VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Time and Energy-Efficient task offloading and Scheduling Algorithm in a 3 tier Fog Architecture

The Project report as part of Fundamentals of Fog and Edge Computing(BCT3005)

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology
In
COMPUTER SCIENCE AND ENGINEERING
SCOPE

Prepared By:

1.K.Harish Gokul-19BCT0049

2.Shiv Sai Indrakanti-19BCT0008

3.Ramakrishnan J-19BCT0057

Under the guidance of

Prof. Umadevi K S
School of Computer Science and Engineering VIT, Vellore.

1. Introduction to Domain

As the complexity of the applications we use on mobile devices increases, a constraint arises due to this, which is the computational capacity of these devices. We have resource intensive applications like augmented reality which have real time constraints or near real time constraints. Mobile devices are hence unable to execute media rich tasks and tasks like data analysis involving a high computational threshold.

One way to navigate through this issue of Mobile devices in comparison to their non-mobile counterparts is the use of Cloud Computing. Cloud computing technology is provided by Cloud Service Providers like Amazon, Rackspace and many more. They provide the operational backbone to execute these tasks on behalf of the mobile devices. The cloud provides the necessary resources and infrastructure for the tasks to be executed. The concept of handing off resource intensive tasks from a mobile device to resource rich infrastructure is called Offloading. Thus it may seem like the concept of Cloud and the process of offloading tasks can alleviate the problem discussed above. This may indeed be a viable solution but we have to factor in key components like time and distance into this equation. Offloading a task involves handing off data to the cloud and receiving the processed data back from the cloud. The key issue however here is that Cloud Resources are located far away from these mobile devices and hence there is excessive latency.

Thus we go for an architecture where there is a layer between the mobile IoT devices and our Cloud layer in order to reduce this latency and offload the tasks to this layer. This is called the Fog layer or sometimes referred to as Edge devices. The compute process which takes place at this layer can be termed as MEC or Mobile Edge Computing. The simplest fog architecture is the 3 layer architecture which is elucidated as follows:

1. **IoT Layer (L1):** IoT device layer generates application tasks. These devices have a limited processing capability, limited computational capacity and operate with comparatively low-energy batteries.

2. **MEC Layer (L2):** MEC server layer has a restricted number of CPUs and less processing capability than the CC environment but more computational capacity than the IoT Layer. MEC servers are closer to the IoT devices, producing smaller communication delays.

3. **CC Layer (L3):** CC Data Centers compose this layer. These servers have high processing capability, are geographically distributed, and are far located from the IoT devices. They also add high network latency due to data transmission with more communication hops compared to other layers.

Now that we have elucidated the three layer architecture, we carry out a process called offloading. There are different types of offloading but the type we are concerned about here in our project is called computational offloading and data offloading.

The task of offloading computation-intensive application components to a remote server is known as computation offloading. A number of computation offloading frameworks for mobile applications have recently been presented, with a variety of techniques. To extend and strengthen the SMD's capabilities, these applications are partitioned at different granularity levels, and the components are transmitted (offloaded) to remote servers for remote execution.

Offloading computations is clearly beneficial only when the local execution (on a mobile device) takes longer and consumes more energy than the offloading overhead. Many factors can influence the decision to offload and the process of offloading. These factors include the battery constraints of mobile IoT cores, the complexity of the application which is executed , the fraction of critical tasks needed to be completed within a hard deadline, security of data when transmitted for remote execution. When the path of data transfer is longer, then the susceptibility for man in the middle attacks increases and thus this gives us one more reason to introduce the intermediate fog layer.

We calculate the power and energy using the capacitance of the CPU cores present in each layer along with the operating frequency of these CPU cores. A technique called DVFS is used here in order to calculate this information. DVFS stands for Dynamic Voltage and Frequency Scaling.We can adjust the CPU frequency dynamically in order to reduce energy or time.This is the tradeoff we encounter.When we increase the CPU frequency the time of execution is less but our energy and power consumed is high.When we decrease CPU frequency the time of execution is more but our energy and power consumed is less.There is a tradeoff of choosing and adjusting an ideal frequency for time-energy tradeoff.

Thus DVFS and a scheduling algorithm based on a cost model can help ensure execution of tasks optimally by consuming less energy. The cost model indicates what layer to be chosen and gets frequency and voltage information using DVFS and the DVFS provides the best frequency-voltage pair for minimum cost

2. Literature Survey

i)Energy-efficient offloading and resource allocation for mobile edge computing enabled mission-critical internet-of-things systems

Authored By: Yaru Fu, Xiaolong Yang, Peng Yang, Angus K. Y. Wong, Zheng Shi, HongWang & Tony Q. S. Quek

Published in: 2021

The energy cost minimization for mission-critical internet-of-things (IoT) in mobile edge computing (MEC) system is investigated in this work. The energy cost minimization challenge for short packet transmission for mission-critical IoT in MEC systems was formulated in this research, and the effect of short packet transmission on radio resource management was shown. The minimization problem is a mixed integer non-linear programming (MINLP) problem, and finding the best solution is difficult. The challenge stems mostly from the intertwining of offloading technique and resource optimization.

ii)Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling
Authors: by T. Q. Dinh, J. Tang, Q. D. La and T. Q. S. Quek.

