**RESOURCE AWARE PLACEMENT OF IOT APPLICATION MODULES IN FOG-COMPUTING**

SUGGULA JASWANTH (2021)

**RESOURCE AWARE PLACEMENT OF IOT APPLICATION MODULES IN FOG-CLOUD COMPUTING PARADIGM**

## INTRODUCTION:

In this increasing evolution and progress of hardware and communication technologies, the Internet of Things is racing forward to endorse all areas of the cyber-physical environments. As a result, various IoT aided programs such as smart transport, smart agriculture etc., are getting worldwide consideration.

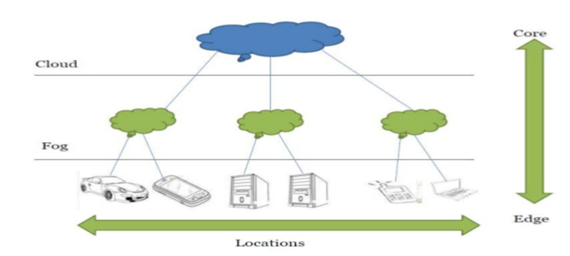
We know that Cloud computing is the platform for providing infrastructure, podium services and software to develop IoT aided systems. So, Cloud data centres remain at a high hop level from IoT data sources that increase delays in data streaming. This problem also has a detrimental effect on the delivery of IoT-enabled system resources and in real-time situations such as critical patient health, emergency fire. IoT devices are distributed geographically and produce a larger amount of data at a time. If the Cloud receives a larger amount of data from IoT devices for processing, the global internet will be overloaded. So to overcome this problem, the involvement of Edge resources could be a solution for using IoT to enable the system.

Edge computing, also known as fog computing, is the most recent installation of computer platforms that point to delivering services such as Cloud to the network in order to assist a greater number of IoT users. Various devices, such as Cisco IOx communication apparatus, micro-data centre, Nano server, Smartphone, personal computer, and Cloudlets, commonly referred to as Fog nodes, allow a large distribution of processing services IoT data near the source in Fog computing. As a result, fog computing will be critical in reducing delays in the delivery of various IoT-enabled applications as well as network acquisition when dealing with large amounts of data. In comparison to Cloud storage, Fog nodes do not receive resource upgrades. As a result, the Fog and Cloud computing paradigms collaborate to address the resources and quality of massive IoT-enabled software services in general.

## Problems in current Fog scenario:

Cloud algorithms are often time-consuming. These results can be clarified by the efficiency of the cloud, which uses a higher latency running path. For low-intensity applications, Edgeward performs slightly better than the Mapping algorithm. When there are more Fog modules, then there exits more application executions, and the edge-wards are more vulnerable to changes in the topology. When the application requires more than the functionality of the Fog System, Edge-ward runs the application modules in cloud devices, increasing latency when compared to the Mapping algorithm. Whereas in Process placement algorithm there exits clustering which means one device needs the same service of the other device, so request is halted and increases more latency. So in this paper, we compare the Genetic algorithm (optimization algorithm) to the current Edgeward algorithm to quantify performance measures and determine which one is the best in terms of latency and energy consumption etc.

**Latency = Time from device to cloud +Time of data analysis + Time from cloud to device.**

****

**Inter-operability between IoT, Fog and Cloud computing**

Fog computer resource management is very complex as it involves a significant number of various fog sites and resources to meet the computer needs of IoT-enabled systems in a distributed way. Its integration with clouds creates other problems in the management of integrated resources. The different sensitivity of IoT devices, app sharing and communication also influence the management of resources in the Fog computer environment. Advancing Fog and its resource management, the need for more in-depth investigation outside question. To grow and validate different perspectives and resource management policies, a strong study of the fog environment is key. As the Fog computer atmosphere includes IoT devices, Fog nodes and Cloud data centers as well as a large number of IoT-data and distributed applications, the actual global implementation of the research fog environment will be more expensive. Moreover, the transformation of any business into a real fog atmosphere can be tedious. In this case, the simulation of the Fog computer atmosphere can be very supportive. Simulation tools not only provide a framework for designing a customized test environment but also help with repetitive testing. There are several simulators like Edge CloudSim, Simple IoT Simulator and iFogSim for modeling Fog computer environment and active testing.

