International Journal of Mathematical, Engineering and Management Sciences
Vol. 10, No. 2, 506-521, 2025
https://doi.org/10.33889/IJMEMS.2025.10.2.025

Ram Arti Publishers

Network Usage and Time Efficient Performance Enhancement Model for IoT Applications in Fog Cloud Computing

Upma Arora
Department of Computer Science,
Gurukul Kangri (Deemed to be) University, Haridwar, Uttarakhand, India.
Corresponing author: upma.arora09@gmail.com

Nipur Singh
Department of Computer Science,
Gurukul Kangri (Deemed to be) University, Haridwar, Uttarakhand, India.
E-mail: nipur@gkv.ac.in

(Received on June 9, 2024; Revised on September 8, 2024 & December 27, 2024; Accepted on January 15, 2025)

Abstract
Fog computing bridges IoT applications and cloud computing, providing low latency services through local computation and storage. Despite its advantages, challenges such as efficient scheduling and placement of IoT applications on fog cloud nodes hinder its widespread adoption. This manuscript presents a Performance Enhancement Algorithm for scheduling IoT applications in a fog cloud environment. The algorithm comprises four key procedures that schedule application modules across the available infrastructure devices. Merge sort along with the heterogeneous shortest module first strategy is used in fusion as the key to improve the performance. The effectiveness of this proposed algorithm was evaluated using the iFogSim simulator and the results demonstrate significant improvements, with total network usage improvement of 66.31% over HSMF, 88.40% over edge-wards, and 98.66% over the cloud-only method. Also, it improves the execution time significantly in most network configurations. Our research contributes to providing a very reliable means for placing applications in the fog-cloud infrastructure as our fusion algorithm makes it a very stable and scalable for CPU bound tasks in fog and cloud computing environment.

Keywords- IoT applications, Fog computing, Cloud computing, iFogSim, Performance, Network usage.

1. Introduction

Fog computing, renowned for its proximity to edge devices, addresses the imperative for latency sensitivity and real-time processing because nodes of fog are closest to the user (Atlam et al., 2018). It is a decentralized computing architecture that extends cloud computing capabilities to the edge of the network, closer to end users and data sources. In this model, data processing, storage, and applications are handled at intermediate layers, often at or near the devices themselves, instead of relying solely on centralized cloud servers.

Placing IoT applications in the fog layer facilitates immediate data processing near the source, thereby minimizing latency and augmenting responsiveness. To maximize the effectiveness of the fog nodes, all the applications to be deployed on system are further split up in number of modules which will be linked to one another. These modules are then placed in sequence based on their requirements (Mahmud et al., 2018). Conversely, cloud computing boasts expansive storage capabilities, substantial computational resources, and scalability. Extending IoT applications to the cloud empowers organizations with centralized data management, extensive analytics capabilities, and adaptable resources. This proves advantageous for applications demanding comprehensive data analysis, prolonged data storage, and intricate computational tasks.

For such scenarios, where very low latency is required for computation and communication, the wiser approach is to place IoT devices near the periphery of computational units. That is where fog computing comes into the picture. One of the statements of OpenFog (Byers and Swanson, 2017) is that fog computing has a hierarchy that gives all the services like computing, networking, software etc from farthest server of cloud to nearest Internet of Things (IoT). Fog computing augments cloud computing and works at an intermediate level between IoT devices and the distant cloud infrastructure. It allows having multiple fog nodes of low capacity as opposed to the traditional cloud centres. Even then it minimizes the delay due to the reduction in network congestion, and the ability to discard the useless data at fog level only. This saves a lot of time and effort by expensive cloud resources because the unrequired data is never communicated to the cloud.

The strategic deployment of application modules within a fog-cloud system entails very purposeful integration of these applications across both fog and cloud computing layers. IoT applications, encompassing interconnected devices and sensors, are positioned to capitalize on the unique advantages offered by each computing paradigm. Network usage and execution time are two main performance indicators to access any application placement strategy in cloud or fog computing environment out of the other indictors like cost, scalability, energy consumption etc.

When considered alone fog computing poses various restrictions that hamper its adaptation (Asensio et al., 2020). So, we need to implement strategies to effectively utilize fog and cloud computing resources by the IoT applications. As IoT applications are made up of various modules that are connected to perform various tasks. So, these modules are needed to be scheduled on the available nodes of combined fog and cloud computing environments. Although there are strategies like cloud-only, edge-wards (Gupta et al., 2017), and HSMF (Arora and Singh, 2021) to deploy the modules on such low-equipped fog nodes, the problem with the existing strategies is that they take a very long time to execute along with their high usage of the network.

Having noted the problem with the existing strategies to schedule the modules of an application in the fog and cloud environment, we are focused to formulae a method to place this application in a combined fog-cloud system that keeps track of network usage and the execution time as these performance matrices are of utmost importance.

The rest of the paper is organized as follows:

- Section 2 gives a detailed literature survey about the already developed IoT application scheduling algorithms in fog, cloud or integrated fog-cloud environments available in the literature.
- Section 3 is regarding the methodology, introducing our proposed algorithm with clearly defined steps for IoT applications placement in a fog-cloud combined system. It has a sub-section 3.1, which gives an insight to the simulation and configuration settings.
- Section 4 elaborates the findings and results for result matrices in two sub-sections: sub-section 4.1 for total network usage and sub-section 4.2 for execution time.
- Section 5 concludes all the findings and results along with a sneak peek to our future endeavours and the scenarios where our proposed algorithm can be utilized.

