters. It computes the fitness value by multiplying the latency component $L_{network}$) with the weight 1w1 and the power consumption component (P_{total})with the weight w_2, and then summing them up. Finally, it returns the computed fitness value.

3. EVALUATION AND ANALYSIS

Evaluating the performance of Genetic Algorithms (GA) and Ant Colony Optimization (ACO) for Mobile Augmented Reality (MAR) offloading involves considering several factors such as processing time, throughput, jitter, scheduling latency, and power consumption. Within Mobile Augmented Reality (MAR) systems, multiple essential tasks collaborate to deliver immersive, responsive, and authentic user experiences. These tasks encompass feature extraction, object recognition and tracking, scene reconstruction, virtual object rendering, and sensor data processing [5]. The simulations will take place within a controlled environment, where the various tasks involved in the Mobile Augmented Reality (MAR) process are isolated. These simulations will encompass diverse parameters, including different types and sizes of tasks, as well as varying resource availability at the local edge device, edge servers, and global servers. For the sake of simplicity, we assume that all tasks are eligible for offloading. To ensure the statistical significance of our findings, we conduct a substantial number of simulations. Upon gathering the results, we proceed with a statistical analysis, which entails computing average performance metrics. This methodology enables a comprehensive comparison between the two algorithms, facilitating the identification of the algorithm demonstrating superior average performance across the designated metrics. Our simulations utilize task and server numbers sourced from iFogSim2 [2][3] as a reference.

3.1 Experiments:

The simulations encompass diverse environments to assess how effectively the algorithms optimize solutions across various scenarios. Table 1 depicts four distinct scenarios for reference. Table 1 depicts four distinct scenarios. In Scenario

1, we simulate a real-world setting where users are positioned at a moderate distance from edge servers, which are situated at varying distances. The global server is positioned 1000km away from the local device. This setup allows us to assess the algorithm's performance in a scenario that mirrors typical usage patterns. Servers 0-3 are equipped with queues, while the global server currently has no tasks. Scenario 2 replicates a scenario where a local device is situated within a city, with servers positioned on the outskirts experiencing high network traffic.in Scenario 3, the edge device is situated within a large city, with servers positioned inside the city center and anticipated to endure substantial usage, particularly in urban areas. Despite the servers boasting high processing speeds, their throughput is constrained by elevated network traffic. Scenario 4 involves simulating a rural setting where offloading servers are situated at a considerable distance from the edge device. Similarly, the global server is also positioned far away from the edge device. Collectively, these scenarios and their respective server setups enable the assessment of task performance across various devices in diverse environments.

Table 1: Server Configurations

	Server Configuration 1								
ID	TYPE	D(KM)	SP(W)	PpDU(W)	T(bps)	PS (ops/s)	AD(s)	JF	Task Queue
0	LOCAL	0	2	0.001	-	150	2	0	Predefined Queue
1	EDGE SERVER	50	0	0.001	3.8G	10K	0.005	0.01	Predefined Queue2
2	EDGE SERVER	100	0	0.001	4.0G	15K	0.001	0.015	Predefined Queue3
3	EDGE SERVER	150	0	0.001	4.2G	20K	0.0015	0.02	Predefined Queue4
4	LOCAL	1000	0	0.001	4.5G	50K	0.002	0.025	
	Server Configuration 2								
ID	TYPE	D(KM)	SP(W)	PpDU(W)	T(bps)	PS (ops/s)	AD(s)	JF	Task Queue
0	LOCAL	0	2	0.001	-	150	5	0.01	Predefined Queue2
1	EDGE SERVER	15	0	0.001	1.8G	20K	0.0001	0.02	Predefined Queue
2	EDGE SERVER	35	0	0.001	1.7G	25K	0.0016	0.04	PredefinedQueue3
3	EDGE SERVER	40	0	0.001	2.8G	30K	0.0024	0.06	Predefined Queue
4	LOCAL	120	0	0.001	2.0G	75K	0.0025	0.05	
	Server Configuration 3								
ID	TYPE	D(KM)	SP(W)	PpDU(W)	T(bps)	PS (ops/s)	AD(s)	JF	Task Queue
0	LOCAL	0	2	0.001	-	150	0	0	Predefined Queue3
1	EDGE SERVER	5	0	0.001	1.5G	40K	0.0012	0.03	Predefined Queue3
2	EDGE SERVER	10	0	0.001	1.8G	70K	0.0018	0.045	Predefined Queue
3	EDGE SERVER	10	0	0.001	2.9G	85K	0.0024	0.006	Predefined Queue8
4	LOCAL	70	0	0.001	1.8G	130K	0.0024	0.006	