As every approach has its benefits and opposite to it also possible that is most common. The things Iot overcome the disadvantage that an approach has in it.

**Benefits of Fog Computing:**

Fog Computing is a supplement to the conventional cloud-based platform since certain functions are best performed in the cloud and others are better performed in the Fog Computing platform. Here are some situations in which one model is more appropriate than the other:

* Time-sensitive applications are best hosted and executed on the Fog computing platform. In such applications, sensor data is processed and promptly managed and evaluated, allowing for timely decision-making and any necessary remedial steps.
* The system executive method in Fog computing supports both the application under development and the cloud platform. The cloud's ability to keep track of a large number of physical devices participating in the IoT model is harmed because system management is performed locally.

### Big data, generated by a huge number of smart devices, are better facilitated on the cloud platform at a time when these tools require powerful computation and capacity capabilities to advance software such as automaton learning algorithms.

### RELATED WORKS:

M. Taneja and A. Davy, et.al [2], Proposed 3 integration algorithms, Algorithm1: Module Mapping Algorithm-Fog-Cloud Placement, Algorithm2: Lower Bound Algorithm - Algorithm used for Search, Algorithm3: Compare Network Node and Application Module. Harshit Gupta, et.al, (2019). [1] Evaluated efficiencies of the two placement strategies, Cloud-Only and Edge-ward in terms of latency, network usage, and energy consumption for each case study.

Salaht et al. [3] delivered a list of optimization techniques: Mono vs. Multi-objective optimization, constrained optimization, to address resource management and placement problems. Taneja, M.; Davy, et.al. [4] Proposed an iterative method for resource deployment of IoT applications in a cloud–fog computing. This method is composed of three algorithms. The first algorithm sorts according to their requirements and capacity. The second algorithm looks for an eligible network mode that meets the module’s requirement. The last algorithm is responsible for ensuring the requirement check, which is done by using the COMPARE function.

Herman Meier, et.al. [5] Proposed algorithms for cloud computing which include the scheduling and migration on different Iot devices. Li, Bo, et al. [6] proposed energy-aware heuristic algorithm that dynamically migrates Virtual Machines between the hosts. The algorithm tries to maximize hosts utilization so that more hosts can be turned off to save energy. Deng, Ruilong, et.al. [7], discusses on power consumption and delay trade off in fog computing. They created a model with a fog and a cloud layer, which are connected over a WAN. Famaey, et al. (2009). [8] Proposed a dynamic and latency-aware distributed service placement policy over multiple homogeneous servers. The services can be assigned with a priority value based on their utility. Ottenwalder et al. (2013). [9] Proposed a plan-based operator placement and migration policy for Mobile Complex Event Processing (MCEP) applications to reduce network overhead and end-to-end delay.

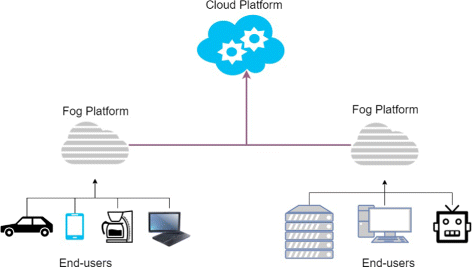
### ARCHITECTURE:

Fog Computing is a wireless distributed computing network that can manage complex latency affectability tasks using a group of sharing resources at the IoT entrance stage in a neighborhood. Fog computing, or cloud-to-device time, is a horizontal system architecture that distributes computation, storage, management, and organizing capabilities closer to users on a cloud-to-device time. These two concepts show some fundamental issues with the Fog Computing model's process and architecture.

Firstly, computation and storage capabilities are sent over a network of IoT devices located near the system layer.

Secondly, the need to optimize the handling and analysis time of collected data engaged space inside the cloud platform guided the increase in FC. As a consequence, an actual time-response and decision-making process is realized. The fog nodes in the Fog Computing model are the computing, storage, and networking elements.

Finally, the fog computing model sits in the middle of the system layer and the cloud. A high-level structural style of the Fog Computing model is depicted in the diagram below. It does demonstrate, however, that a series of disparate IoT devices can interact with the cloud platform using fog computing.