This paper illustrates the combined use of Computational offloading and DVFS mechanism in order to optimize the cost and energy consumed by a system. Past authors have usually taken the mathematical approach of optimization by adopting various cost aware, delay aware optimization schemes, but here authors have provided insight into how the CPU core frequency can be adjusted so that the power consumed is less while also ensuring that the offloading scheme allocates task to a layer where DVFS ensures minimum cost. This forms one of the basis for this project where we use DVFS to calculate frequency and schedule or allocate tasks to different layers discussed as above in our 3 layer fog architecture.

iii)Energy efficient scheduling in IoT networks

Authors: Smruti R.Sarangi, Sakshi Goel and Bhumika Singh

This paper highlights the importance of limiting energy consumption in IoT networks due to the lack of accessible power which is usually unreliable.Two algorithms are proposed in this paper where one exchanges information between neighboring nodes and the other uses a global server that uses a snapshot of the global state of the network.The paper also theorized that global approach is preferred for hard real time and the former for soft real time applications.

iv)Towards effective offloading mechanisms in fog computing

Authors: Maryam Sheikh Sofla, Mostafa Haghi Kashani, Ebrahim Mahdipour & Reza Faghih Mirzaee
This report presents a thorough and complete analysis of existing and recent fog dumping strategies.The study divided fog offloading mechanisms into four categories: computation-based, energy-based, storage-based, and hybrid approaches. This study also looks at offloading measurements, application algorithms, and evaluation methodologies for fog systems' chosen offloading strategies.Fog computing can be used to extend cloud computing's processing and storage capabilities by acting as a middle layer between consumer devices and the cloud.

v)Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities

Authors: Mohammad Aazam, Khaled A.Harras

As applications get more sophisticated and intelligent, we must manage the execution of more complicated tasks efficiently based on the application's requirements. The resource needs of some of these tasks may be much beyond the capacity of the end-device. user's. The duties are offloaded to the middleware in certain cases, which may then transfer them to the cloud.

When tasks are offloaded in the fog computing environment, the study discusses the various criteria employed by recently suggested middleware systems.They discuss the primary technologies that are currently being used in fog computing to enable offloading, as well as some frequent scenarios in which offloading may occur.In fog computing, they outline future research issues that must be addressed in order to improve task offloading performance, efficiency, and reliability.

3. Proposed Work

We propose a Time and energy efficient scheduler which allocates tasks from a particular application based on characteristics like the energy consumer, total time taken, nature of the task whether it is a critical or non critical task and its respective deadline. The task of finding minimal cost and optimization is a NP hard problem due to the following reasons:

1. A single model cost and energy optimization problem is not a NP hard problem but modeling our cost based on just these 2 factors would not be sufficient. We need to take into consideration factors which contribute to time and energy like the energy consumed during data transmission, the energy consumed for data download, the time taken for data transfer and the objective of minimizing these variables for each individual layer i.e IOT, FOG and cloud layer. Thus this is a multi-objective optimization problem.
2. We need data at a granular level and a method of calculating these parameters at each and every level. Only then a comprehensive analysis of the energy and time factor can be feasible for developing a real life solution. Thus we need a simulation procedure which provides us the tools to simulate and collect data at each and every step.

To Tackle the first problem, we develop a dynamic cost optimization model wherein there is a tradeoff between the energy consumed and the time taken for execution. This is done by calculating the cost as the weighted sum of the energy and time. This process is called Integer Linear Programming. We assign a weight of 0.8 to the energy parameter and a weight of 0.2 to our time parameter. Thus giving greater importance to energy consumed vs the time taken. The calculation of these costs however is done for each and every layer with considerations for the transmission cost, energy etc. This is done using a special tool.

The tool ,we used is the IFOG sim Tool to develop and construct a three layer architecture wherein we have access to application data at a granular level.Each application is broken down into granular threads or tasks and each application is divided into fundamental

characteristics.The parameters related to the application will be elucidated under the same section under sub section Application Data.

Here are the parameters considered by the project team in building the scheduler.

1. D: Set of IOT Devices
2. S: Set of MEC Server
3. r: Data transfer rate
4. d: input data
5. f: CPU operating frequency
6. sc: Source Code Size
7. A: Task set that will be executed
8. i: Individual Task
9. W: Bandwidth Associated with a channel
10. C_c: CPU Cycle for a core
11. T: Total Execution Time
12. $T_{i\text{-}mec}$: Total Execution time in the MEC Server
13. $T_{i\text{-}local}$: Total Execution time in the IoT device
14. $P_{i,mec}$: The power consumed in the MEC server
15. $E_{i,mec}$:The dynamic energy consumed in the MEC server
16. Cost $_{i,mec}$: The total cost in the MEC server

Model Description

The cost is the weighted sum of the net energy and net time taken and we can define this variable as:

Cost-local=A*(Total time taken locally)+ B*(Energy consumed locally)

Where A=0.2 and B=0.8.

To compute processing costs, the DVFS method is employed, which identifies the optimal pair of CPU cores for lowering total costs.We then analyze the local cost for each allocation layer using our scheduler which would be our IOT layer, Fog layer and cloud layer. Here for a particular task t_i we consider the cost allocation at all 3 levels. This would entail the calculation of cost of IOT devices which would just include the calculation of energy and time consumed without any data transmission, since no offloading is done when tasks are allocated at that level.

We the calculate the cost for the scheduler to allocate the same task at the MEC and cloud, where the net energy,time and cost would not only involve the local energy and time consumed by the CPU cores involved in the process, but also involve the data transmission cost for the data to move up the layers to emulate offloading and data to move down the layers emulating a download.

From the cost calculated for each layer we then calculate the minimum cost possible from the three , and allocate our task to that particular layer. This is the procedure we follow for every normal task.

This might not be the case for critical tasks. Here every application as a certain percentage of total tasks flagged as critical tasks with a hard deadline. Thus these tasks are given the higher priority of execution. The methodology of task execution would be that the critical tasks are executed first irrespective of their task size and other characteristics like.computational load CPU cycles, data entry size etc. The same policies of allocation and method are used to schedule the critical tasks but this time the cost is biased towards the time factor. Thus instead of a higher weight being given to the energy , all the priority is only given to the time of execution since we need to satisfy the hard deadline constraints.

Cost Model:

1.Local Compute Cost at the IoT Devices

The Calculations for the cost at the IoT layer can be expressed as follows:

$T_{i,local} = C_c/f\ local,j,k$

$P_{i,local} = Ci,_{local} * V^{\wedge 2}{}_{i,local} * f_{i,local}$

$E_{i,local} = P_{i,local} * T_{i,local}$

$Cost_{i,local} = u_{localT} * Ti,local,total + u_{localE} * Ei,local$

Here u_{localT} and u_{localE} are the priority weights assigned to each parameter i.e (time and energy) to represent a trade-off and minimize one of the costs.