2. Literature Survey

In recent years, research in fog computing has gained momentum. Possibilities are being searched in fog computing for various dimensions ranging from designing the architecture, workload allocation, scheduling policies, application module placement and many more with the intent to improve QoS and QoE as compared to only cloud infrastructure. Researcher (Gupta et al., 2017) has given two placement strategies for application modules.

The first one is cloud-only strategy that directly deploys the modules on the nodes of cloud only, which results in highest network usage because of the modules being placed on the farthest cloud nodes (Gupta et al., 2017). This strategy also takes highest time for execution due to the obvious fact of not having fog nodes to place them nearby to reduce the time. The same authors have given another strategy which is edge-wards strategy that deploys the modules near the edge while going upwards towards the farthest available cloud data center. This strategy somehow performs better in terms of placing the application modules near to the IoT devices itself and improves network usage and execution time significantly in comparison to the cloud-only method.

Renowned authors (Taneja and Davy, 2017) present an algorithm that effectively maps the IoT applications to fog devices while properly utilizing the resources. Similarly, (Mahmud et al., 2020) have implemented a placement policy for applications in fog-cloud environments which considers the overall profitability.

To select appropriate nodes for placing IoT application modules (Natesha and Guddeti, 2018) designed a heuristic-based First Fit Decreasing (FFD) approach. Also, pioneers in this field (Rezazadeh et al., 2018) give algorithm using simulated annealing for finding right devices for the module in fog computing environment. Their results have claimed to be reducing cost delay and energy by high percentage.

Another model for scheduling IoT applications is Heterogeneous Shortest Module First (Arora and Singh, 2021). This approach first sorts the application modules in order of their requirement from smallest to the highest and then places these modules on the fog nodes of heterogeneous capacities. This strategy further improves results in comparison to edge wards but still there is room for improvement.

Similar to our objectives, (Gavaber and Rajabzadeh, 2021) have developed a method for placing applications in fog based IoT system. The method intends to minimize the delay by using the network efficiently. (Baranwal and Vidyarthi, 2021) proposed a model for selecting Fog Orchestrator Node (FON). This model enables even the least powerful IoT device to select FON. Concerns like latency along with the power consumption are taken care of by the researchers (Fang and Ma, 2020) using a heuristic that executes the tasks dynamically.

A nature inspired optimization algorithm given by Harris Hawks (Heidari et al., 2019), is also widely being used in a variety of areas such as data mining, engineering optimization, power systems etc. and can also be applied to examine its efficacy in fog computing. (Rezazadeh et al., 2019) propose an algorithm for fog-cloud environment that takes care of latency. This algorithm places modules of an application into the available devices with less delay by setting appropriate location of host device modulus. Their results show the efficiency of the algorithm. For placing the applications in the decentralized environment (Mann, 2022) consideration is to divide the complete infrastructure in different fog colonies and place these applications in the colonies that consist of fog nodes with required capabilities to handle the demand and they compare four approaches viz. centralized decision making, independent fog colonies, fog colonies with communication and fog colonies with overlaps. Their results demonstrate that for large problems particularly combining decentralized and coordination provides the best results.

For utilizing the resources in fog environment, the authors provide a placement scheme for smart farming system (Mohamed et al., 2023). The developed scheme results significantly in reducing delay and network usage. To schedule the IoT modules in fog-cloud environment, authors have given a hybrid meta-heuristic algorithm, which is inspired from (Fard and Hajiaghaei-Keshteli, 2016) RDA and HHO (Heidari et al., 2019).

Execution time and energy consumption minimization is another major goal addressed by the researchers by developing a weighted cost model in fog-based systems (Goudarzi et al., 2021). An Antlion optimization algorithm (Dewangan et al., 2020) that reduces the SLA (Service Level Agreement) rate by allocating resources in cloud also seem to be quite efficient.

Table 1 summarizes all the available strategies to schedule the applications of IoT in fog computing, cloud computing, or fog-cloud computing environments. There are various objectives like resource utilization, latency awareness, dynamicity, and cost etc. that are taken into considerations by various researchers while strategizing the placement of IoT applications.

Table 1. Related work available in the literature.

Model	Objective base for module placing	Application placement environment			Improved result matrices				
		Fog	Cloud	Fog-Cloud	Latency	Cost	Exe-Time	Energy	N/W usage
Taneja and Davy (2017)	Resource Utilization			✓	✓			✓	✓
Natesha and Guddeti. (2018)	Heuristic Based Placement			✓	✓			✓	
Rezazadeh et al. (2018)	Simulated Annealing based Placement	✓			✓	✓		✓	
Al-Tarawneh (2022)	Bi-objective Knapsack based			✓	✓		✓		✓
Mahmud et al. (2018)	Latency aware placement			✓	✓		✓		
Fang and Ma (2020)	Heuristic dynamic based			✓	✓				
Gavaber and Rajabzadeh (2021)	Dynamic voltage and Frequency Scaling			✓	✓				✓
Tärneberg et al. (2017)	Cost Optimal Application Placement		✓						
Faticanti et al. (2020)	Containerized Microservice Modules			✓		✓			✓
Barnwal and Vidyarthi (2021)	FON selection				✓				✓
Mann (2022)	Decentralize Application Placement	✓			✓		✓	✓	
Mohamed et al. (2023)	Module placement for smart Farming System	✓			✓				✓
Goudarzi et al. (2021)	Weighted cost model	✓					✓	✓	
Maiti et al. (2022)	Heuristic Dynamic	✓					✓		
Arora and Singh (2021)	Modules placement in fog-cloud environment with heterogeneity			✓			✓		✓
Muhamad et al. (2023)	Task offloading	✓						✓	

Although each of the research works mentioned in this section addresses various problems and objectives related to placing and scheduling of application in fog and cloud infrastructure, the gaps remain as these approaches do not collectively place the IoT applications in fog and cloud environment together with the considerations of executing them timely along with the proper usage of available network configurations. For this problem, we formulate a Performance Enhancement Algorithm that places applications onto the combined infrastructure of fog and cloud with utmost efficiency of time and network usage in Section 3.