# OBJECTIVE OF STUDY:

This paper's solution to the problem discusses the following:

1. To discuss the fact that Genetic and Edgeward placement algorithms are the best possible solutions to the data processing issue in IoT.
2. The latency-sensitive requirements of upcoming and future IoT applications, as well as the Fog-Cloud architecture's deployment strategy.
3. Efficient resource management in network infrastructure.
4. To be verified that in fog computing Data processing is done more securely. As data is evaluated locally rather than send to the cloud this increases the security.
5. To address that the data processing time and cost will be minimum in Genetic and Edge computing and saves a lot of band-width along with time.
6. To address that both the Fog and cloud computing paradigm works as a combined way to address the resources.

## PROPOSED WORKS:

**Intelligent Monitoring Using Distributed Camera Networks:**

For elaborating the flexibility of iFogSim, The Intelligent Surveillance cameras are kept on several physical configurations, The number of physical surveillance has been varied from 1-16, each surveillance area has 4 smart cameras to monitor the area and each surveillance has 4 smart cameras connected to area gateway, this gateway which is responsible for providing the internet access where the number of surveillance areas is varied across configurations [confg1,confg2,confg3,confi4] having [1,2,3,4] areas respectively. The network latencies between devices are:

|  |  |  |
| --- | --- | --- |
| **SOURCE** | **DESTINATION** | **LATENCY(ms)** |
| CAMERA | AREA SWITCH | 2 |
| AREA GW | ISP GATEWAY | 2 |
| ISP GATEWAY | CLOUD DC | 100 |

Based on configurations of entities a physical topology is designed where the topology has a cloud data centre at the cloud and smart cameras at the edge. Smart cameras provide live video streams for performing motion detection. Two placement strategies that include cloud-only and edge-ward are used for placing application modules on physical network wherein cloud-only all operators in applications are placed in cloud data centre whereas in edge-only all operators are pushed to wifi gateways connected through surveillance are to internet. Now we will see three algorithms Cloud-Only, Edgeward and Genetic to compare the efficient algorithm for less energy consumption, latency and average control loop.

**EDGEWARD PLACEMENT ALGORITHM:**

Daniel Maniglia Amancio da Silva et.al [1], Edgeward is based on the "FCFS" strategy, which involves putting data on fog nodes as near to the network's edge as possible. This is a placement technique that solely relies on Edge. Edgeward selects other fog devices if one of the fog nodes is unable to meet the specifications of any of the applications. This Edgeward algorithm generates a large number of system tuples to describe the paths taken by application modules. Since this algorithm is based on FCFS, application requests are processed in the order they arrive until enough computational resources are available at each hierarchical stage.

**WORKING OF EDGEWARD PLACEMENT ALGORITHM:**

This algorithm shows the interplay between the fog and the cloud by the placement of modules both at the network edge and in the cloud.

1. Traverse across all the paths until the installation of application modules are placed close to the edge of the network.

2. Create a placelist array to add required modules needed for devices in this device list.

3. Starting from leaf to root, traverse if the current modules are needed for the application module to be placed.

4. If needed then check if all the predecessors of the module are placed and then add the module to the placelist.

5. Now placelist contains all the modules that are required for the devices, so traverse until the selected module belongs to the placelist and once the selected module belong to the placelist then,

6. Check if this selected module already placed in the device ‘f’, if placed,

7. Then merge the selected module with its upstream instance and store the device holding the merged instance of the device in a variable ‘f’. And traverse until the selected module searches for a Fog device with capacity at the top layer of the network topology hierarchy. And place the fog node at that device ‘f’.

8. Else if the selected module is not placed in the nearest device‘d’, then check the computing capacity with the nearest device and place the module in the device ‘d’ itself at the first level of hierarchy.

**ALGORITHM: EDGEWARD-PLACEMENT:**

\* While p belongs to PATHS do // Iterate through all of the paths (p: path)

\* PlaceTheModules = { }; // Create a list for system

\* While Fog-system s belongs to p do {} // Traversal from leaf to root (s: system)

\* ModulesToBePlace = { };

\* While module w belongs to app do // locate the modules that are ready to be installed on system s

\* If all the predecessors of w are in PlaceTheModules then // (w: modules)

\* Add w to ModulesToBePlace

\* End of if

\* End Of while

\* While module t belongs to ModulesToBePlace do

\* If s has instance of t as t ′ then // (t: selected module)

\* If CPUt >= CPUs then // Check to see if system s has enough CPU power to host t.

