OpERA: A Multi-Objective Optimization Approach for Edge Based Resource Allocation

Habiba Hussein Mohamed

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science

in Computer Science and Systems

University of Washington

2022

Committee:

Eyhab Al-Masri, Chair

Minjia Zhang

Hossam Fattah

Program Authorized to Offer Degree:

Computer Science and Systems

University of Washington

Abstract

OpERA: A Multi-Objective Optimization Approach for Edge Based Resource Allocation

Habiba Hussein Mohamed

Chair of the Supervisory Committee:
Eyhab Al-Masri
School of Engineering & Technology

In the Internet of Things (IoT) era, as system complexity increases and IoT systems require greater complexity, adaptive optimization strategies are required to improve computation offloading across computing infrastructures. For most offloadable tasks, offloading is targeted to the cloud, in recent years efforts have been made to migrate away from the reliance on the cloud because of the pitfalls it presents for latency-sensitive applications. Solutions proposed include task preprocessing or computation offloading to multi-access edge computing nodes. However, those solutions come with challenges related to complex hardware and software requirements, expansive implementation and complex architecture. As part of this thesis, we propose OpERA, an edge-based resource allocation optimization framework for aiding in seamlessly offloading tasks across edge, fog, and cloud computing layers and architectures. By capturing the task attributes and requirements from the end user, OpERA can identify suitable resources to execution the task and then optimizes the process by singling out the computational resource with the best residual gains

(i.e., cost reduction, minimal energy consumption, high processing capability, etc.). By optimizing resource allocation in computation offloading in the status quo, we are able to increase the likelihood of successful task offloading for computationally intensive tasks such robotic surgery, autonomous driving, smart city monitoring device grids, and deep learning tasks.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

Table Number Page

ACKNOWLEDGEMENTS

I would first like to express my sincere gratitude to Professor Eyhab Al-Masri. I am thankful for all the insightful guidance from my advisor, Dr. Eyhab Al-Masri, provided throughout this process. Professor Eyhab Al-Masri provided immense knowledge and invaluable input throughout the formulation of this research methodology.

I am also thankful to Dr. Minjia Zhang for suggestions and guidance throughout this thesis project. Dr. Minjia Zhang's critical questions helped shape the research at every stage of the thesis project.

DEDICATION

I dedicate my thesis work to my family and many friends. A special feeling of gratitude to my loving parents whose words of encouragement and pushed for me to persevere in my work. I also dedicate this thesis to my husband and children who have supported me throughout the process.

Chapter 1

INTRODUCTION

The Internet of Things (IoT) is an expansive network and interconnected service infrastructure consisting of heterogeneous devices that are seamlessly incorporated into the Internet. The use of IoT systems allows us to connect to a plethora of other devices and each other from any location. IoT systems give us the ability to equip everyday devices with technological advances. As IoT systems become more integrated into our daily tasks, they also require an increase in hardware sophistication and computational power. As the resource capabilities of native IoT devices are not able to execute such tasks, reliance on the cloud computing layer and multi-access edge computing have been introduced to mitigate this problem.

The rapid growth of sophistication in IoT systems includes the emergence of new applications for use cases in mobile gaming, virtual reality, healthcare, and transportation. Computationally intensive tasks are used by these applications which generally require low latency and high energy consumption. However, even as the hardware sophistication of these IoT devices increases, there are some computational tasks that cannot be performed on them due to hardware and software limitations. As illustrated in Figure 1.1, the offloading schema in an IoT perspective, the number of computations performed increases as the magnitude of the number of individual nodes and resources decreases. In the cloud computing landscape, there are more robust resources that are more localized to hundreds of nodes. As we migrate to the fog layer and further to the edge layer, there is less localization of both computations and nodes. We also see a decrease in both the number and robustness of computational resources as the number of devices increases on a

billion scale. We define a task as parts of a program or some process that is offloaded to a peer device (edge node), a fog node or a cloud node, as illustrated in Figure 1.1.

This deficiency brings on opportunities such as task offloading. Task offloading is not a novel concept and has been used in areas such as TCP offloading [9]. TCP offloading is used in order to reduce the CPU overhead of TCP/IP on fast networks. This sort of offloading divides large volumes of data into smaller units of data that are sent between source and destination. Computation offloading follows a similar paradigm in which offloading is done to either the cloud layer or a multi-access edge computing network.