2.Local Compute Cost at the MEC Server

This is the time required to offload the source code to the MEC layer from the device layer. The time of data transfer up the data lines is defined as.

1.$T_{i,mec-up} = sc_i + di/r_i$

This is the time required to download the processed input data.Thus the time of data transfer down the data lines is defined as.

2.$T_{i,mec-down} = d_i / r_i$

The total time of execution is the: offloading time + the time required for local processing + the total time of download

3.$T_{i,mec} = T_{i,mec-up} + T_{i,mec} + T_{i,mec-down}$

The total energy of execution is the: offloading energy + the energy required for local processing + the total energy of download

4.$E_{i,mec} = E_{i,mec-up} + E_{i,mec} + E_{i,mec-down}$

5.$Cost_{i,mec} = u_{mecT} * T_{i,mec} + u_{mecE} * E_{i,mec}$

Here u_{localT} and u_{localE} are the priority weights assigned to each parameter i.e (time and energy) to represent a trade-off and minimize one of the costs.

3.Remote compute Cost at the Cloud

This is the time required to offload the source code to the Cloud layer from the device layer. The time of data transfer up the data lines is defined as.

1.$T_{i,cloud-up} = s_i + d_i/r$

This is the time required to download the processed input data.Thus the time of data transfer down the data lines is defined as.

2.$T_{i,cloud-down} = d_i / r$

3.$E_{i,cloud-up} = p_{wireless} * T_{i,cloud-up}$

4.$E_{i,cloud-down} = p_{wireless} * T_{i,cloud-down}$

The total time of execution is the: offloading time to the MEC+offloading time to the Cloud + the time required for local processing + the total time of download from MEC + total time to download from the Cloud

5.$T_{i,cloud,total} = T_{i,mec-up} + T_{i,cloud-up} + T_{i,cloud} + T_{i,cloud-down} + T_{i,mec-down}$

The total energy of execution is the: offloading energy to the MEC+offloading energy to the Cloud + the energy required for local processing + the total energy of download from MEC + total energy to download from the Cloud

6.$E_{i,cloud,total} = E_{i,mec-up} + E_{i,cloud-up} + E_{i,cloud+} E_{i,cloud-down} + E_{i,mec-down}$

7.$Cost_{i,cloud} = u_{cloud}T * T_{i,cloud} + u_{cloudE} * E_{i,cloud}$

Here u_{localT} and u_{localE} are the priority weights assigned to each parameter i.e (time and energy) to represent a trade-off and minimize one of the costs.

Cost Minimization:

$Cost_i = \min(Costi,local, Costi,mec, Costi,cloud)$

Total Cost:

Cost System = $A \sum i=1 Cost_i + E_{local,idle} + E_{mec,idle}$

Scheduler Algorithm:

The Scheduling mechanism for the above cost model can be elucidated as below:

Task Allocation:

1.for each task Ai from list of **critical tasks** do

for each free CPU core do

policy 1: it calculates IoT device execution time

foreach free CPU core do

policy 2: it calculates MEC server execution time and transmission times

policy 3: it calculates CC execution time and transmission times

Evaluate IoT battery level and offload task to the CPU core with **minimal total time**

For each task Ai from list of **regular tasks** do
For each free CPU core do
policy 1: it calculates energy consumption, execution time and cost for the IoT device
For each free CPU core do
policy 2: it calculates energy consumption for dynamic processing and data transmission
policy 2: it calculates execution time and transmission times
policy 2: it calculates MEC server cost

policy 3: it calculates energy consumption for dynamic processing and data transmission
policy 3: it calculates execution time and transmission times
policy 3: it calculates CC total cost

Evaluate IoT battery level and offload task to the CPU core with **minimal cost**
The Time Complexity of this Algorithm: O(Scheduler) = $O(n^2)$
Application Data:

We have decided to split the application into its fundamental characterstics, to get a granular look at the energy and time. The characteristics considered are as follows:

1.Task Generation rate: Tasks generated per second.

2.Data Size Entry: Data Size Offloaded/ Needs to be processed

3.Result Size: Define the size of the results that a job produces once it has been processed.

4.Computational Load CPU Cycles: We assume that the user creates cyclical jobs and is aware of their CPU cycle count. The computational load is mostly used to calculate the execution time.

5.Deadline: Critical/ Non Critical Tasks: If a critical task is created, data transfers (data entry and results) and processing must be completed in less time than the deadline**.**

Application Considered here for Analysis:

1. Task Generation Rate: 0.1 * 10^6 Tasks/second
2. Data Entry Size: 4 MB
3. Result Size: 625 MB
4. Computational Load: 200 million cycles of CPU
5. Critical Task: 50%
6. Deadline for Critical Task: 100ms