\* t” = merge (t, t ′)

\* f = parent (s)

\* While CPUt” >= CPUf do // Find a system for hosting t near the cloud of s

\* f = parent (f)

\* End Of While

\* Place t” on system f //system can host t”

\* Add t to PlaceTheModules

\* Else if

\* Place t on system s // system s can host t

\* Add t to PlaceTheModules then

\* End of if

\* Else if no system near the cloud of s has an instance t then

\* If CPUt <= CPUs then // If not, subsequent iterations will manage it.

\* Place t on system s

\* Add t to PlaceTheModules

\* End of if

\* End of if

\* End Of while

\* End Of while

**CLOUD ONLY PLACEMENT ALGORITHM:**

In Harshit Gupta, et.al [2], the application modules are assumed to operate in data centers in the cloud-only placement. This algorithm is based on a scenario in which sensors collect data, the data is processed in the cloud, and the cloud sends the information to the actuators if necessary.

**WORKING OF CLOUD-ONLY-PLACEMENT ALGORITHM:**

A fog system may be scheduled in the cloud if it fails to fulfil the specifications of an application module.

1. Traverse across all the paths until the installation of application modules are placed in cloud of the network adds the required modules to the placelist.

2. And if selected module belongs to placelist then add the selected module directly to the cloud.

**ALGORITHM: CLOUD ONLY PLACEMENT:**

\* While p belongs to PATHS do // Iterate through all of the paths (p: path)

\* PlaceTheModules = { } // system list

\* While Fog system s belongs to p do // (s: system)

\* While module w belongs to app do // (w: modules)

\* If all predecessors of w are placed then

\* Add w to PlaceTheModules

\* End of if

\* End of while

\* While module t belongs to PlaceTheModules do

\* Place t on system CLOUD // (t: selected module)

\* End of while

\* End of while

\* End of while

**GENETIC ALGORITHM:**

Genetic algorithm is and optimization problem that can be used to choose the best fog node among the available modules to be placed at the parent node. A Genetic algorithms involves mainly four functions

1) Fitness Function

2) Selection Function

3) Crossover Function

4) Mutation Function

Fitness Function: It is to evaluate new modules selected from the system of available modules. Assign a fitness score to each node, which is the value of the Fitness function for that solution of the optimization problem.

Selection Function: It is to select the modules to generate the next generation.

Crossover Function: Cut the chromosome and mix it up /change over.

Mutation Function: Take single parental chromosome modify one single gene in a chromosome.

Chromosome

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F1 | F4 | F8 | F1 | F3 | F6 |

One particular cell is called as gene

M1 M2 M3 M4 M5 M6

Steps that involve in genetic algorithm:

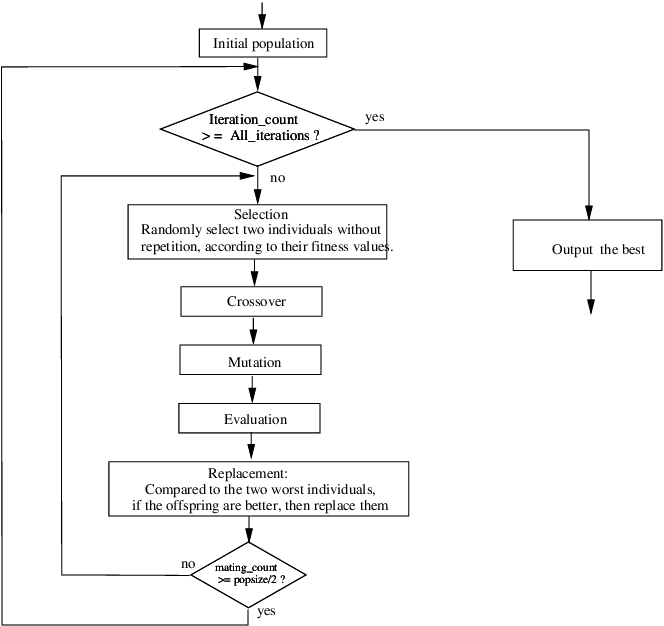
1. Choose the Configuration size
2. Randomly choose Initial module
3. Select parental chromosome
4. Crossover and Mutation
5. Evaluation of the offspring
6. If stopping criteria not reached go to step 3 again,

And perform up to size no of times.