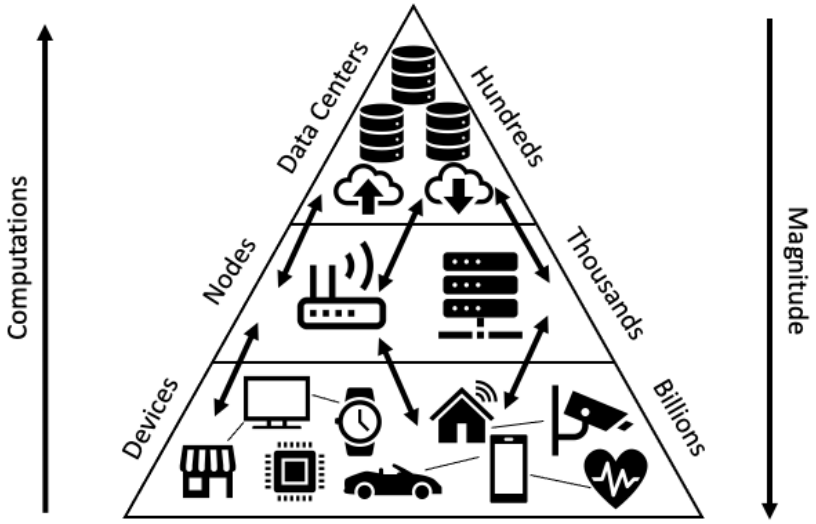

Figure 1.1: Offloading Schema in an IoT Paradigm

The emergence of edge computing architecture allows for information to be processed in a node that is nearby the native device. The introduction of multi-access edge computing has decreased the transfer latency of information but as with any architecture, as shown in Figure 1.2, it requires a complex ecosystem of hardware, software and networking. The use of multi-access edge computing presents security concerns. The network of interconnected devices will need to

apply uniform security policies. Multi-access edge computing network administrators will also need to define identity and access management rules for a large number of users spread across the multi-access edge computing network. Even though multi-access edge computing is promising for latency reduction there are still challenges with deployment in relation to the heterogeneity of the edge nodes and all of their requirements. The use of multi-access edge computing has challenges meeting service level agreements (SLA) with different service architectures.

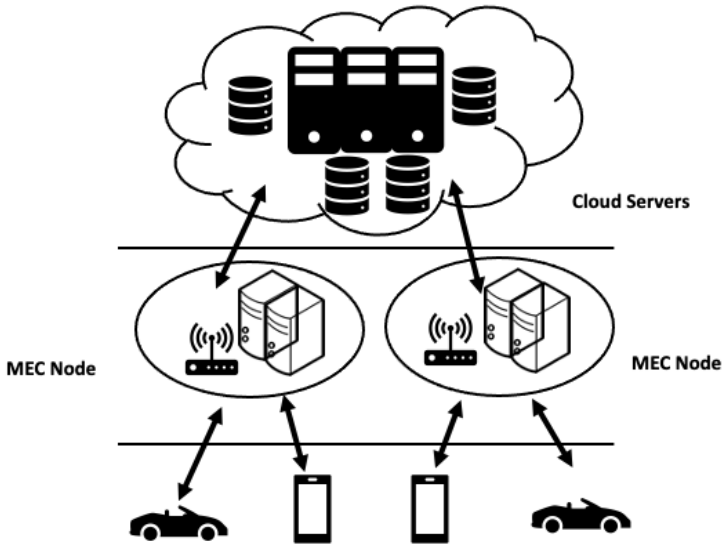

Figure 1.2: Multi-Access Edge Computing Architecture

Computation offloading can also be done by transferring tasks to remote cloud resources for execution and returning of the deliverables to the native device. Cloud service providers provide access to a plethora of computing resources in the areas of software applications, artificial intelligence, storage and processing power. Even though computation offloading allows access to robust and heterogeneous processing resources, it has its limitations. Majority of the limitations can be tied to latency issues when relying on external computing resources as well as downtime. Downtime can occur with cloud computing services in regard to outages or network connection

disruptions. Also, the latency factor introduced when computing over the internet cannot be tolerated by mission-critical applications, such as autonomous driving and image classification analytics.

Whether offloading occurs to a cloud service provider or a multi-access edge computing network there are a few main challenges that need to be considered in an offloading strategy. Some of those include how to offload a task for example whether to use a wireless network and which offloading path to follow. Another concern gets introduced with being able to identify parts of the computation that are offloadable and what parts aren't. Then, finding appropriate resources with the necessary hardware requirements to execute the computation. Finally, when to begin offloading a task in an appropriate way such that the network signal isn't dropped, and the migration of the task is not disrupted.