	Server Configuration 4								
ID	TYPE	D(KM)	SP(W)	PpDU(W)	T(bps)	PS (ops/s)	AD(s)	JF	Task Queue
0	LOCAL	0	2	0.001	-	150	2	0	
1	EDGE SERVER	70	0	0.001	5.0G	45K	0.014	0	Predefined Queue
2	EDGE SERVER	140	0	0.001	5.6G	47K	0.014	0	
3	EDGE SERVER	200	0	0.001	4.5G	50K	0.021	0	
4	LOCAL	1800	0	0.001	2.6G	90K	0.028	0	Predefined Queue2

The simulation encompasses tasks with diverse computational rates and environmental complexities, as outlined in Table 2 (refer to the Appendix).

Table 2: Task Configurations

Task Configuration 1			
ID	Data Size (bytes)	Task Load (ops)	Description
0	23,325,000	1	Image processing
1	6,220,800	0.75	Object recognition and tracking
3	875,000	1.2	Scene reconstruction
5	6,250,000	1.2	Virtual object rendering (many objects)
8	100	0.4	User input and interaction (low user input)
9	375,000	1.1	Sensor data processing
Task Configuration 2			
ID	Data Size (bytes)	Task Load (ops)	Description
0	23,325,000	1	Image processing
1	6,220,800	1.5	Object recognition and tracking
3	1,275,000	2	Scene reconstruction
5	6,250,000	1.2	Virtual object rendering (many objects)
8	100	0.4	User input and interaction (low user input)
9	375,000	1.1	Sensor data processing
Task Configuration 3			
ID	Data Size (bytes)	Task Load (ops)	Description
0	23,325,000	1	Image processing
1	6,220,800	1.5	Object recognition and tracking
3	1,275,000	2	Scene reconstruction (complex scene)
6	2,250,000	0.6	Virtual object rendering (few objects)
4	1000000	0.8	User input and interaction (low user input)
9	375,000	1.1	Sensor data processing
Task Configuration 4			
ID	Data Size (bytes)	Task Load (ops)	Description
0	23,325,000	1	Image processing
1	6,220,800	1.5	Object recognition and tracking (many objects in the scene)
3	1,275,000	2	Scene reconstruction (complex scene)
7	4,250,000	0.9	Virtual object rendering (medium number of objects)
10	705,000	0.6	User input and interaction (medium user input)

9	375,000	1.1	Sensor data processing

The initial task configuration prioritizes executing an application within a scene featuring a small number of objects. It involves rendering a high volume of objects onto the scene without necessitating extensive user input or interaction. In the second task configuration, implemented within a scene containing numerous objects, the objective is to render many objects onto the scene without heavily relying on user input or interaction. The third task, set within a scene featuring a high number of objects, focuses on rendering few objects while involving substantial user interaction and input. The fourth task entails a moderate level of user input and interaction, rendering a moderate number of objects within a scene containing many objects. The final task operates within a scene featuring a minimal number of objects, focusing on rendering few objects and necessitating minimal user input and interaction.