Termination or stopping condition may be either:

i) Best module is found from set of modules to be placed

ii) If reached ‘n’ number of configurations



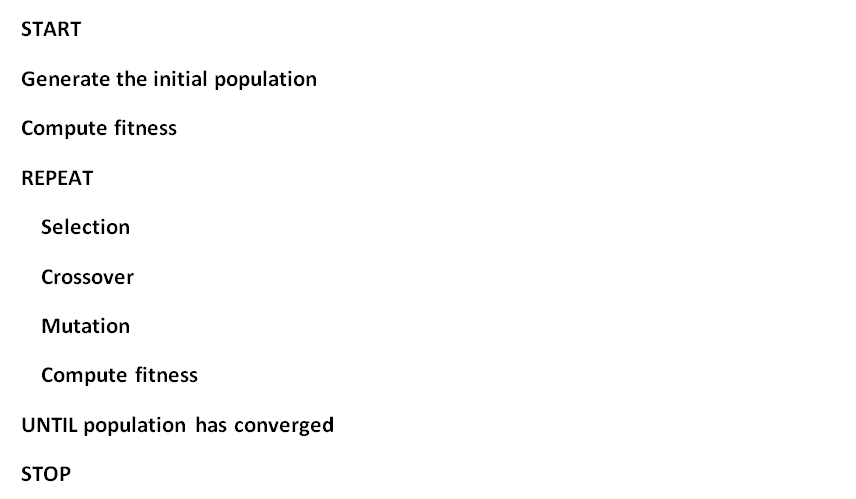
**NEED OF GENETIC ALGORITHM:**

To find a suitable module combination that provides low latency, compare the time it takes to compute the result. Finding the most effective module to place can be formulated as an optimization problem with an objective function in the case of complex systems with a large number of measurable variables; similarly, finding the most efficient module to place can be formulated as an optimization problem with an objective function in the case of complex systems with a large number of measurable variables. Genetic algorithms can evolve a final population that can store a variety of individuals with the same fitness value.

This type of algorithm begins from an initial population of candidates and uses genetic operators such as crossover, mutation, and selection to try to emulate natural selection laws and simulate biological evolution, creating new populations of better individuals at each iterative stage. The algorithm finds the optimum solution to the problem as the best-fit individual after many iterations, which vary depending on the problem's complexity.

An individual is marked by a code called a chromosome, which is mapped to a specific value of the objective feature that represents the individual's fitness. The algorithm uses genetic operators to create new individuals and pick the fittest ones at each level to increase the fitness of the each individual.

**PSEUDOCODE:**



**ALGORITHM: GENETIC ALGORITHM FOR MODULE PLACING**

**Input:** size (n) of population,

No of genes or configuration (b),

Rate (y) of mutations,

Number of (t) iterations,

**Output:**

//Initialization

\* Generate **n feasible solutions randomly;**

\* Save **them in population Pop;**

//Loop until the terminal condition is met

\* **for i=1 to t do**

//b number of selection

\* Number of selection ne=n\*b;

\* select the best ne solutions in Pop and save them in Pop1;

//Do the Crossover

\* number of crossover nc= (n-ne)/2;

\* **for** j=1 to nc **do**

\* randomly select two solutions XA and XB from Pop;

\* generate XC and XD by one point crossover to XA and XB;

\* save XC and XD to Pop2;

\* **End of for**

//Mutation

\* **for** j=1 to **nc** do

\* select a solution Xj from Pop2;

\* mutate each bit of Xj under the rate y and generate new solution Xj’;

\* **if** Xj’ is unfeasible

\* update Xj’ with feasible solution by reparing Xj’;

\* **End of if**

\* update Xj with Xj’ in Pop2;

\* **End of for**

//update

\* update Pop=Pop1+Pop2;

\* **End of for**

//Return the best solution

\* return the best solution X in Pop;

### RESULTS:

Compare the results of (i) Total Energy Consumption. (ii) Total Network Usage over,