3. Methodology

The decision to integrate fog computing environment for organizations depends on various factors. One of the most considerable questions can be whether it will enhance the performance of the existing system. There are numerous indicators of performance for any placement strategy in any fog computing environment viz. latency, throughput, reliability, resource utilization, energy efficiency, network usage, etc. Our proposed model enhances the performance by placing the IoT applications such that it reduces the overall time of executing the modules and network usage. It is presented as **Algorithm I**, Performance Enhancement Algorithm, which is again divided into four procedures.

Our algorithm uses the Heterogeneous Shortest Module First (HSMF) strategy along with merge sort for scheduling the modules because of the following reasons:

- For CPU-bound tasks, especially in environments like cloud computing or fog networks, HSMF ensures that critical tasks are completed first, enhancing user experience and system responsiveness.
- Along with HSMF we use merge sort because of it being stable sorting technique as it takes care of the sequence of items with equal keys.
- Merge sort guarantees the time complexity of O(n log n) in every cases. This makes it consistently efficient, compared to other algorithms such as quick sort, which have time complexity of $O(n^2)$ in the worst case.
- Also, the choice of using merge sort makes it more scalable and stable for large data sets.

Algorithm I starts by taking list ML as a collection of application modules $m_1, m_2, \ldots m_n$ to be placed on the available fog devices list FD as a collection of fog devices FD_1, $FD_2, \ldots FD_n$ and we call these as Module_List and Fog_Device_List respectively. In the next step we define two sub-lists L and R of the original module list ML and the lengths of these two lists as N_1 and N_2 respectively. Step 3 involves adding sensors and actuators and the latencies between fog-cloud layers are set up in Step 4. Step 5 calls the method mergeSort(), which first divide the list of modules in two parts and calls mergeSort() method recursively on both of the lists.

Algorithm I: ***Performance Enhancement Algorithm***
Input: Module_List, Fog_Device_List
Output: Sorted_ List, Execution_Time, Total_Network_Usage

Step 1. Define Module_List as ML[$M_1 \ldots . M_m, \ldots M_n$] and Fog_Device_List as FD[$FD_1, \ldots, FD_n$]
Step 2. Define First Sublist as L and Second Sublist as R and their lengths as N_1 and N_2 respectively
Step 3. Add sensors and actuators
Step 4. Setup the latencies between fog-cloud devices layers
Step 5. mergeSort (ML[], M_1, M_n)
{
middle $M = M_1 + (M_n - M_1)/2$
mergeSort (ML[], M_1, M_m)

```
                mergeSort (ML[], M_{m+1}, M_n)
                call Procedure I merge (ML[], M_1, M_m, M_n)
                Return Sorted_List
        }
```

Step 6. call Procedure II placeModulesOnFogDevices ()
Step 7. call Procedure III calculteExecutionTime()
Step 8. call Procedure IV calculateTotalNetworkUsage()
Step 9. End

Here, in **Procedure I**, we are sorting and merging the two sub arrays of the application modules based on their MIPS(Million Instructions Per Second) requirements because MIPS is the measure of processing speed and this is the factor that is of most importance when we aim to improve the performance matrices such as execution time and network usage.

Procedure I

```
merge (ML[], M_1, M_m, M_n)
{
N_1 = M_m-M_1+1
N_2 = M_n-M_m
/*Copy the data to temporary Lists L and R*/
for (int α = 0; α < N1; ++ α)
L[α] = ML[M_1 + α]
for (int β = 0; β < N_2; ++ β)
R[β] = ML[M_m + M_1 + β]
        Initialize α =0, β =0, μ=0
        while (α < N_1 && β < N_2)
        {
                if (L[α].getMips() < R[β].getMips())
                {
                Sorted_List[μ]= L[α]
                α ++;
                }
                else
                {
                Sorted_List[μ]= R[β]
                β ++;
                }
                μ ++; /* move to the next position in the merged list*/
        }
while (α < N_1) /*Copy the remaining elements of L[], if any*/
        {
        Sorted_List[μ]= L[α]
        α ++;
        μ ++;
        }
        while (β < N_2) /*Copy the remaining elements of R[], if any*/
        {
```

```
            Sorted_List[μ]= R[β]
            β ++;
            μ ++;
}
}
```

After sorting the list of modules by merge sort taking the base for merge sort as given MIPS of different modules, **Procedure II** then takes over which calls a function placeModulesOnFogDevices () to place the list of sorted modules on available fog-cloud nodes from Fog_Device_List.