4. Code

Scheduler:
https://drive.google.com/file/d/1z0sNuZaEF1jxg2aayICeQ-WjfkW7hIVi/view?usp=sharing
Simulator:
https://drive.google.com/file/d/15Lhmf1XRjAIK3T7G2Dn2KKkjo9x_Vy8N/view?usp=sharing
Entire Code for the Project along with the Output files used in the comparative analysis is attached as below:
https://drive.google.com/drive/folders/1e3ca0T6HY6v2Cod8j2U9YvJavX6noz-1?usp=sharing
Comparative Analysis:

```
#Cost Calculation for a single task
#iot_cost
cpu_core_energy_iot=0
transfer_energy_iot=0
cpu_core_time_iot=0
transfer_time_iot=0
cost_iot=0
#mec_cost
cpu_core_energy_mec=245025
transfer_energy_mec=140313
cpu_core_time_mec=266666
transfer_time_mec=32021
cost_mec=368008
#cloud_cost
cpu_core_energy_cloud=1245128
transfer_energy_cloud=257132
cpu_core_time_cloud=51282
transfer_time_cloud=64010
cost_cloud=1024221
#zero mec server
total_tasks=500
iot_tasks=0
mec_tasks=0
cloud_tasks=500
plt.xlabel('Policy Type')
plt.ylabel('Number of Tasks')
plt.title('Task Allocation Scheme using 0 MEC Server')
x=["1-IOT DEVICE","2-MEC SERVER","3-CLOUD"]
y=[iot_tasks,mec_tasks,cloud_tasks]
plt.bar(x,y)
```

```
plt.show()
p_pie=[iot_tasks/total_tasks,mec_tasks/total_tasks,cloud_tasks/total_tasks
]
plt.pie(p_pie,labels=['IOT','CLOUD','MEC'])

#total_execution_energy
t_energy=p_pie[0]*(cpu_core_energy_iot+transfer_energy_iot)+p_pie[1]*(cpu_
core_energy_mec+transfer_energy_mec)+p_pie[2]*(cpu_core_energy_cloud+trans
fer_energy_cloud)
print("The total Execution Energy with Zero MEC Servers: " +
str(t_energy))


#total_execution_time
t_time=p_pie[0]*(cpu_core_time_iot+transfer_time_iot)+p_pie[1]*(cpu_core_t
ime_mec+transfer_time_mec)+p_pie[2]*(cpu_core_time_cloud+transfer_time_clo
ud)
print("The total Execution time with Zero MEC Servers: "+ str(t_time))

#total_cost_time
t_cost=p_pie[0]*(cost_iot)+p_pie[1]*(cost_mec)+p_pie[2]*(cost_cloud)
print("The total Cost with zero MEC Servers: "+ str(t_cost))

#one mec server
total_tasks=500
iot_tasks=0
mec_tasks=40
cloud_tasks=460
plt.xlabel('Policy Type')
plt.ylabel('Number of Tasks')
plt.title('Task Allocation Scheme using 1 MEC Server')
x=["1-IOT DEVICE","2-MEC SERVER","3-CLOUD"]
y=[iot_tasks,mec_tasks,cloud_tasks]
plt.bar(x,y)
plt.show()
#total_execution_energy
t_energy=p_pie[0]*(cpu_core_energy_iot+transfer_energy_iot)+p_pie[1]*(cpu_
core_energy_mec+transfer_energy_mec)+p_pie[2]*(cpu_core_energy_cloud+trans
fer_energy_cloud)
print("The total Execution Energy with one MEC Servers: " + str(t_energy))
```

```
#total_execution_time
t_time=p_pie[0]*(cpu_core_time_iot+transfer_time_iot)+p_pie[1]*(cpu_core_t
ime_mec+transfer_time_mec)+p_pie[2]*(cpu_core_time_cloud+transfer_time_clo
ud)
print("The total Execution time with one MEC Servers: "+ str(t_time))

#total_cost_time
t_cost=p_pie[0]*(cost_iot)+p_pie[1]*(cost_mec)+p_pie[2]*(cost_cloud)
print("The total Cost with one MEC Servers: "+ str(t_cost))

#two mec server
total_tasks=500
iot_tasks=0
mec_tasks=80
cloud_tasks=420
plt.xlabel('Policy Type')
plt.ylabel('Number of Tasks')
plt.title('Task Allocation Scheme using 2 MEC Server')
x=["1-IOT DEVICE","2-MEC SERVER","3-CLOUD"]
y=[iot_tasks,mec_tasks,cloud_tasks]
plt.bar(x,y)
plt.show()

#total_execution_energy
t_energy=p_pie[0]*(cpu_core_energy_iot+transfer_energy_iot)+p_pie[1]*(cpu_
core_energy_mec+transfer_energy_mec)+p_pie[2]*(cpu_core_energy_cloud+trans
fer_energy_cloud)
print("The total Execution Energy with two MEC Servers: " + str(t_energy))


#total_execution_time
t_time=p_pie[0]*(cpu_core_time_iot+transfer_time_iot)+p_pie[1]*(cpu_core_t
ime_mec+transfer_time_mec)+p_pie[2]*(cpu_core_time_cloud+transfer_time_clo
ud)
print("The total Execution time with two MEC Servers: "+ str(t_time))

#total_cost_time
t_cost=p_pie[0]*(cost_iot)+p_pie[1]*(cost_mec)+p_pie[2]*(cost_cloud)
```

```
print("The total Cost with 2 MEC Servers: "+ str(t_cost))
```

https://colab.research.google.com/drive/1XjSKBFSLZgj1kczQuYg_iadKzaL_eWeC#scrollTo=Yb7f90om2pY6