3.2 Assessment:

Server Set 1: Setup 1 and 2 Setups 1 and 2 are constructed using Server Configuration 1 from Table 1 but handle different sets of tasks. Setup 1 utilizes Task Configuration 1, whereas Setup 2 utilizes Task Configuration 5 from Table 2. The results of the typical usage scenario depicted in Figure 1 illustrate that, in general, the GA algorithm outperforms the ACO algorithm in selecting servers that offer the lowest latency. In setup 1, the average latency for the GA algorithm is 12.93 ms, whereas the average latency for the ACO algorithm is 14.16 ms. Likewise, in setup 2, the average latency for the GA algorithm is 11.97ms, compared to 14.20ms for the ACO algorithm (as indicated in the third column of Figure 1). Throughout the 10 iterations, the GA consistently exhibits stable latency performance, showing minimal fluctuations. In contrast, the ACO algorithm experiences two noticeable latency spikes in both setups 1 and 2. The distribution of tasks among servers varies significantly between the two algorithms. ACO primarily offloads tasks to a single server, as evident from the first column in Figure 1. In contrast, GA distributes tasks more evenly across servers 1 through 3, as depicted in the second column of Figure 1. This broader distribution potentially contributes to its lower latency performance.
In each iteration, the total power consumption remains constant for both setups, as indicated in the fourth column of Figure 1. In setup 1, the ACO algorithm consumes 34.45mW of energy, while the GA algorithm utilizes 37.05mW. Conversely, in setup 2, the GA algorithm demonstrates lower power consumption at 33.05mW, while the ACO algorithm still consumes 34.45mW.

Server set 2: setups 3 and 4. Setups 3 and 4 are built up using Server Configuration 2 with Task Configuration 2 and Task Configuration 1, respectively. The results (refer to Figure 2) indicate that the ACO algorithm outperforms the GA in terms of latency on the selected servers. In particular, setup 4 shows 15.81ms for GA and 13.48ms for ACO, whereas setup 3 produces 12.71ms for GA and 16.08ms for ACO. Note that in configuration 3, every time an iteration occurs, the ACO reliably offloads all work to server 3, which produces a constant delay. Similar results are seen in configuration 4, where every task—aside from iteration 7 which results in a delay of 20.47ms is offloaded to server 3. However, GA displays a heterogeneous distribution of work offloading, with jobs allocated to servers 1-4 throughout all configuration iterations 3 and 4. With intervals ranging from 14.98ms to 17.30ms in Setup 3 and from 14.94ms to 16.37ms in Setup 4, this distribution produces a variety of latency values. On the other hand, the ACO algorithm finds the optimal offloading latency more accurately than the GA method. Compared to GA's 16.08ms and 15.81ms, offloading servers display average latency values of 12.71ms and 13.48ms. In setups 3 and 4, the GA shows a greater power consumption of 37.05mW and 37.45mW, respectively (see the subfigures in Figure 2's fourth column).

Server set 3: setups 5 and 6. Using Server Configuration 3 and Task Configurations 3 and 2, respectively, Setups 5 and 6 are constructed. Figure 3 presents the findings. With a few exceptions, the ACO offloading shows a similar result, primarily offloading work to server 3. In configuration 5, the average latency for GA is 13.58ms and the average latency for ACO is 14.16ms with two separate spikes.

In setup 6, the average latency is 15.32ms due to ACO's more consistent result of 13.50ms, while GA shows a more diversified offloading technique that results in a wider latency range with a peak at 17.78 ms and the lowest peak at 14.19ms. The power consumption of the GA and ACO algorithms in setup 5 and setup 6 are 34.45mW and 37.45mW, respectively. The subfigures in the fourth column of Figure 3 show the greater power consumption of the GA method.