In relation to offloading strategies there are a lot of optimization models that have been applied in order to increase the likelihood of successful offloading. In the area of resource allocation, some of those optimization strategies include gathering data from historical offloading instances and resource scheduling. Existing research efforts for resource allocation in an edge-based computing environment for a given task include algorithms that allocate resources on a predetermined schedule or manual resource allocation [1, 2, 3, 4]. Additionally, some work has been done on calculating the resource requirements for a task and preemptively allocating resources. While many of these methods are effective they require human intervention and do not enable IoT devices to make offloading decisions.

A main focal point of the offloading strategy is optimal resource allocation. The strategy of relying on only cloud computing or the use of a multi-access edge computing network both have drawbacks. A hybrid approach that is able to identify and allocate resources from all three

computing layers (cloud, fog, edge) is integral and a missing key part in the offloading strategy. A strategy that is able to take input from the end-user on the specific attributes that are important to their use case, as shown in Table 1.1, whether it be a mission critical scenario such as a self-driving car or a scenario such as compounded data collection from a soil sensor in a smart city. The task types outline Table 1.1 are based on earlier research efforts. The identification of resources from a variety of attributes in a multi-object optimization strategy is necessary. This multi computational layer approach would not restrict users to resources available in any individual computing environment.

Table 1.1 Offloading Task Strategy Types based on [10]

Task Type	Description	Offloading Strategy Key Attributes
Robotic Surgery - Compute Centric	A mission-critical and time-sensitive	Low runtime, high memory availability and processing power
Soil Sensor - Cost Centric	Somewhat time sensitive, requires efficient processing and cost	Low execution cost
Eigen-decomposition of a Laplacian Matrix - Memory Centric	A lot of processing of incoming data	High memory availability

With these requirements in mind, the objective of our research was the development of a reliable resource allocation offloading strategy that can be utilized on both the edge and fog computing layers. In order to allocate resources across multiple computing environments in an optimal manner we need to develop an optimization framework that performs in respect to the requirements of the task from the end-user. Also, this framework needs to be able to handle a variety of attributes that are critical for the offloadable task while working to maintain Quality of Service (QoS) level agreement. The optimization strategy would also need to work to minimize

the task execution time, energy consumption and cost spending. Finally, the optimization framework will need to create a buffer for processing and memory availability. To execute on these needs, we've formulated OpERA: a multi-objective optimization approach for edge based resource allocation.

The prime objective of OpERA is to identify resources to offload a task in a complex environment with multiple sometimes opposing objectives. OpERA aims to provide end users a flexible and adaptable approach to a wide range of environments and needs while providing performance gains and efficiency of executing tasks. OpERA utilizes a multi-criteria decision making optimization (MCDM) technique for order of preference by similarity to idea solution (TOPSIS).

Our contribution in this thesis is as follows.

- We investigate an edge based resource allocation optimization strategy to support task offloading operations.
- We develop OpERA, a multi-objective optimization model that is based on runtime prediction, cost derivation, and energy consumption objectives.
- We conduct a series of experiments that focus on the usefulness of MCDM methods such as TOPSIS as a multi-objective optimization strategy.
- We evaluate the performance of the OpERA optimization model using real world clustering datasets.
- We provide insights on the feasibility of MCDM methods on resource allocation across computing environments for supporting offloading operations.

The rest of this thesis is as follows: Chapter 2 discusses the related work. Chapter 3 discusses the prediction model, cost derivation model, and energy consumption model. Chapter 4 discusses the architecture of the OpERA framework. Chapter 5 discusses the datasets that were used, experimental design and outcomes of the optimization strategy. Chapter 6 discusses the conclusions and plans for future work.

Chapter 2

RELATED WORK

Task offloading enables end users to maintain a level of QoS as the complexity of device computations grows. This section will explore related works in offloading, resource allocation, and some background optimization techniques. Offloading strategies can be categorized into offloading to a MEC network or edge/fog to cloud offloading and vice versa.