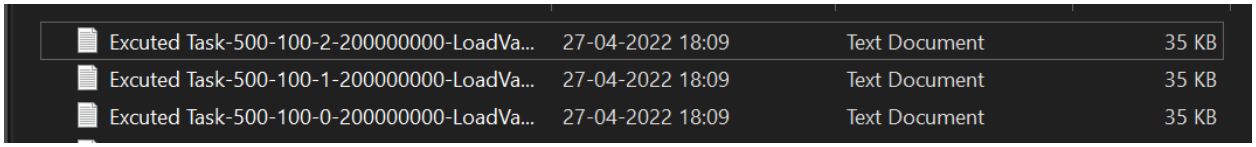
5. Analysis

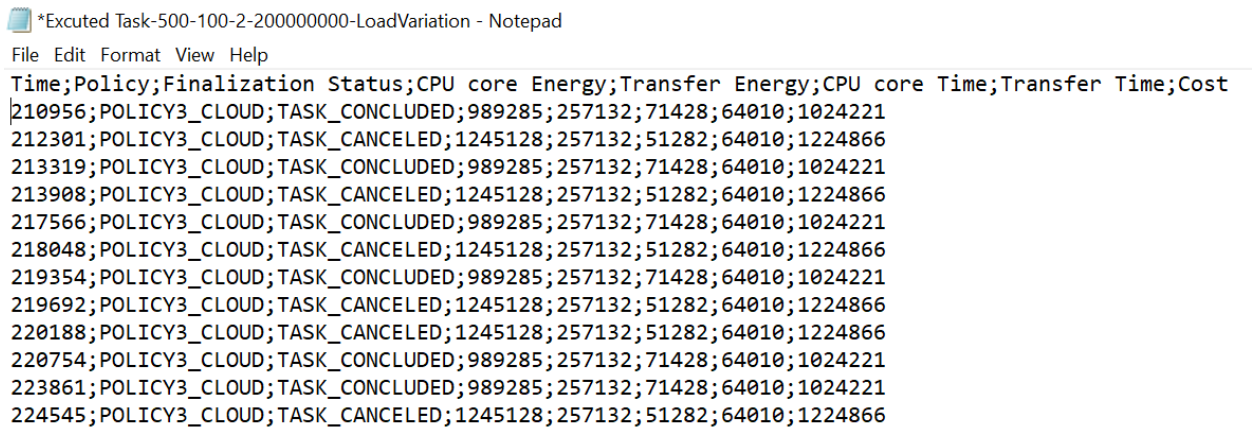
Now that the simulation for our 3 Tier Fog System is completed , let's analyze the results obtained from the simulator. The data for analysis is obtained using the following procedure:

Step-1. Run the Simulation

Step-2. Open the Simulation Log files which are generated in the form of a text file as follows. The taxonomy of file naming is as follows:

File Name: Executed Task- Total number of tasks- number of IoT devices-Number of MEC Servers- Computational CPU Cycles





Step-3: Now open the same file using Microsoft Excel to better visualize the data columns

	Time	Policy	Finalization Statu	CPU core Ener	Transfer Ener	CPU core Tin	Transfer Tin	Cost
158	299704	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
159	301328	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
162	302552	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
167	305321	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
174	312163	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
178	314340	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
181	316339	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
182	316843	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
187	320230	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
191	322494	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
192	323403	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
194	324481	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
196	325634	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
198	327769	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
199	329706	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008
202	331797	POLICY2_MEC	TASK_CONCLUDED	245025	140313	266666	32021	368008

The data file obtained from the simulator can be found in the link below.

1. Data for zero MEC Server: https://docs.google.com/spreadsheets/d/1rVCh8xv-ndzaRIz2CvFADnpYKdm09dxU/edit?usp=sharing&ouid=118065716154764533379&rtpof=true&sd=true
2. Data for one MEC Server: https://docs.google.com/spreadsheets/d/1CPb_i6Ggi8GBxvjWPrrUCaPTpIMrz_27/edit?usp=sharing&ouid=118065716154764533379&rtpof=true&sd=true
3. Data for two MEC Server: https://docs.google.com/spreadsheets/d/18RabXLTbJZpeSqbNK1EZnI5jg1ai6JA0/edit?usp=sharing&ouid=118065716154764533379&rtpof=true&sd=true

Impact of Number of MEC Servers on the Total Cost of the System.

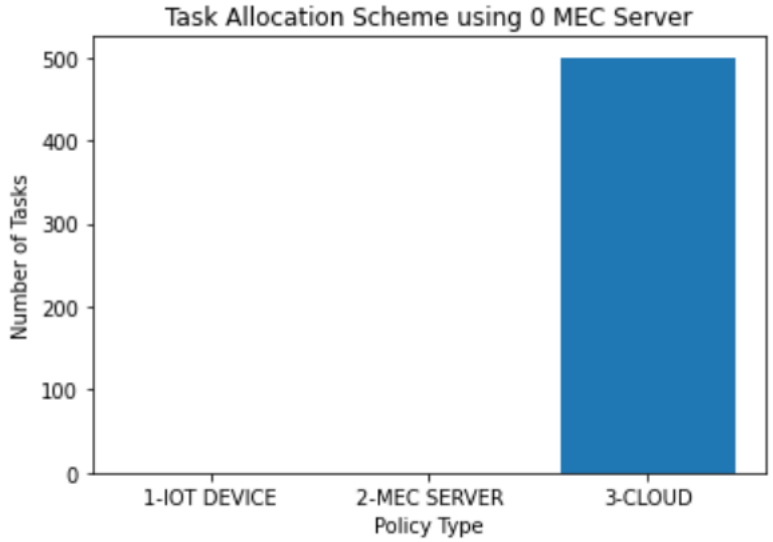
Here we analyze the impact of the number of MEC Cores on the allocation of tasks to the Cloud Layer. The analysis is carried out on the application with the below specified characteristics and the screenshots of the results are as follows:

Characteristics of Application

1. Task Generation Rate: 0.1 * 10^6 Tasks/second
2. Data Entry Size: 4 MB
3. Result Size: 625 MB
4. Computational Load: 200 million cycles of CPU
5. Critical Task: 50%
6. Deadline for Critical Task: 100ms

CASE-1: WHEN ZERO MEC SERVERS ARE DEPLOYED

Task Allocation:



Task Allocation Ratio:

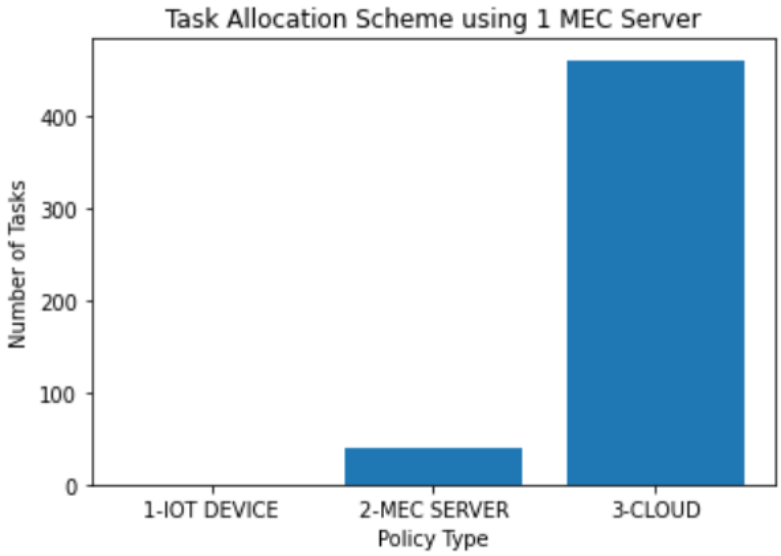
Cloud Layer: 92% MEC Layer : 0% IOT Layer: 0%