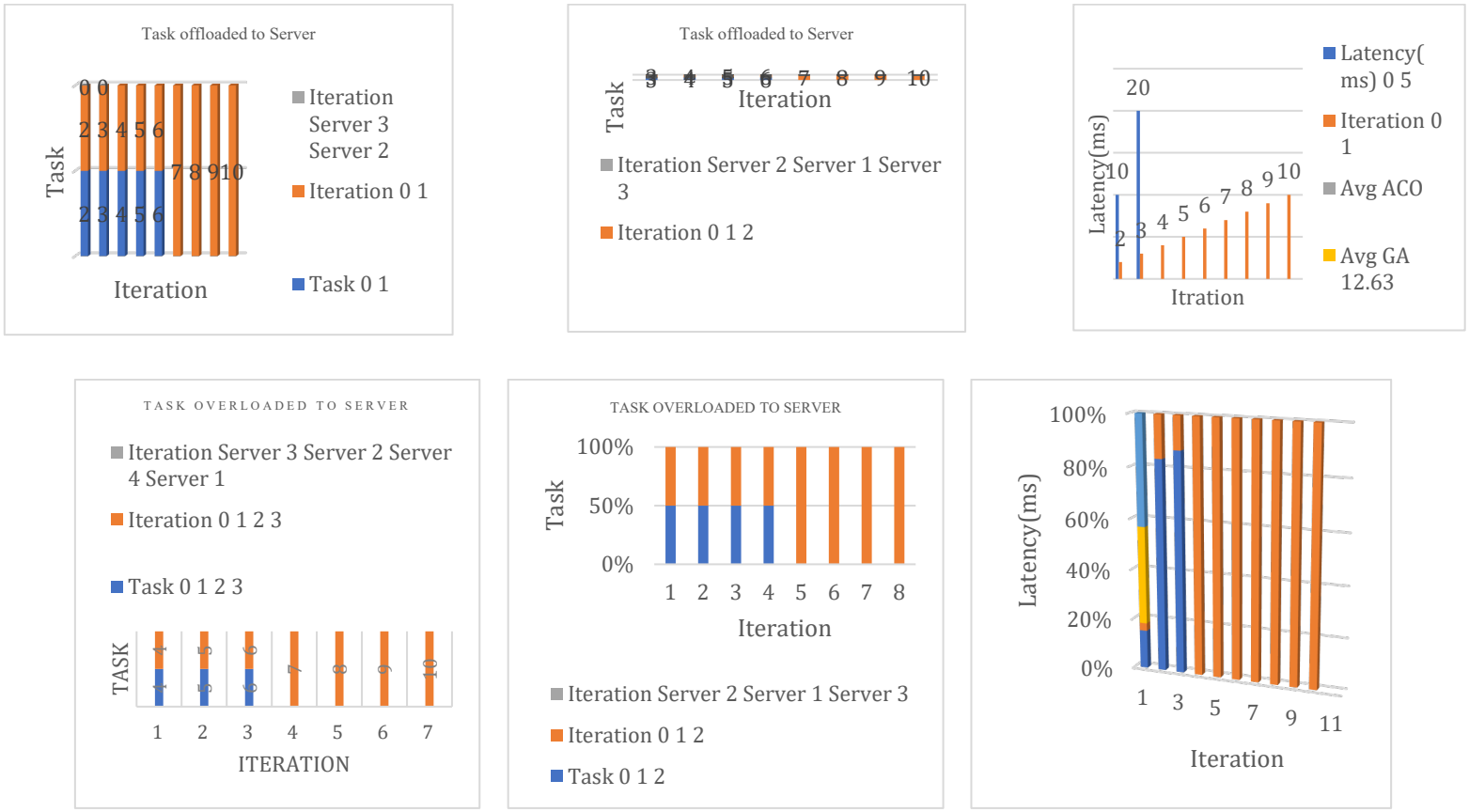