2.1 Offloading on Edge Layer, Fog Layer and Cloud Layer

Task offloading can be done by moving the task from the edge or fog layer to the cloud layer. When a task is offloaded from the edge to the cloud layer it can be referred to as full-offloading [20]. When a task is offloaded from the fog to the cloud layer it can be referred to as partial-offloading [20]. The edge and fog layers are not as resource rich as the cloud layer. Cloud computing is especially useful for computation heavy workloads but falls short on low latency requirements [21]. In order to reduce the latency pre-processing of data may be done at the edge or fog layer before offloading the task to the cloud layer [22]. However, this added step also increases the bandwidth due to the large data transfer that occurs as a result of preprocessing [22]. Optimizations are needed in the space in order to reduce latency as well as minimize bandwidth due to pre-processing. This thesis focuses on optimizing task offloading by allocating the adequate resources that can complete a task in less time than the native device without a need for task preprocessing.

2.2 Offloading on Multi-Access Edge Computing (MEC) network

Offloading strategies discussed in Section 2.1 are more suited for non-mobile devices that are more focused on reduction in energy consumption and reduction of processing resources. In order to avoid network traffic that mobile devices can encounter with the offloading methods described in Section 2.1, a new computing paradigm, Multi-Access Edge Computing (MEC), was introduced. A compilation of research explorations of MEC offloading in relation to resource allocation is detailed in Table 2.1.

Table 2.1 MEC Task Offloading Comparison Compiled from [27]

Publication	Issues presented	Methods	Deficiency
[23]	Task placement, Resource allocation	Greedy Algorithm	Doesn't take into account possible resources in neighboring base stations that have MEC servers
[24]	Resource allocation	Greedy heuristic	Can only get a near optimal solution
[25]	Task offloading, Resource allocation	Greedy heuristic	Very complex offloading plan
[26]	Resource allocation, channel allocation, power control	Greedy heuristic	Task partitioning is mentioned as an advantage but no process outlined on how to partition task

The task offloading strategies noted in [23, 24, 25, 26] are mathematical optimization based on greedy metaheuristics algorithms. Unlike an iterative metaheuristic method, greedy algorithms do not start with a possible optimal solution set. In order to deduce the optimal alternative the processing in [23, 24, 25, 26] needs optimization parameters from the surrounding MEC network

such as the current network delay. Offloading strategies that focus on MEC offloading reduce the set of feasible solutions by not taking into account edge to cloud and fog to cloud offloading.

2.3 Resource Allocation Optimization in Task Offloading

Various mathematical and computational algorithms and models have been developed in frameworks to address optimization in task offloading. In [47] a heuristic algorithm is used to optimize resource allocation in the computational fog layer for car based applications. In [30] researchers utilize Reinforcement Learning (RL) to address resource scheduling problems to minimize offloading delays. Particle swarm optimization (PSO) has been utilized to find optimal offloading schemes for specific path scenarios [28]. All of these optimization strategies have limitations, for example heuristic algorithms can only be used to find an approximate solution [30]. RL based resource allocation optimization requires extensive amounts of data and computation [29]. PSO has a very low convergence rate which increases its deduction time as well [28]. However, MCDM methods such as TOPSIS provide the most optimal solution set of solutions to the end user.

Chapter 3

METHODOLOGY

In this chapter, we outline the inputs to the optimization model that will be discussed in Chapter 4. In order to lay the foundation for the optimization model, prerequisite processes must be completed such as data preparation and calculating derived input values to the optimization model. Even though the definitions were touched upon for a resource, task, and offloading in Chapter 1. When we refer to offloading, a computational node, a resource and a task they are defined to be the following.

- Node: An IoT device, virtual machine, or containerized resource.
- Offloading: The migration of a resource intensive task to a separate computation node.
- Resource: A computing unit that exists on either the edge, fog, or cloud computational layer that can include tools and applications such as networking, storage, databases, servers, hardware and software that can execute a task.
- Task: A process or part of a program can be offloaded to a computational node.

In Section 3.1 we begin by discussing the datasets used in detail and the placement of the computational nodes in the edge, fog, and cloud layers.

3.1 Data processing & Computing Layers Derivation

The datasets that we use in designing the OpERA framework are Materna [15] and Auvergrid [14] from the Grid Workload Archives (GWA) [13]. Table 3.1 outlines the metadata

for the datasets and the key observed metrics that we utilize. The datasets were stored in Big Query tables.