Observation:

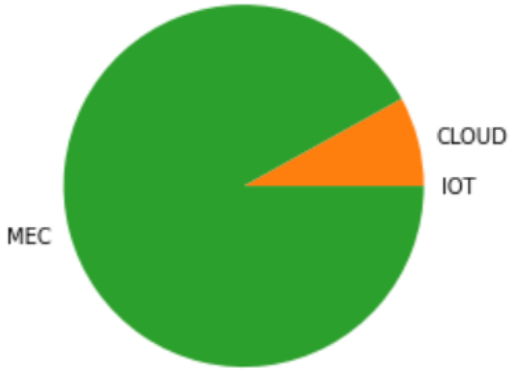
Here we can observe that since there are no MEC Servers , all the tasks are allocated to the cloud. Thus we have 500 tasks as the total number of tasks for application 2 and they are offloaded to the Cloud. This is due to the high Computational Complexity of Application-2. Thus the green portion of the pie chart signifies a 100 percent ratio of task allocation to our Cloud Layer.This is due to the fact there is an absence of fog layer and if tasks cannot be executed on the IOT devices, they have no option but to be offloaded to the cloud.

CASE-2: WHEN ONE MEC SERVER ARE DEPLOYED

Task Allocation:



Task Allocation Ratio:



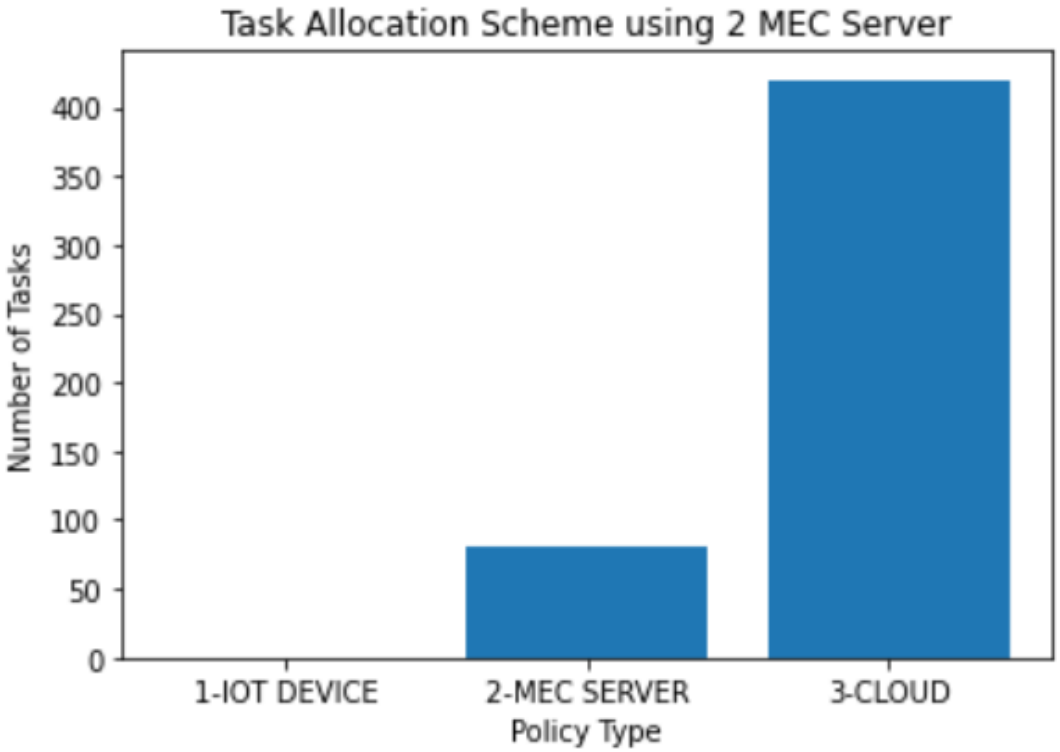
Cloud Layer: 92% MEC Layer : 8% IOT Layer: 0%

Observation:

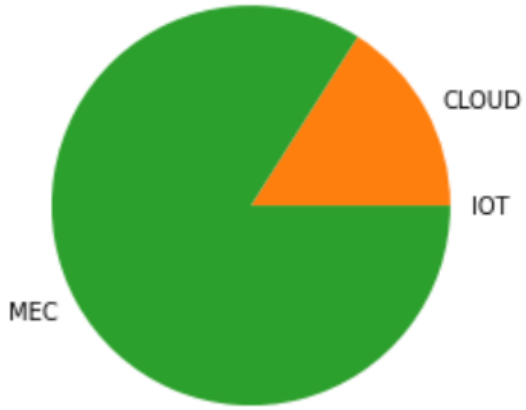
Here we can observe that when one MEC server is being allocated , then a certain portion of the work done by the cloud is done by our MEC layer. Here we can see that out of the 500 tasks, the number of tasks allocated to our cloud layer this time around is 460 and the number of tasks allocated to our MEC Server is 40. Since there is an introduction of an intermediate fog layer, a few tasks are allocated to the intermediate layer.