Procedure II

```
placeModulesOnFogDevices (Sorted_List, Fog_Device_List)
{
      for k=1 to Sorted_List.size()
      {
      If (Sorted_List[k].getMips() < (Fog_Device_List[k].getMips())
      {
      Sorted_List[k]→ Fog_Device_List[k]
      }
      else
      {
      Sorted_List[k]→ Fog_Device_List[k+1]
      }
}
}
```

Procedure III

```
calculateExecutionTime()
{
      Execution_Time= Endtime() - StartTime()
}
```

Procedure IV

```
calculateTotalNetworkusage()
{
      Total_Network_Usage= Network_Usage+ (Latency of Computation* Network Size of Tuples)
}
```

3.1 Simulation Setup and Configuration Details

Our proposed algorithm is implemented using the classes and methods of iFogSim simulator. The simulation was performed on a Dell Inspiron, quad-core, 11th generation system with an 8 GB inbuilt RAM. The following steps demonstrate on how to setup the iFogSim simulator to integrate our proposed strategy:

Step 1. Download and install iFogSim from its official GitHub repository. https://github.com/Cloudslab/iFogSim1.

Step 2. Java and Eclipse IDE set up should already be there in the system since iFogSim is a Java-based framework.

Step 3. Import the iFogSim project into the Eclipse workspace and ensure all dependencies are correctly configured.
Step 4. Define fog nodes (like proxy, gateways, and cloud) using FogDevice class.
Step 5. Define the modules of your application and data flow between these modules using AppModule class.
Step 6. Define the Application Placement Strategy. An Abstract class named ModulePlacement is extended to include our proposed algorithm.
Step 7. Set up the link latencies among different level fog devices using FogDevice.setUplinkLatency().
Step 8. Define scheduling policy using the updateAllocatedMips method inside the FogDevice class.
Step 9. Use the FogBroker to create users, applications, and manage the resources.
Step 10. Use the Controller class to initiate and control the simulation.
Step 11. Analyze the resultant matrices.

Our proposed algorithm is integrated in the distributed camera network application in iFogSim simulator. **Figure 1** shows the directed acyclic graph of this application. There are 5 modules named Motion Detector (M.D.), Object Detector (O.D.), Object Tracker (O.T.), User Interface (U.I.) and PTZ_controller (P.C.) in the application along with the edges among these modules.

We assume two things for configurations:

- As this distributed network spans across different number of areas, our proposed algorithm is tested on 5 different configurations. In each configuration, there are 4 Cameras but each configuration has a different number of areas to be monitored. So, in configuration 1, there is only 1 area, in configuration 2, there are 2 areas, in configuration 3, there are 4 areas. Similarly in configuration 4 and configuration 5 there are 8 and 16 number of areas respectively. The configurations are chosen such so as to make the results comparable using all the four strategies of application placement.
- We have kept number of Cameras in each area as 4 in our work, while we can increase the number of areas. This makes the infrastructure more stable and scalable at the same time.

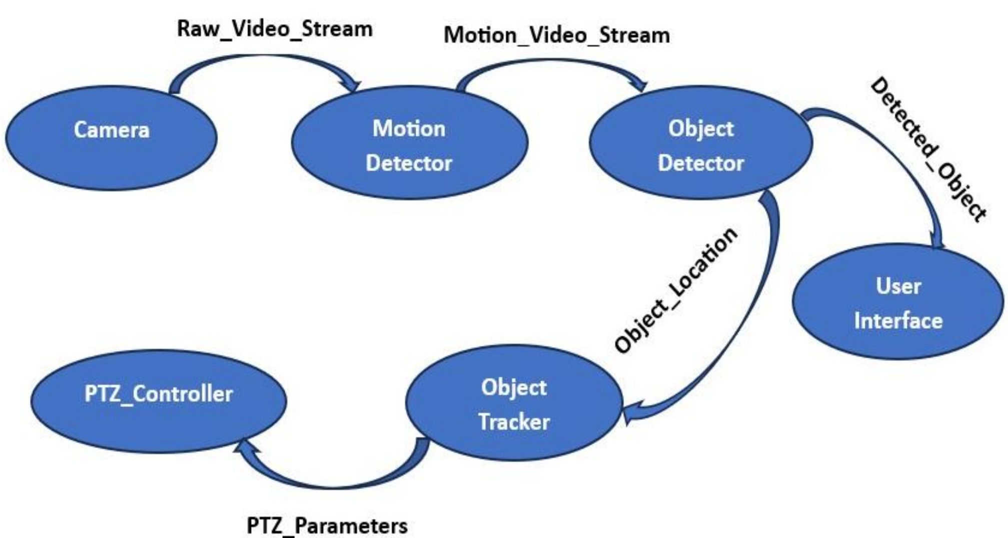

Figure 1. Modules and edges in distributed camera application.

Table 2. Latency for devices in configuration 1.

Cloud -Proxy (80-100ms)	Proxy-Area Gateway1 (1-10ms)	Area Gateway1-Camera (1-4) (1-10ms)	
86	7	**C 1**	4
		C 2	10
		C 3	3
		C 4	7

Table 2 shows that the latency for devices in configuration 1. Between the cloud and proxy server the latency can range from 80 ms to 100 ms and for this particular instance, the latency is 86 ms (where, ms stands for milliseconds). Similarly, the latency between proxy and Area gateway 1, the latency can range from 1 ms to 10 ms and for this instance, the latency is 7 ms. Likewise, the latency between Area Gateway1 to Cameras- C 1, C 2, C 3, C 4 can range from 1ms to 10 ms and for this particular instance the latency values are 4 ms, 10 ms, 3 ms and 7 ms respectively.

Table 3. Latency for devices in configuration 2.