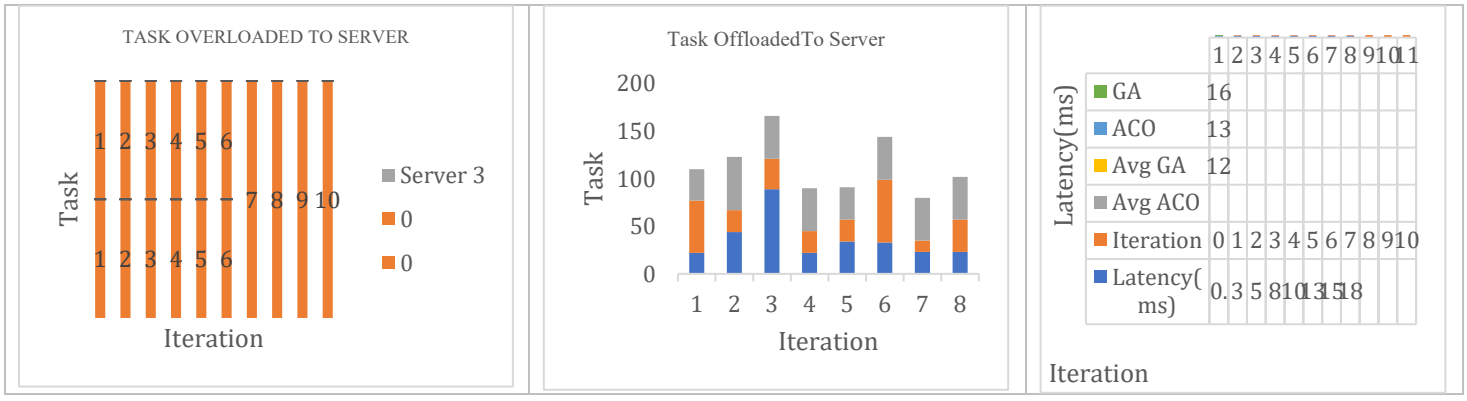