1. Cloud-Only-Placement algorithm

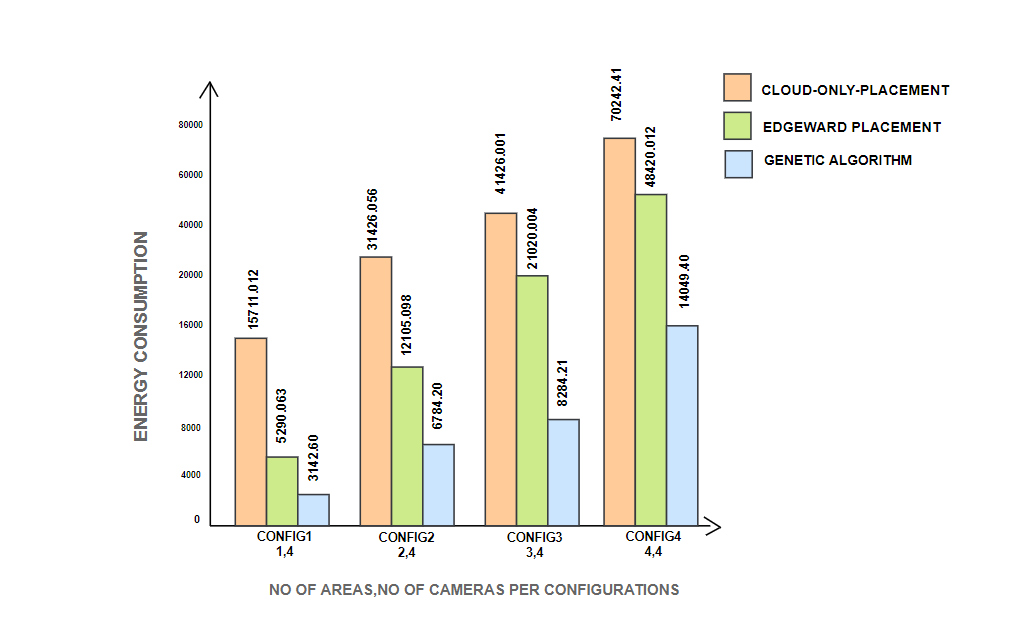
2. Edgeward Placement algorithm

3. Genetic algorithm

The below **Table** consists of the comparison of Total Energy Consumption, Total network Usage over 3 different algorithms i.e. Cloud-only-placement, Edgeward-placement and Genetic algorithm.

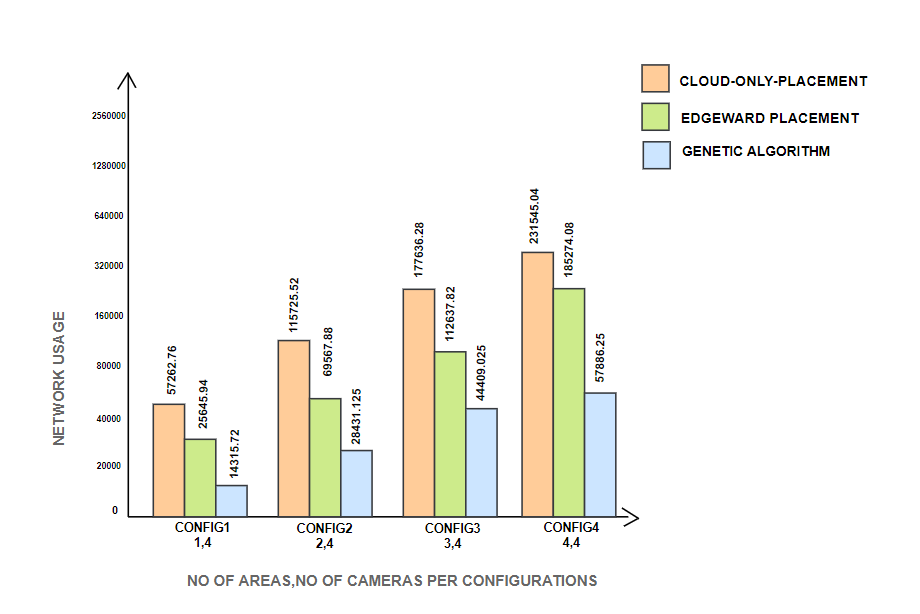
|  |  |  |  |
| --- | --- | --- | --- |
| **No. of cameras=4**  **No. of areas=i, where**  **i=1,2,3,4** | **CLOUD-ONLY-PLACEMENT ALGORITHM** | **EDGEWARD PLACEMENT ALGORITHM** | **GENETIC ALGORITHM** |
| **Total Energy Consumed**  **(No. of areas =1)** | 15711.012 | 5290.063 | 3142.60 |
| **Total Network usage**  **(No. of areas =1)** | 57262.76 | 25645.94 | 14315.72 |
| **Total Energy Consumed**  **(No. of areas =2)** | 31426.056 | 12105.098 | 6284.2 |
| **Total Network usage**  **(No. of areas =2)** | 115725.52 | 69567.88 | 28431.125 |
| **Total Energy Consumed**  **(No. of areas =3)** | 41426.001 | 21020.004 | 8284.21 |
| **Total Network usage**  **(No. of areas =3)** | 177636.28 | 112637.82 | 44409.025 |
| **Total Energy Consumed**  **(No. of areas =4)** | 70242.41 | 48420.012 | 14049.40 |
| **Total Network usage**  **(No. of areas =4)** | 231545.04 | 185274.08 | 57886.25 |