Cloud-Proxy (80-100ms)	Proxy-Area Gateway (1-10ms)		Area Gateway-Camera (1-10ms)	
92	**Area Gateway 1**	4	**C 1**	5
			C 2	2
			C 3	6
			C 4	5
	Area Gateway 2	7	**C 1**	9
			C 2	4
			C 3	3
			C 4	6

Table 3 shows the latency values for the devices in configuration 2. The latency between the cloud and proxy server can range from 80 ms to 100 ms. Similarly, the latency between proxy and Area gateway 1, the latency can range from 1 ms to 10 ms. Likewise, the latency between Area Gateway1 to Cameras can range from 1 ms to 10 ms.

Table 4. Latency for devices in configuration 3.

Cloud-Proxy (80-100ms)	Proxy-Area Gateway (1-10ms)		Area Gateway-Camera (1-10ms)	
92	**Area Gateway 1**	4	**C 1**	5
			C 2	2
			C 3	6
			C 4	5
	Area Gateway 2	7	**C 1**	9
			C 2	4
			C 3	3
			C 4	6
	Area Gateway 3	5	**C 1**	10
			C 2	4
			C 3	8
			C 4	3
	Area Gateway 4	10	**C 1**	2
			C 2	10
			C 3	9
			C 4	7

Table 4 shows the latency values for the devices in configuration 3. In configuration 3, number of areas to monitored are 4. So, for every area, there is a gateway. The latency between the cloud and proxy server can

range from 80 ms to 100 ms. Similarly, the latency between proxy and Area gateway 1, the latency can range from 1 ms to 10 ms. Likewise, the latency between Area Gateway 1 to Cameras can range from 1 ms to 10 ms.

Table 5. Latency for devices in configuration 4.

Cloud-Proxy (80-100ms)	Proxy-Area Gateway (1-10ms)		Area Gateway-Camera (1-10ms)	
92	**Area Gateway 1**	10	**C 1**	9
			C 2	3
			C 3	7
			C 4	10
	Area Gateway 2	6	**C 1**	4
			C 2	2
			C 3	8
			C 4	4
	Area Gateway 3	2	**C 1**	7
			C 2	4
			C 3	3
			C 4	8
	Area Gateway 4	7	**C 1**	2
			C 2	10
			C 3	2
			C 4	7
	Area Gateway 5	3	**C 1**	9
			C 2	10
			C 3	1
			C 4	6
	Area Gateway 6	5	**C 1**	8
			C 2	1
			C 3	4
			C 4	2
	Area Gateway 7	5	**C 1**	3
			C 2	8
			C 3	10
			C 4	9
	Area Gateway 8	6	**C 1**	4
			C 2	2
			C 3	8
			C 4	9

Table 5 shows the latency values for the devices in configuration 4. Here, the number of areas that are to be monitored is 8. So, there are 8 area gateways, one for each area. The latency between the cloud and proxy server can range from 80 ms to 100 ms. Similarly, the latency between proxy and Area gateways can range from 1 ms to 10 ms. Likewise, the latency between Area Gateways to Cameras can range from 1 ms to 10 ms.

Table 6. Latency for devices in configuration 5.

Cloud-Proxy (80-100ms)	Proxy-Area Gateway (1-10ms)		Area Gateway-Camera (1-10ms)	
92	**Area Gateway 1**	10	**C 1**	9
			C 2	3
			C 3	7
			C 4	10
	Area Gateway 2	6	**C 1**	4
			C 2	2
			C 3	8
			C 4	4

Table 6 continued…

	Area Gateway 3	2	**C 1**	7
			C 2	4
			C 3	3
			C 4	8
	Area Gateway 4	7	**C 1**	2
			C 2	10
			C 3	2
			C 4	7
	Area Gateway 5	3	**C 1**	9
			C 2	10
			C 3	1
			C 4	6
	Area Gateway 6	5	**C 1**	8
			C 2	1
			C 3	4
			C 4	2
	Area Gateway 7	5	**C 1**	3
			C 2	8
			C 3	10
			C 4	9
	Area Gateway 8	6	**C 1**	4
			C 2	2
			C 3	8
			C 4	9
	Area Gateway 9	9	**C 1**	6
			C 2	2
			C 3	7
			C 4	10
	Area Gateway 10	1	**C 1**	4
			C 2	6
			C 3	9
			C 4	9
	Area Gateway 11	8	**C 1**	5
			C 2	7
			C 3	10
			C 4	3
	Area Gateway 12	9	**C 1**	6
			C 2	1
			C 3	10
			C 4	7
	Area Gateway 13	10	**C 1**	3
			C 2	8
			C 3	9
			C 4	5
	Area Gateway 14	5	**C 1**	8
			C 2	10
			C 3	9
			C 4	3
	Area Gateway 15	6	**C 1**	1
			C 2	7
			C 3	4
			C 4	7
	Area Gateway 16	8	**C 1**	3
			C 2	6
			C 3	7
			C 4	5

Table 6 shows the latency values for the devices in configuration 5. Here, the number of areas that are to be monitored is 16. So, there are 16 area gateways, one for each area. The latency between the cloud and proxy server can range from 80 ms to 100 ms. Similarly, the latency between proxy and Area gateways can

range from 1 ms to 10 ms. Likewise, the latency between Area Gateways to Cameras can range from 1 ms to 10 ms.

Table 7. MIPS of devices in the application.

Fog device	Cloud	Proxy	Router	Camera
MIPS	44800	2800	2800	500

Table 7 shows the MIPS values of all the fog devices and similarly, **Table 8** shows MIPS of modules to be scheduled on the fog devices.