Table 3.1 Materna and Auvergrid GWA Dataset Schema

Dataset	Metadata	Key Metrics
Materna VM trace	1,592 machines 13,940,000 traces	Number of CPU cores CPU usage (MHz) Memory usage(KB)
Auvergrid Job trace	405 users 404,176 jobs	Number of CPU cores CPU runtime(s) Memory usage(KB)

In reference to the definitions for a computational node, resource and task we will explore how the dataset provides each of these key elements. The 1,592 virtual machines from the Materna dataset are resources, they will each be mapped to computational nodes in either the edge, fog, or cloud computational layer. As shown in Figure 3.1 we create a 1-to-1 mapping for each virtual machine to a resource unit in either the cloud, fog, or edge computational layer. We follow a model found in [10] to determine the number, proportion, variety and robustness of resources in each layer. As references in our related works section we can expect the resources that have higher amounts of CPU and memory usage to be prevalent in the cloud layer because of the number of computational nodes that are available in the cloud and the heavy usage of cloud resources.

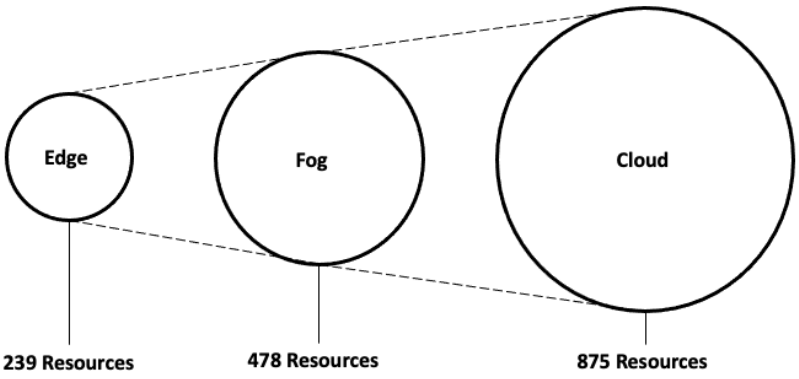

Figure 3.1: Placement of Resources in Computational Layer

The 404,176 jobs from the Auvergrid dataset are tasks, they will also be mapped to a computational layer in which the task is being performed. The task runtimes lasted anywhere from 15 minutes to about 4 days. The memory utilization for the tasks ranged anywhere from 1,700 KB to 3,667,652 KB. About 15% of the Auvergrid jobs had missing data for key metrics such as CPU runtime and memory usage so that data was not utilized.

The Auvergrid and Materna datasets provide measured values which we will use to derive calculated values such as estimated runtime, energy consumption, and cost. The reason why we require these values is to simulate how an offloaded task will perform on alternative resources. The dataflow of the process outlined in Figure 3.2 shows that the entry point is job or task information the Auvergrid job traces. We then derive the estimated runtime, energy consumption, and cost which will be covered in more details on Sections 3.2-3.4.