Figure 1 Offloading techniques, delay, and power use in configurations 1 (top) and 2 (bottom). ACO offloading techniques are described in the first column, GA offloading strategies are represented in the second column, and the latency and power consumption of those two algorithms are displayed in the third and fourth columns, respectively.



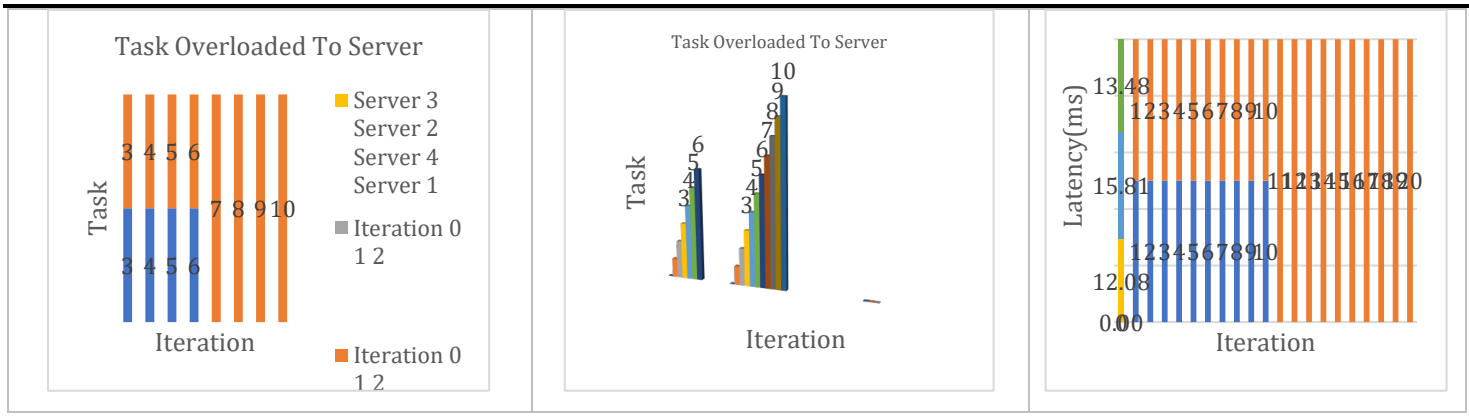
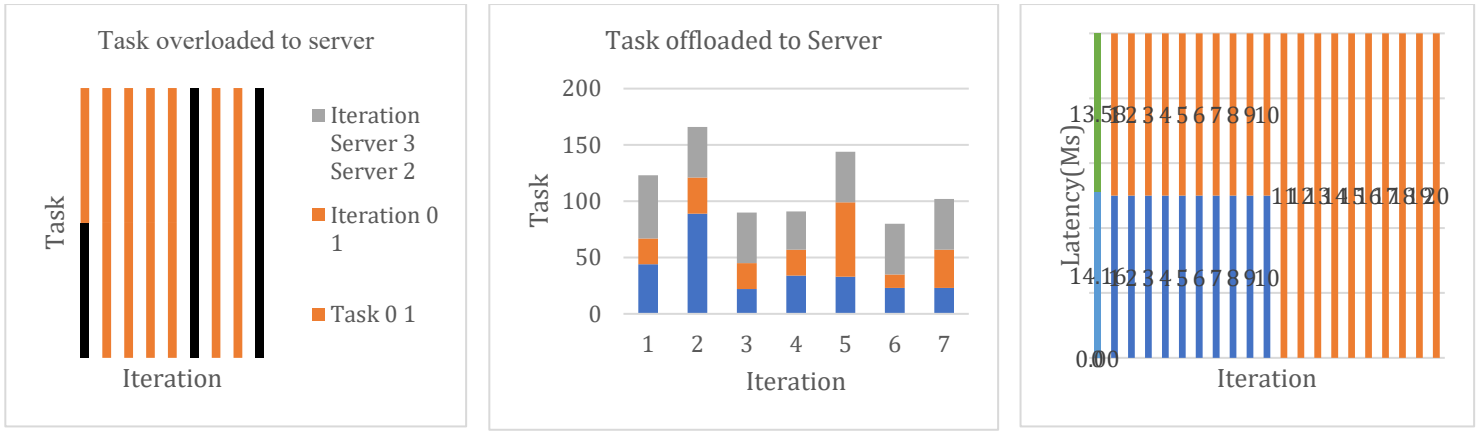


Figure 2: Offloading tactics, latency, and power use in configurations 3 (top) and 4 (bottom). ACO offloading techniques are described in the first column, GA offloading strategies are represented in the second column, and the latency and power consumption of those two algorithms are displayed in the third and fourth columns, respectively.