Table 8. MIPS of modules in the application.

Modules	Object Detector	Motion Detector	Object Tracker	User Interface	PTZ Controller
MIPS	1549	978	12753	2250	349

4. Result and Discussion

For five different configurations of the network (as explained in section 3.1), we have evaluated the results by applying four approaches namely cloud-only, edgewards, HSMF and Performance Enhancement Algorithm. The resulting matrices are explained as follows:

4.1 Total Network Usage (KB)

Figure 2 shows the total network usage TNU (Kilo Bytes) for four different approaches of application placement. As we increase the overall number of areas to be covered in the network in different configurations, the total network usage also increases gradually because there is an increase in the load

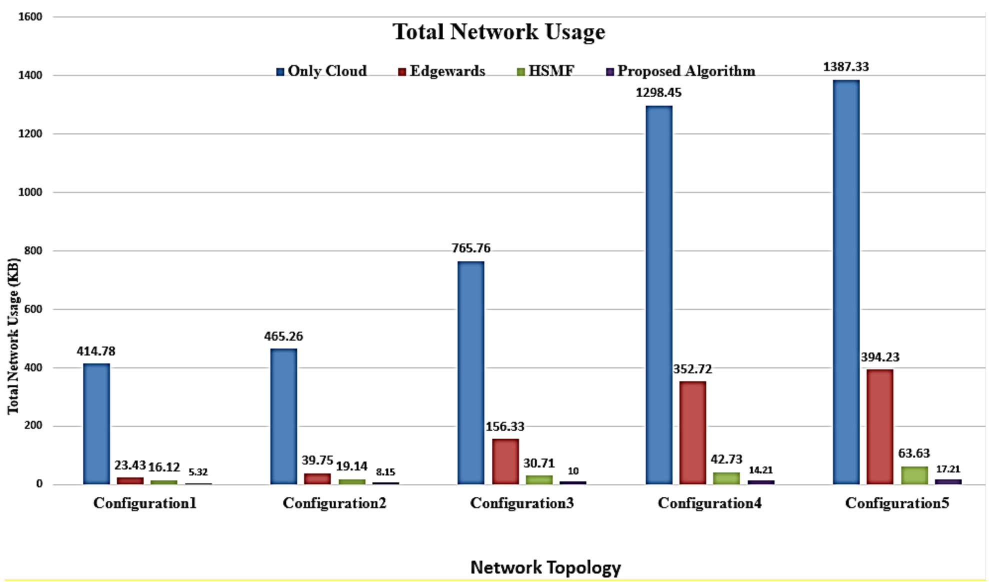

Figure 2. TNU after applying four different algorithms.

for the network. Here, the minimum network usage is in case of using Performance Enhancement Algorithm as compared to the maximum network usage in case of cloud-only because preprocessing in terms of sorting of the modules is being done by merge sort which has the lowest complexity. As and when required, the infrastructure can be scaled up and the number of areas can be increased. By increasing number of areas, there will be an obvious increment in the total network usage but that will always be far less than the other three strategies. The results show substantial enhancements, with total network usage improved by 66.31% compared to HSMF, 88.40% compared to edge-wards, and 98.66% in comparison to the cloud-only approach.

Further, if we keep on scaling up the network by adding significantly large number of devices (or increasing workload), the trend will follow in same fashion. The TNU will always be quite lesser in comparison to the other three algorithms. The improvements in network usage are due to the fact that our algorithm minimizes idle time in fog nodes by allowing shorter tasks to run immediately and locally, preventing unnecessary task migrations to the cloud, which would otherwise increase network traffic.

4.2 Execution Time

Figure 3 shows that for all 5 configurations, the execution time (milliseconds) shows linear increment when we increase number of areas in case when the modules are placed directly by only-cloud strategy. In all five configurations the maximum execution time is for this placement approach which is very obvious due to the fact that all the modules are being placed on the farthest cloud server. For configuration 1 and configuration 2, as we increase the number of areas, the execution time is minimal in case of our propose algorithm but as the number of areas further increase in configuration 3 to configuration 5, the execution increases slightly in comparison to HSMF algorithm though this increase is not much if we consider the overall cost incurred to setup the network. But as we scale up the network by adding significantly large number of devices, and ultimately increase the workload, the execution time will impose the bottleneck. This concludes that there will sometimes be a trade-off call between execution time and network usage for the users.