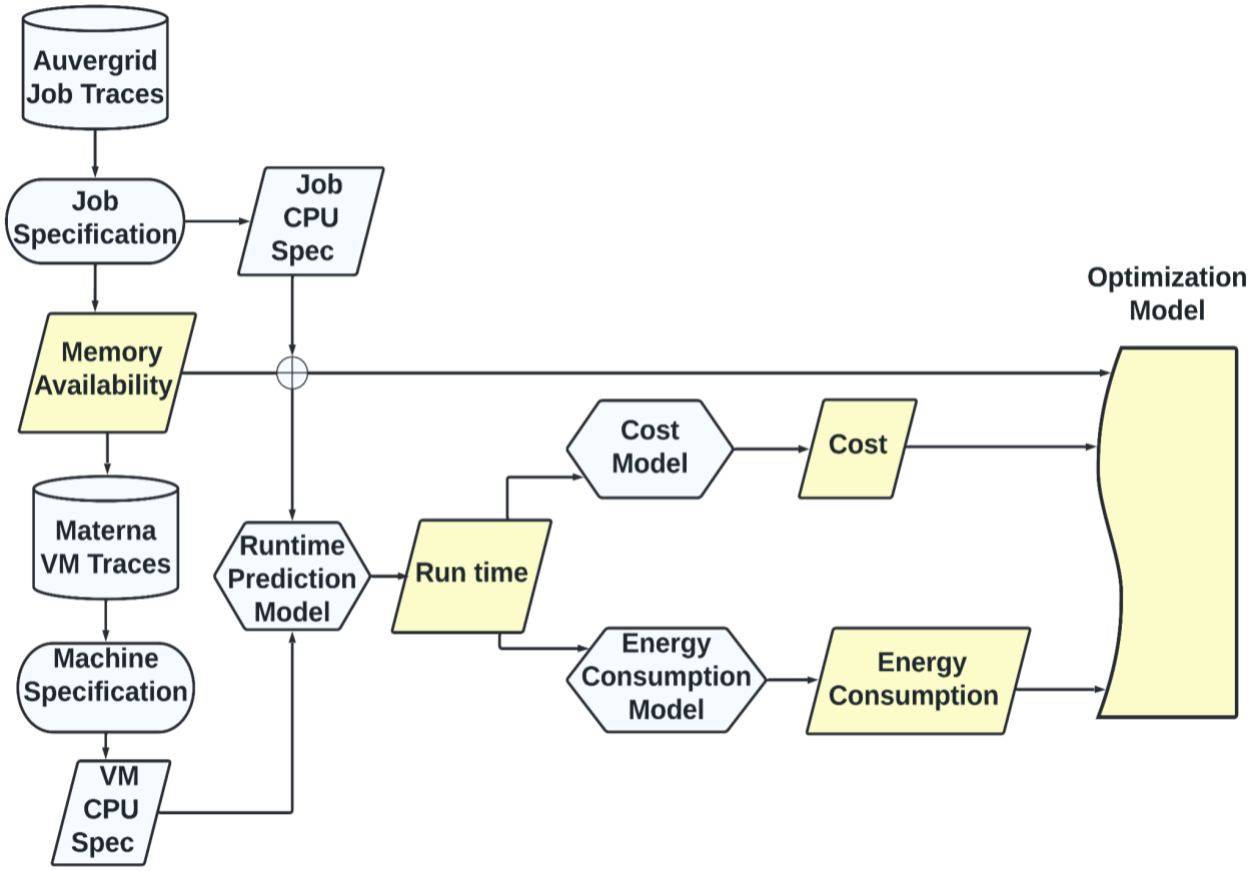

Figure 3.2: Data Ingest and Derivation of Optimization Model Inputs

3.2 Hypothesis

H1: If we use a resource allocation optimization model, we can identify alternative resources to offload a task with a reduction in the task cost, energy consumption, and runtime.

3.3 Runtime Prediction Model

The inputs to the optimization model include measured values from the job traces and virtual machine traces as well as derived values. The reason why we need the derived values is for

attributes to input into the optimization model. With the derived values we are essentially projecting what the runtime would be on a chosen alternative.

Amdahl's law [16] is used to predict the theoretical maximum speedup in parallel computing when a single processor is used. All of the tasks from the Auvergrid dataset were run on a single processor. Amdahl's law is used to find the maximum speedup of a system when multiple processors are used.

$$Speedup(n) = \frac{1}{(1-p) + (\frac{p}{n})} \qquad (3.1)$$

$$n \text{ is the number of cores}$$

$$p \text{ is the portion of a program that is parallelizable } (0.2 \text{ to } 0.99)$$

To use Amdahl's law (Equation 3.1) we need to know the portion of the program that is parallelizable. That information was not provided in the Auvergrid jobs trace data. We generated a p value for each job from values within the range of 0.2 – 0.99 such that all of the p values fit a Gaussian distribution. It should be noted that generating the p values in a Gaussian distribution with a standard deviation of 0.1 and mean we use is 0.595 is ideal to reduce bias in the random number generator.

We created a BigQuery client via a data manipulation language (DML) script that scanned the Auvergrid table and used the Numpy python library to write to a table that contained a job id and a p value.

The calculated speedup factor can then be used to derive the predicted runtime, as shown in Equation 3.2, we multiply the speedup factor by the observed runtime for a job. The predicted runtime is reflective of how long the job would take to run an alternative resource.

$$\boldsymbol{Runtime_{Predicted}} = \frac{\boldsymbol{Runtime_{Observed}}}{\boldsymbol{Speedup\ Factor}} \quad (\mathbf{3.2})$$

3.4 Energy Consumption Model

The energy consumption of a processor can be defined as its thermal design power (TDP) in Watts multiplied by the runtime [11] as shown in Equation 3.3.

$$\boldsymbol{Energy} = \boldsymbol{TDP} * \boldsymbol{Runtime} \quad (\mathbf{3.3})$$

The trace metrics from the datasets did not include hardware specifications; we know only CPU was utilized to execute computations. Since the CPU specification was not provided we conducted research on the CPU types that would be reflective of resources found in the edge, fog, and cloud layers which is listed in Appendix A.

To calculate the energy consumption for a task on its native device or an alternative device, the runtime that was calculated in section 3.1 in combination with the TDP from the CPU specification for the node is utilized.