The Figure1 shows the comparison of Total Energy Consumption in 4 different configurations over three different algorithms.



**FIGURE 1**

The Figure2 shows the comparison of Total Network Usage in 4 different configurations over three different algorithms.

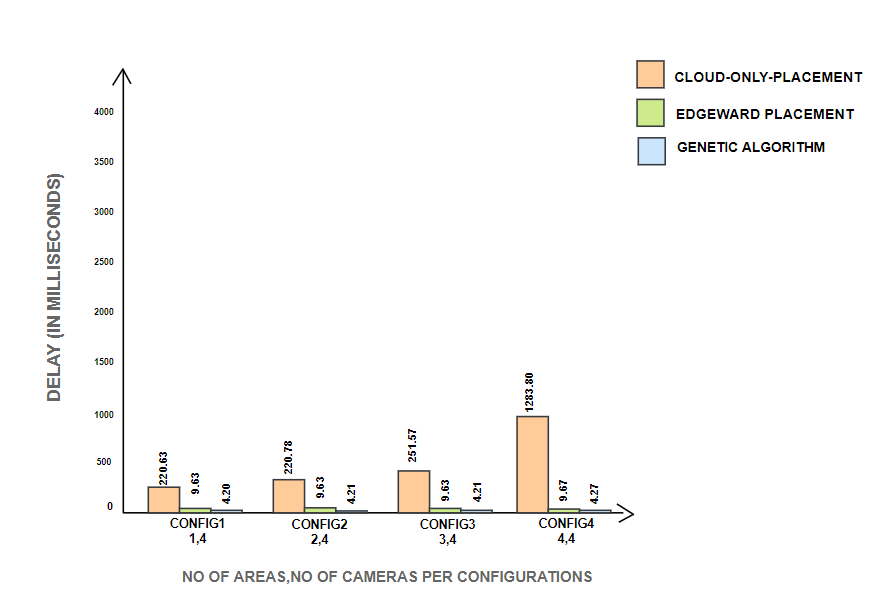
****

**FIGURE 2**

This below **Table** consists of comparison of Control-Loop-Delay over 3 different algorithms i.e. Cloud-only-placement, Edgeward-placement and Genetic algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
| **No. of cameras=4**  **No. of areas=i, where**  **i=1,2,3,4** | **CLOUD-ONLY-PLACEMENT ALGORITHM** | **EDGEWARD PLACEMENT ALGORITHM** | **GENETIC ALGORITHM** |
| **CONTROL LOOP DELAY (in ms)**  **(No. of areas= 1)** | 220.63 | 9.63 | 4.20 |
| **CONTROL LOOP DELAY (in ms)**  **(No. of areas= 2)** | 220.78 | 9.63 | 4.21 |
| **CONTROL LOOP DELAY (in ms)**  **(No. of areas= 2)** | 251.57 | 9.63 | 4.21 |
| **CONTROL LOOP DELAY (in ms)**  **(No. of areas= 4)** | 1283.80 | 9.67 | 4.27 |

The Figure3 shows the comparison of Control-Loop-Delay in 4 different configurations over three different algorithms.