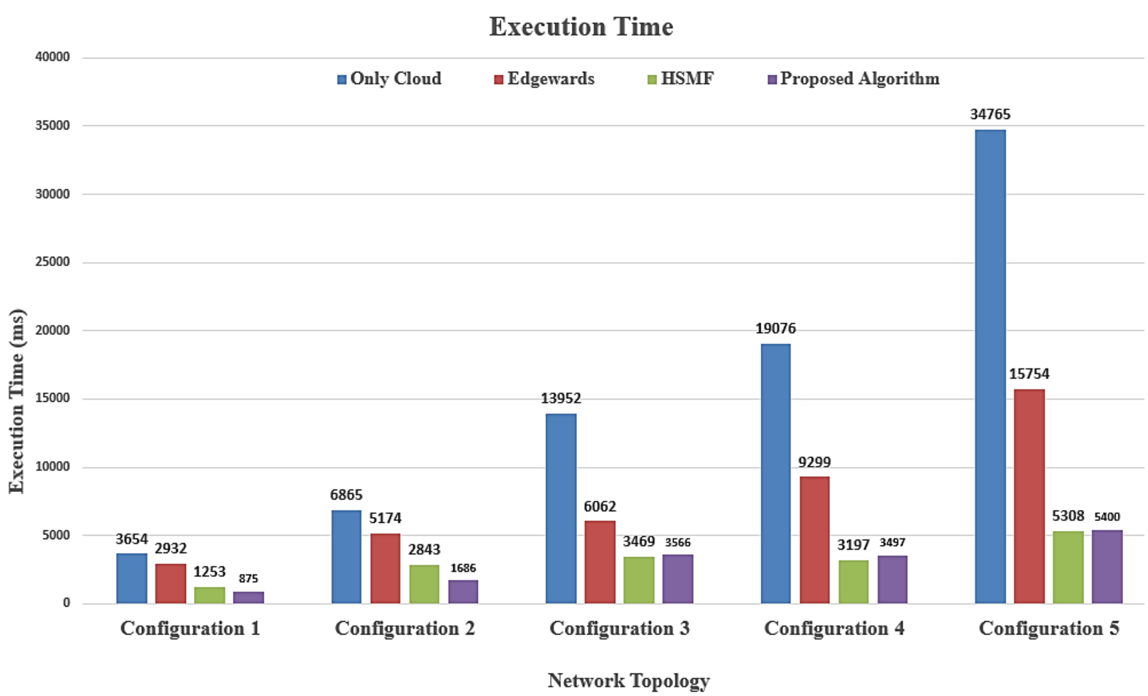

Figure 3. Execution time after applying four different algorithms.

5. Conclusion

Placing IoT applications in a combined fog-cloud system is an utmost challenging task. In our work, we have come up with a solution to this problem. In this study, we have presented a performance enhancement algorithm designed to optimize the scheduling of IoT applications in fog and cloud environment together. Our proposed approach leverages a merge sort and the shortest module first strategy to efficiently allocate application modules across available infrastructure devices.

The results demonstrate that our algorithm significantly enhances performance compared to existing methods. Specifically, our proposed algorithm achieves a notable reduction in total network usage, with improvements of 66.31% over the HSMF strategy, 88.40% over the edge-wards approach, and 98.66% over the cloud-only method. Additionally, while execution time increases with the number of areas, our algorithm consistently outperforms the cloud-only strategy and shows competitive performance compared to the HSMF algorithm.

The results are represented in five configurations of network, increasing the number of areas and ultimately adding more devices to the network results in similar trend. The network usage will be minimum in case of using our proposed algorithm in comparison to other three approaches even if number of areas keep on increasing. So, our algorithm can be used as and when there is a need of scaling up in the infrastructure with the requirement to improve total network usage.

Though, when we have such IoT applications that emphasize on reducing execution time, the trade-off call would limit us to use our proposed algorithm in a network with very limited number of fog-cloud devices. Because, when we try to add large number of devices to the network, the execution time, instead of decreasing, will start to increase (though not significantly much). So, as the results suggest, applications with the very specific need to have less execution time in larger network conditions can opt for HSMF also. These findings show the effectiveness of our approach in reducing network load and improving efficiency, making it a valuable contribution to the field of fog-cloud computing. It also gives us insight to the trade-off condition as to when should one go for our proposed algorithm or alternatively for HSMF.

Future work could explore further optimizations of other QoS Parameters, such as overall cost incurred, application loop delays, power consumption etc. We also plan to scale up this algorithm in more complex network configurations like flexibility to change number of the cameras in each area etc. Furthermore, our findings that improves the total network usage gives us substantial direction to apply this algorithm in Industrial Internet of Things (IIOT), gaming and healthcare applications etc.

Conflict of Interest

The authors confirm that they do not have any conflict of interest to declare for this publication.

AI Disclosure

The author(s) declare that no assistance is taken from generative AI to write this article.

Acknowledgments

The authors extend their gratitude and heartfelt thanks for his guidance to Dr. Vivek Kumar, Professor and Head, Department of Computer Science, Gurukul Kangari (Deemed to be University), Haridwar, India. The authors extend their gratitude and heartfelt thanks for his guidance to Dr. Rajkumar, Assistant Professor, Department of Computer Science, Gurukul Kangari (Deemed to be University), Haridwar, India.

References

Al-Tarawneh, M.A.B. (2022). Bi-objective optimization of application placement in fog computing environments. *Journal of Ambient Intelligence and Humanized Computing*, *13*(1), 445-468. https://doi.org/10.1007/s12652-021-02910-w.

Arora, U., & Singh, N. (2021). IoT application modules placement in heterogeneous fog–cloud infrastructure. *International Journal of Information Technology*, *13*(5), 1975-1982. https://doi.org/10.1007/s41870-021-00672-4.

Asensio, A., Masip-Bruin, X., Durán, R.J., de Miguel, I., Ren, G., Daijavad, S., & Jukan, A. (2020). Designing an efficient clustering strategy for combined fog-to-cloud scenarios. *Future Generation Computer Systems*, *109*, 392-406. https://doi.org/10.1016/j.future.2020.03.056.

Atlam, H.F., Walters, R.J., & Wills, G.B. (2018). Fog computing and the internet of things: a review. *Big Data and Cognitive Computing*, *2*(2), 10. https://doi.org/10.3390/bdcc2020010.

Baranwal, G., & Vidyarthi, D.P. (2021). FONS: a fog orchestrator node selection model to improve application placement in fog computing. *The Journal of Supercomputing*, *77*, 10562-10589. https://doi.org/10.1007/s11227-021-03702-x.