3.5 Cost Basis Model

The cost basis was derived from a cloud service provider for the cost to host a virtual machine with similar specifications to the CPU types assigned to the virtual machines and memory capabilities. The cloud service provider provided values from the cloud computing layer and the estimation for the fog and edge layers were interpolated. For the fog and edge layers, we assume that the cost differentials are going to be lower, 2 times cheaper for fog and 1.5 times cheaper for edge. Using the Azure Pricing Calculator [12], the monthly cost basis in Table 3.2 was derived. To predict the cost of a computation we take the cost basis and multiply it by the predicted runtime.

Table 3.2 Cost Basis of VM Instances in Edge, Fog, and Cloud Computing Layer

Layer	Monthly Cost Basis for 1 vCPU
Cloud	$146.88
Fog	$73.44
Edge	$48.96

3.6 Database Schema

As a cumulation of the data processing in Sections 3.2-3.4 the inputs to the optimization model which are in the following database architecture is depicted in Figure 3.3. The Task table contains a 1-to-1 relationship with the Parallelization table. The Task table has a 1-to-n relation with the Hardware Specification table. The Resource table has a 1-to-n relation with the Hardware Specification table.

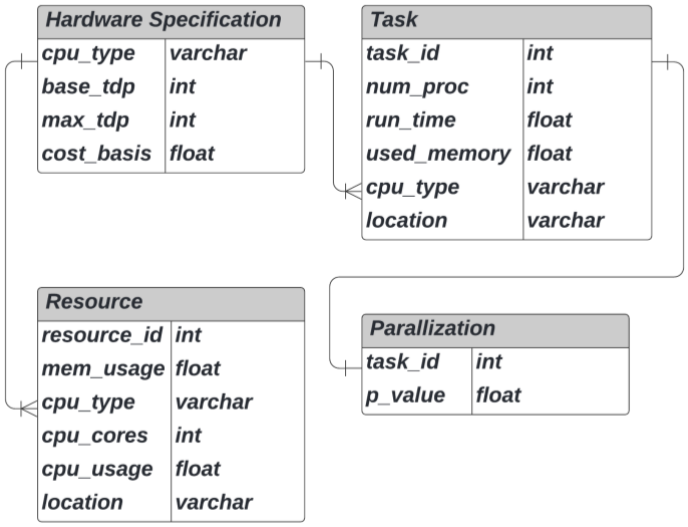

Figure 3.3 Optimization Model Database Schema Inputs

Chapter 4

OPTIMIZATION AND DECISION MAKING MODEL

The use of MCDM processes is a classical example of mathematical optimization for business applications or in our case resource allocation for task offloading. MCDM processes utilize linear programming to achieve objectives given a set of linear constraints [17]. MCDM processes also utilize the minimization or maximization of certain attributes via linear objective functions [17]. When the objective functions for the attributes are graphed. We can see the region containing the feasible solutions as seen in Figure 4.1. The positive slopes represent objective functions for attributes that are being maximized while the negative slopes are for objective functions whose attributes are being minimized.

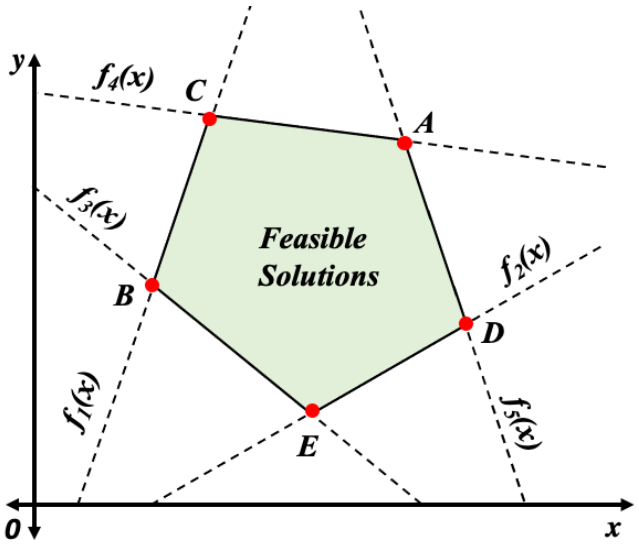

Figure 4.1: Bounded Region of Feasible Solutions of Objective Functions Adapted from [17]

As shown in Figure 4.1 for objective functions 1 and 2, the ideal values within the bounded regions are vertex C and vertex D. For objective functions 3,4, and 5 the ideal solutions exist at vertex B,

vertex C, and vertex A. Identifying the region of feasible solutions is deduced however choosing an optimal solution requires further analysis.