****

**FIGURE 3**

**Energy Consumption:** In Figure 1 we observe that Genetic algorithm performs best when compared to Cloud-Only-Placement and Edgeward Algorithm. Genetic algorithm takes 80% less Energy consumption than the Cloud-only-placement and 40% less Energy Consumption than the Edgeward placement algorithm.

* Which means that only 20% percent of Cloud-only-placement energy consumption is consumed by the Genetic algorithm and 60% of Edgeward placement energy consumption is consumed by the Genetic algorithm

**Network Usage:** Figure 2 depicts the network use of the Surveillance application for placement strategies; as the number of devices connected to the application grows, the load on the network grows automatically in the case of the cloud only, Edgeward-based. While in the Genetic execution, the majority of data-intensive communication takes place over low latency links, reducing the volume of data sent to the cloud.

* We can observe that Genetic algorithms Network usage is 75% less when compared to Cloud-only-placement and 44% less when compared to the Edgeward placement algorithm.

**The Average Latency of the Control Loop:** Figure 3 depicts the average processing latency of the control loop's sensing-actuation. Cloud data centres became a bottleneck in the execution of modules in a cloud-only placement strategy, resulting in a significant increase in latency. Edge-ward placement, on the other hand, achieves low latency by placing the control loop's vital modules near to the network edge. When compared to Edgeward placement, genetic placement has achieved even less latency when compared to Edgeward placement algorithm.

* We can observe that Genetic algorithms Average latency of Control loop is 98% less when compared to Cloud-only-placement and 57% less when compared to Edgeward placement algorithm.

**CONCLUSION:**

In this paper, we defined the efficiency differences between the current cloud algorithms, Edgeward algorithm and Genetic algorithm. As opposed to Cloud-only and Edgeward placement algorithms, we discovered that the “Genetic algorithm” produces more efficient and optimal performance. We demonstrated the effectiveness of the new fog computing model in addressing the most challenging challenges, such as energy usage, latency, and finishing work on schedule for time-sensitive operations, all of which are critical for IoT applications.

## We tested different examples, such as camera networks; with both existing algorithms and the Genetic algorithms we are developing to demonstrate that they are more efficient than the current algorithms. We've provided a comparative comparison based on latency, energy use, and network use.

## Based on the results of the preceding case studies, we infer that the Genetic model influences the outcomes of IoT applications. It was designed with many use cases in mind, and it can be applied to a wide range of Iot devices.

## FUTURE WORK:

We want to propose novel algorithms in the future to increase the Iot applications performance and offer a new IoT resolution.

## REFERENCES:

[1]. Daniel Maniglia Amancio da Silva 1 , Godwin Asamooning1 , Hector Orrillo1 , Rute C. Sofia2 , and Paulo M. Mendes “An Analysis of Fog Computing Data Placement Algorithms”.

[2]. H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, “ifogsim: Atoolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments,” Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, 2017.

[3] Salaht, F.A.; Desprez, F.; Lebre, A. An Overview of Service Placement Problem in Fog and Edge Computing. ACM Comput. Surv.2020, 53, 1–35.

[4] Taneja, M.; Davy, A. Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017;pp. 1222–1228

[5] Herman Meier.; “Simulating energy efficient fog computing”, Bachelor’s Thesis (9 ECTS).

[6] Li, Bo, et al. "Enacloud: An energy-saving application live placement approach for cloud computing environments." 2009 IEEE International Conference on Cloud Computing. IEEE, 2009.

[7] Deng, Ruilong, et al. "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing." 2015 IEEE International Conference on Communications (ICC). IEEE, 2015.

[8] Famaey, et al. (2009). “Dynamic and latency-aware distributed service placement policy over multiple homogeneous servers”.IEEE, 2009.

[9] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran. 2013. MigCEP: Operator migration for mobility driven distributed complex event processing. In Proceedings of the 7th ACM International Conference on Distributed Event-based Systems (DEBS’13). ACM, New York, NY, USA, 183–194.

[10] alheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 2011.