Byers, C., & Swanson, R. (2017). OpenFog consortium OpenFog reference architecture for fog computing. *OpenFog Consortium Architecture Working Group, Fremont* (pp. 27-28), CA, USA.

Dewangan, B.K., Agarwal, A., Choudhury, T., & Pasricha, A. (2020). Cloud resource optimization system based on time and cost. *International Journal of Mathematical, Engineering and Management Sciences*, *5*(4), 758-768. https://doi.org/10.33889/ijmems.2020.5.4.060.

Fang, J., & Ma, A. (2020). IoT application modules placement and dynamic task processing in edge-cloud computing. *IEEE Internet of Things Journal*, *8*(16), 12771-12781. https://doi.org/10.1109/jiot.2020.3007751.

Fard, A.F., & Hajiaghaei-Keshteli, M. (2016). Red deer algorithm (RDA); a new optimization algorithm inspired by Red Deers' mating. In *International Conference on Industrial Engineering* (Vol. 12, pp. 331-342). IEEE, Iran.

Faticanti, F., De Pellegrini, F., Siracusa, D., Santoro, D., & Cretti, S. (2020). Throughput-aware partitioning and placement of applications in fog computing. *IEEE Transactions on Network and Service Management*, *17*(4), 2436-2450. https://doi.org/10.1109/tnsm.2020.3023011.

Gavaber, M.D., & Rajabzadeh, A. (2021). BADEP: bandwidth and delay efficient application placement in fog-based IoT systems. *Transactions on Emerging Telecommunications Technologies*, *32*(8), e4136. https://doi.org/10.1002/ett.4136.

Goudarzi, M., Wu, H., Palaniswami, M., & Buyya, R. (2021). An application placement technique for concurrent IoT applications in edge and fog computing environments. *IEEE Transactions on Mobile Computing*, *20*(4), 1298-1311. https://doi.org/10.1109/tmc.2020.2967041.

Gupta, H., Dastjerdi, A.V., Ghosh, S.K., & Buyya, R. (2017). iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Journal of Software: Practice and Experience*, *47*(9), 1275-1296. https://doi.org/10.1002/spe.2509.

Heidari, A.A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., & Chen, H. (2019). Harris hawks optimization: algorithm and applications. *Future Generation Computer Systems*, *97*, 849-872. https://doi.org/10.1016/j.future.2019.02.028.

Mahmud, R., Ramamohanarao, K., & Buyya, R. (2018). Latency-aware application module management for fog computing environments. *ACM Transactions on Internet Technology*, *19*(1), 1-21. https://doi.org/10.1145/3186592.

Mahmud, R., Srirama, S.N., Ramamohanarao, K., & Buyya, R. (2020). Profit-aware application placement for integrated fog–cloud computing environments. *Journal of Parallel and Distributed Computing*, *135*, 177-190.

Maiti, P., Sahoo, B., Turuk, A.K., Kumar, A., & Choi, B.J. (2022). Internet of Things applications placement to minimize latency in multi-tier fog computing framework. *ICT Express*, *8*(2), 166-173. https://doi.org/10.1016/j.icte.2021.06.004.

Mann, Z.A. (2022). Decentralized application placement in fog computing. *IEEE Transactions on Parallel and Distributed Systems*, *33*(12), 3262-3273. https://doi.org/10.1109/tpds.2022.3148985.

Mohamed, B., Abdellatif, E., Mouhajir, M., Zerifi, M., Rabah, Y., & Youssef, E. (2023). A module placement scheme for fog-based smart farming applications. *International Journal of Electrical and Computer Engineering*, *13*(6), 7089-7098. https://doi.org/10.11591/ijece.v13i6.pp7089-7098.

Muhamad, W.N.W., Ribep, A.C., Dimyati, K., Yusof, A.L., & Abdullah, E. (2023). Improvement of energy consumption in fog computing via task Offloading. *Journal of Advanced Research in Applied Sciences and Engineering Technology*, *36*(2), 199-212. https://doi.org/10.37934/araset.36.2.199212.

Natesha, B.V., & Guddeti, R.M.R. (2018). Heuristic-based IoT application modules placement in the fog-cloud computing environment. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion* (pp. 24-25). IEEE. Zurich, Switzerland. https://doi.org/10.1109/ucc-companion.2018.00027.

Rezazadeh, Z., Rahbari, D., & Nickray, M. (2018). Optimized module placement in IoT applications based on fog computing. In *Electrical Engineering (ICEE), Iranian Conference on* (pp. 1553-1558). IEEE. Mashhad, Iran. https://doi.org/10.1109/icee.2018.8472469.

Rezazadeh, Z., Rezaei, M., & Nickray, M. (2019). Lamp: a hybrid fog-cloud latency-aware module placement algorithm for IoT applications. In *2019 5th Conference on Knowledge Based Engineering and Innovation* (pp. 845-850). IEEE. Tehran, Iran. https://doi.org/10.1109/kbei.2019.8734958.

Taneja, M., & Davy, A. (2017). Resource aware placement of IoT application modules in fog-cloud computing paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management* (pp. 1222-1228). IEEE. Lisbon, Portugal. https://doi.org/10.23919/inm.2017.7987464.

Tärneberg, W., Mehta, A., Wadbro, E., Tordsson, J., Eker, J., Kihl, M., & Elmroth, E. (2017). Dynamic application placement in the mobile cloud network. *Future Generation Computer Systems*, *70*, 163-177. https://doi.org/10.1016/j.future.2016.06.021.