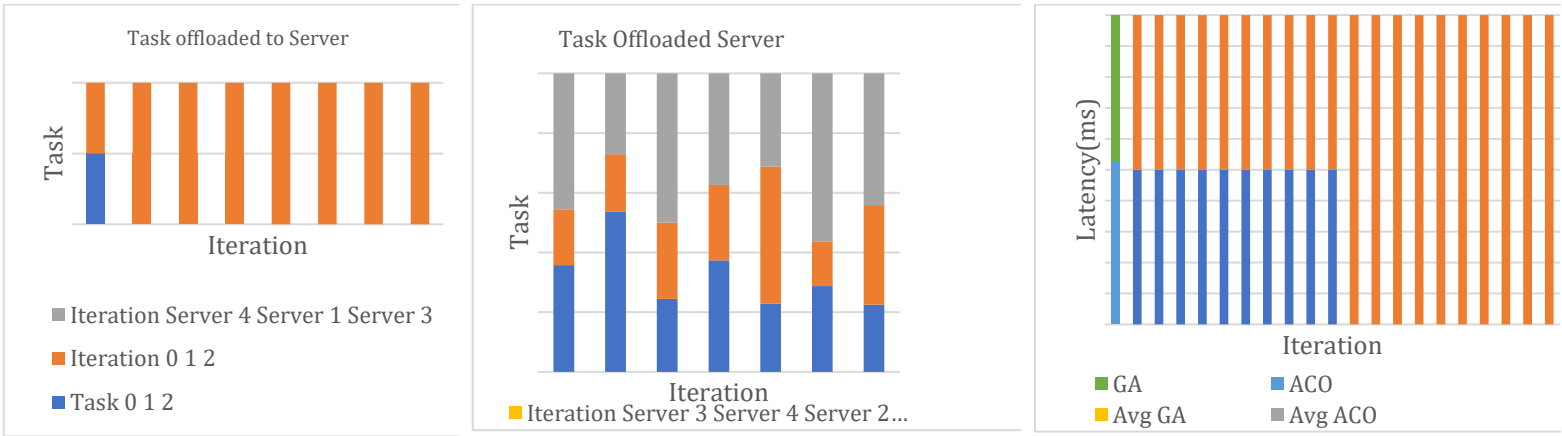


Figure 3: Offloading strategies, latency, and power consumption in setups 5 (above) and 6 (below). The columns have the same representations as Figure 1.

Total Execution Time and Power Consumption Every configuration using the ACO algorithm has a constant total power usage of 34.45mW. The power consumption of the GA, on the other hand, varies substantially, from 33.05mW to 37.45mW. At the moment, just the task offloading delay is taken into account; the algorithm's execution time is not. The ACO algorithm performs better than the GA method in terms of overall delay. The maximum execution time of the ACO algorithm is 25.24ms, but the GA can reach up to 503.19ms in some configurations.

3.3 Carryout

The latency performance research showed that, in configurations 3, 4, 5, and 6 (corresponding Server Configurations: 2 and 3), ACO consistently shows lower latency than GA, but in setups 1 and 2 (corresponding Server Configurations: 1 and 4) GA beats ACO. The particulars of the configurations and the types of jobs given determine how effective both algorithms are. There is a clear pattern in the offloading strategies: the ACO primarily offloaded jobs to one server (usually server 3), which produced steady delay rates. The GA, on the other hand, uses a more varied method of job offloading, dividing up the work across several servers and the local device. Variations in the offloading patterns are the main cause of latency changes in the ACO, especially when jobs are offloaded to servers other than the ones that are used most frequently. With a wider range of offloading techniques, the GA shows fewer notable latency variances. When discussing the overall execution time in a real-world scenario, it's important to keep in mind that the GA algorithm may exhibit up to a 20-fold greater execution time than the ACO method.

4. CONCLUSION AND FUTURE WORK

We assess the efficacy of the GA and ACO algorithms in optimizing task offloading for edge computing in the context of the MAR application. A range of tasks and various server configurations are used to carry out the evaluation. The outcomes showed that both algorithms' success depends significantly on the unique features of the jobs and surroundings. When the servers are situated closer to the edge device, the ACO algorithm continuously exhibits lower latency than the GA. However, in settings where the servers are located farther apart, the GA algorithm performs better than the ACO method. Due to the ACO algorithm's primary duty offloading to a single server, latency rates were consistently maintained. On the other hand, the GA exhibits a more varied approach to task offloading, dividing jobs across many servers and the local device, leading to minor variations in latency. The main cause of latency variations in the ACO algorithm is modifications to the offloading patterns, especially when workloads are offloaded to servers other than the ones that are utilized most frequently. The GA showed fewer notable latency changes due to its more varied offloading strategy. In comparison to the ACO algorithm, the GA exhibits greater power consumption in the majority of settings.

Some of the limitations of this study include the use of a dataset that is not based on realistic numbers, the use of the same hardware to measure the algorithms' execution times, and the use of relatively modest server queues when compared to what is typically encountered in real-world scenarios. Notwithstanding these drawbacks, the study offers insightful information about how well GA and ACO perform in task offloading optimization for edge computing environments, emphasizing how the choice between GA and ACO depends on the particular use case and the intended trade-off between latency and power consumption. Additionally, the research might be expanded by examining other optimization techniques and how well they function in circumstances involving more intricate edge computing.