CASE-3: WHEN TWO MEC SERVERS ARE DEPLOYED

Task Allocation:



Task Allocation Ratio:



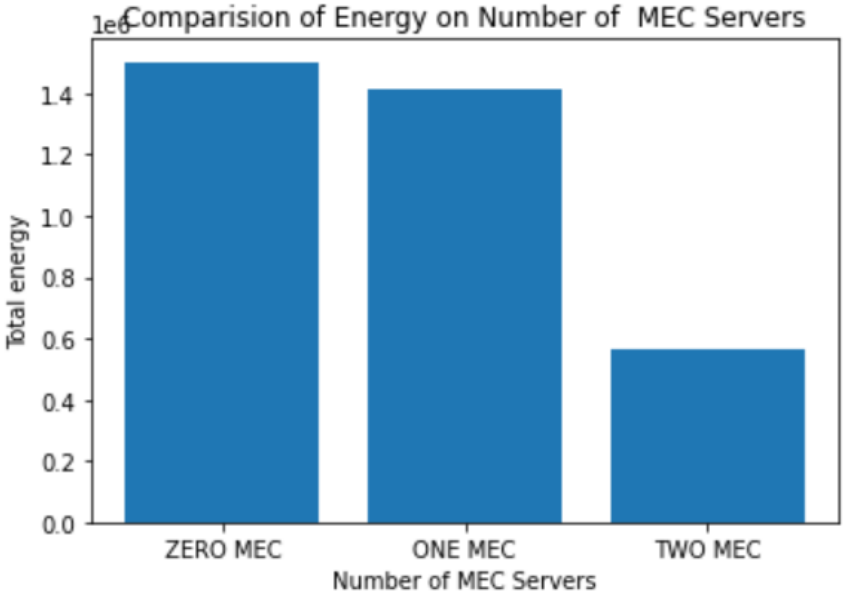
Cloud Layer: 84% MEC Layer : 16% IOT Layer: 0%

Observation:

Here we can observe that when we allocate 2 MEC servers, then more tasks are offloaded now to the intermediate layer. Thus less number of tasks are allocated on our Cloud thereby reducing the overall distance and cost of data transfer. According to theory and logic we can ascertain this hypothesis but let's look at some empirical data to prove this point.

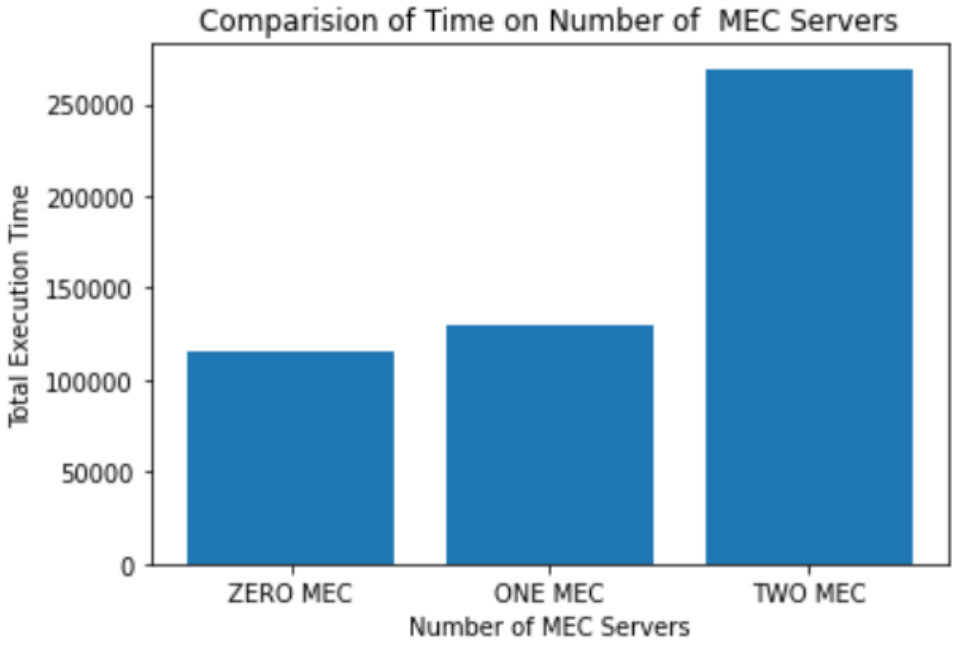
COMPARATIVE ANALYSIS

Total Energy Comparison:



Observation: Here we can take a look at the net energy consumed for the execution of all 500 tasks and we can see that the energy reduces as the allocation of tasks on our cloud layer decreases. Thus we have a decrease in the net energy consumed when the number of MEC servers are increasing. We can see that there is a decrease of about 6 percent in terms of energy when a single mec server is used and a decrease of about 62 percent in energy when 2 MEC servers are deployed. That is a huge decrease in the energy consumed.

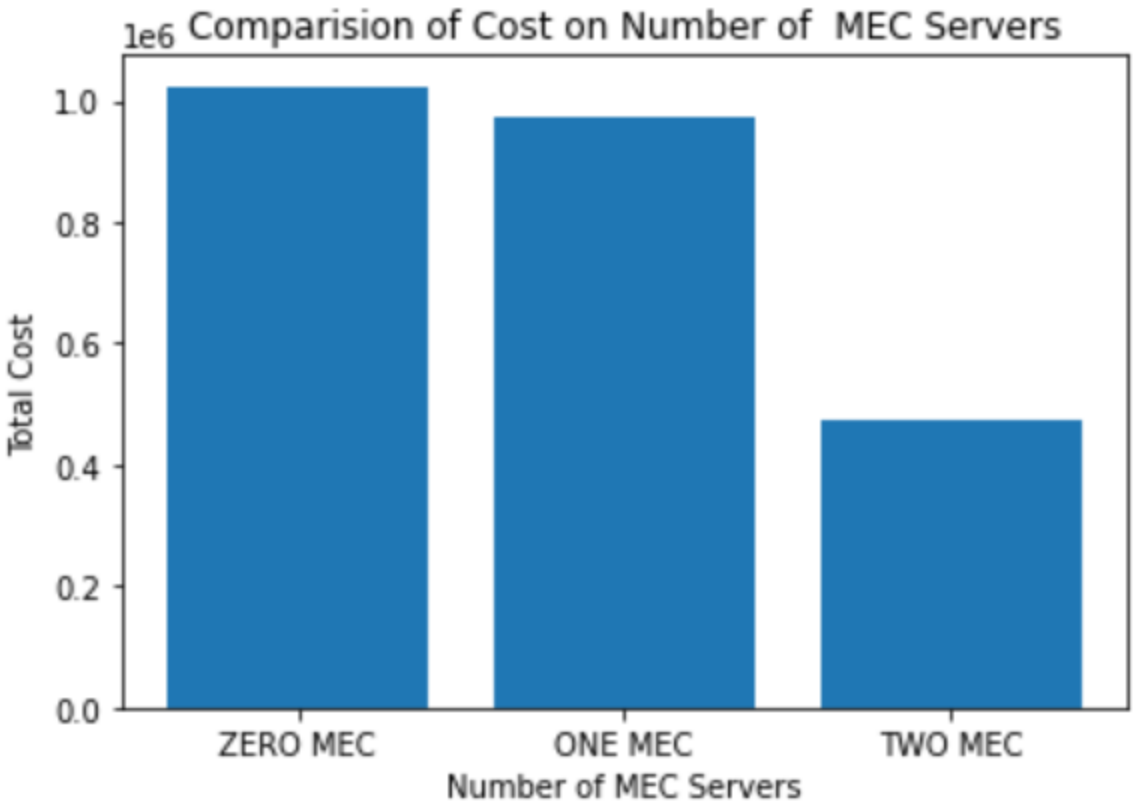
Total Time Comparison:



Observation:
Here we can observe that although the number of tasks allocated to our cloud layer is decreasing we can see that total time of execution increases massively in comparison to the deployment of no and 1 MEC server. Here we can see that the 2 important factors of performance analysis are Energy and time. Here we can see there is a significant increase in the total time of execution. As compared to the absence of the MEC layer , there is an increase of about 12% when one MEC server is deployed and an increase of about 67 % when 2 MEC servers are deployed.That is a large amount of increase in time coupled with a large amount of decrease in energy. This is where cost gives us a better picture

Now let's look at the net cost which is a weighted sum of both these factors according to their weight priority. Here energy is given a priority of 0.8 and time is given a priority of 0.2.

Total Cost Comparison:



Observation:

Here we can see that our net cost is decreasing as the number of MEC servers are increasing. Thus we can ascertain the fact that the addition of an intermediate layer optimizes the entire system by reducing the net cost entailed with energy as our primary priority.. As compared to the absence of the MEC layer , there is an decrease of about5 % when one MEC server is deployed and an decrease of about 54% when 2 MEC servers are deployed

Thus we can conclude successfully from these test cases that the addition of FOG or intermediate layer in our system architecture can indeed provide a much needed boost to our system by reducing the net cost entailed for execution.

Conclusion:

The team successfully designed and implemented a offloading and scheduling algorithm which uses a Integer linear programming approach to optimize the net cost of the system which is a weighted tradeoff between the energy consumed during execution and total time of execution. The above procedure was carried out in the Java programming language. The team later used the data generated using this simulation to analyze the effect of the number of MEC servers on the total cost of the system. We reached a conclusion from the analysis that the use of MEC servers has a positive effect on the total cost of the system and it reduces the number of tasks allocated to the cloud thereby reducing the total cost.

References:

1. Yu, H.; Wang, Q.; Guo, S. Energy-efficient task offloading and resource scheduling for mobile edge computing. In: 2018 IEEE International Conference on Networking, Architecture and Storage (NAS). [S.l.: s.n.], 2018. p. 1–4.
2. Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling," by T. Q. Dinh, J. Tang, Q. D. La and T. Q. S. Quek. IEEE Transactions on Communications, vol. 65, no. 8, pp. 3571-3584, Aug. 2017.
3. Guevara, Judy C., and Nelson L. S. da Fonseca. "Task Scheduling in Cloud-Fog Computing Systems." *Peer-To-Peer Networking and Applications*, vol. 14, no. 2, Jan. 2021, pp. 962–77, https://doi.org/10.1007/s12083-020-01051-9.Aazam, Mohammad, et al. "Offloading in Fog Computing for IoT: Review, Enabling Technologies, and Research Opportunities." *Future Generation Computer Systems*, vol. 87, Oct. 2018, pp. 278–289, 10.1016/j.future.2018.04.057. Accessed 8 Apr. 2020.
4. Almezeini, Nora, and Alaaeldin Hafez. "Review on Scheduling in Cloud Computing." *IJCSNS International Journal of Computer Science and Network Security*, vol. 18, no. 2, 2018, p. 108, paper.ijcsns.org/07_book/201802/20180214.pdf. Accessed 29 Apr. 2022.
5. Mukherjee, Mithun, et al. "Computation Offloading Strategy in Heterogeneous Fog Computing with Energy and Delay Constraints." *IEEE Xplore*, 1 June 2020, ieeexplore.ieee.org/document/9148852. Accessed 29 Apr. 2022.
6. Islam, Mir Salim Ul, et al. "Context-Aware Scheduling in Fog Computing: A Survey, Taxonomy, Challenges and Future Directions." *Journal of Network and Computer Applications*, vol. 180, Apr. 2021, p. 103008, 10.1016/j.jnca.2021.103008. Accessed 25 Feb. 2021.
7. Bukhari, Muhammad Mazhar, et al. "An Intelligent Proposed Model for Task Offloading in Fog-Cloud Collaboration Using Logistics Regression." *Computational Intelligence and Neuroscience*, vol. 2022, 25 Jan. 2022, p. e3606068, www.hindawi.com/journals/cin/2022/3606068/, 10.1155/2022/3606068. Accessed 29 Apr. 2022.
8. Mukherjee, Mithun, et al. "Computation Offloading Strategy in Heterogeneous Fog Computing with Energy and Delay Constraints." *IEEE Xplore*, 1 June 2020, pp. 1–5, https://doi.org/10.1109/ICC40277.2020.9148852.
9. Yin, Jiaying, et al. "Energy Efficient Priority-Based Task Scheduling for Computation Offloading in Fog Computing." *Algorithms and Architectures for Parallel Processing*, 2022, pp. 564–77, https://doi.org/10.1007/978-3-030-95384-3_35.
10. Movahedi, Zahra, et al. "An Efficient Population-Based Multi-Objective Task Scheduling Approach in Fog Computing Systems." *Journal of Cloud Computing*, vol. 10, no. 1, Oct. 2021, https://doi.org/10.1186/s13677-021-00264-4.