CONFLICT OF INTERESTS

None

ACKNOWLEDGMENTS

None

REFERENCE

An improving query optimization process in Hadoop MapReduce using ACO-Genetic algorithm and HDFS map reduce Technique Chandra Shekhar Gautam1 and Dr. Prabhat Pandey2 International Journal of Current Engineering and Technology, Volume 13, Year 2022

Nouf Matar Alzahrani. 2020. Augmented Reality: A Systematic Review of Its Benefits and Challenges in E-learning Contexts (2020), 1–21.

Redowan Mahmud, Samodha Pallewatta, Mohammad Goudarzi, and Rajkumar Buyya. 2022. Ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments. Journal of Systems and Software 190 (2022), 111351.

Marco Dorigo and Luca Maria Gambardella. 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1, 1 (1997), 53–66.

Jacky Cao, Kit-Yung Lam, Lik-Hang Lee, Xiaoli Liu, Pan Hui, and Xiang Su. 2023. Mobile augmented reality: User interfaces, frameworks, and intelligence. Comput. Surveys 55, 9 (2023), 1–36.

Basmah K. Alotaibi and Uthman Baroudi. 2022. Offload and Schedule Tasks in a Health Environment using Ant Colony Optimization at Fog Master. In 2022 International Wireless Communications and Mobile Computing (IWCMC). 469–474.

Melike Erol-Kantarci and Sukhmani Sukhmani. 2018. Caching and computing at the edge for mobile augmented reality and virtual reality (AR/VR) in 5G. Ad Hoc Networks (2018), 169–177.

Jiayu He. 2022. Optimization of Edge Delay Sensitive Task Scheduling Based on Genetic Algorithm. In 2022 International Conference on Algorithms, Data Mining, and Information Technology (ADMIT). 155–159. https://doi.org/10.1109/ADMIT57209.2022.00032

Amit Kishor and Chinmay Chakarbarty. 2021. Task offloading in fog computing for using smart ant colony optimization - wireless personal communications. https://link.springer.com/article/10.1007/s11277- 021-08714-7

Li Qin, Qi Zengqing, Lin Weiwei, and Xu Zhiqiang. 2021. Smart Energy Station Terminal 5G Adaptation Strategy Based on Genetic-Algorithm Task Offloading Method. In 2021 IEEE 21st International Conference on Communication Technology (ICCT). 559–564.

Jinke Ren, Yinghui He, Guan Huang, Guanding Yu, Yunlong Cai, and Zhaoyang Zhang. 2019. An edge-computing-based architecture for mobile augmented reality. IEEE Network 33, 4 (2019), 162–169

Chandra Shekhar Gautam1, Pandey* (2019)" A REVIEW ON GENETIC ALGORITHM MODELS FOR HADOOP MAPREDUCE IN BIG DATA" IJSRS ISSN:0976-3031, Vol.13, Issue-03(E), Pageno771-775, June 2022

Clustering of Bigdata Using Genetic Algorithm in Hadoop MapReduce Chandra Shekhar Gautam, Mr. L N SONI, P Pandey European chemical bulletin Year 2022, issue 12,963-973

iFogSimToolkit. 2022. The iFogSimToolkit (with its new release iFogSim2) for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge, and Fog Computing Environments. https://github.com/Cloudslab/iFogSim

Yumei Li, Xiumin Zhu, Shudian Song, Shuyue Ma, Feng Yang, and Linbo Zhai. 2023. Task offloading and parameters optimization of MAR in multi-access edge computing. Expert Systems with Applications 215 (2023), 119379.

An Efficient Approach for Cloud Computing based on Hierarchical Secure Paravirtualization System Resource Model Deepika Patidar, P S Patheja and Akhilesh A. Waoo Year 2012, Vol.7

Dr. Amol Ramesh Ranadive, Dr. Akhilesh A. Waoo et. al., Augmented Reality and Artificial Intelligence Based Visual Learning Education System for Deaf People with Optional Sign Language Tool, IN, 202121010361, 2021-03-11, 2021/4/30.