TOPSIS is a MCDM technique that seeks to choose the best alternative that is the smallest euclidean distance from the ideal best solution and the largest euclidean distance from the ideal worst [18]. It takes into account weighted preferential attributes from the user. The preference for an attribute is beneficial (will be maximized) or non-beneficial (will be minimized) [18]. The input weights to TOPSIS can be arbitrarily decided but they must all add to 1. There are statistical methods to determine attribute weights [19] such as:

- AHP
- Entropy
- Standard Deviation
- CRITIC

Furthermore, for this thesis in order to design a more user-centric product, we used non-statistical methods that required user input.

TOPSIS has 6 main steps as shown in [18], they are as follows.

1. Construct a normalized decision matrix R

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} {x_{ij}}^2}} \quad for\ i\ = 1,2,3,\ldots,m\ and\ j\ = 1,2,3,\ldots,n$$

2. Construct the weighted normalized decision matrix V

$$v_{ij} = r_{ij} \times w_j\ for\ i\ = 1,2,3,\ldots,m\ and\ j\ = 1,2,3,\ldots,n$$

3.

a. Determine the positive-ideal value which is the largest observed value in each column

$$IDR = (max_i v_{i1,} max_i v_{i2}, \ldots, max_i v_{in}) = ({v_1}^+, {v_2}^+, \ldots, {v_n}^+)$$

b. Determine the negative-ideal value which is the smallest observed value in each column

$$IDR = (min_i v_{i1,} min_i v_{i2}, \ldots, min_i v_{in}) = ({v_1}^+, {v_2}^+, \ldots, {v_n}^+)$$

4. Measure the Euclidean distance for each alternative to the positive ideal one and the negative ideal one.

$${d_i}^+ = \left[\sum_{j=1}^{n} \left(v_{ij} - {v^+}_j\right)^2\right]^{\frac{1}{2}} \; for\; i = 1,2,3,\ldots,m$$

5. Produce a composite index that can range from 0 to 1 by calculating the relative closeness of the alternative to the ideal solution.

$${d_i}^- = \left[\sum_{j=1}^{n} \left(v_{ij} - {v^-}_j\right)^2\right]^{\frac{1}{2}} \; for\; i = 1,2,3,\ldots,m$$

6. Rank the alternatives from 1 to m depending on the values of their composite indices.

$$CI_i = \frac{{d^+}_i}{{d^+}_i + {d^-}_i} \; for\; i = 1,2,3,\ldots,m$$

For this thesis we will use attributes available based on the metrics of the Materna VM dataset. The attributes, outlined in Table 4.1,we will consider are memory availability, CPU usage, runtime, execution cost, and energy consumption. The weights used are adapted from the edge

base operation types from [10] which are outlined in Table 1.1. The preference for an attribute can either be beneficial or non-beneficial. In a computational task that requires a large memory segment for processing matrices would require high memory availability. That is, the higher the memory availability would be desired in such cases which makes it beneficial for the computational task. A long runtime, high CPU usage, and high execution costs are attributes we would aim to minimize since they are not favorable for offloading outcomes.

Table 4.1 Preferential Attributes Inputs to Optimization Model

Attribute	Preference
Memory Availability	+ Beneficial
CPU Usage	- Non-beneficial
Execution Cost	- Non-beneficial
Energy Consumption	- Non-beneficial
Runtime	- Non-beneficial

After the ranking of the alternatives is complete, the final step is to do a manual verification and choose the best alternative that fits with the specific task. We analyze the performance scores and then we identify for each alternative the granularity of the data within each attribute. After performing a verification of the data, we then do a comparison between the alternatives to determine if actual metrics within the bounded region of the optimal are in the feasible solutions as shown in Figure 4.1.

Chapter 5

EVALUATION AND ASSESSMENT

To evaluate the OpERA framework that we developed, we conducted tests with varying operation types by incorporating several attributes and by varying our weights. This chapter describes the details of the experimental setup and the tests we performed. The purpose of using experimentation and assessment is to show the effectiveness of our framework by revealing the performance assessment obtained.

The inputs to the optimization model will be our weight criteria, decision matrix, and preference criteria. For the decision matrix, the values are queried from the resource table. Given a task's attributes, the resources that meet the minimum threshold for memory availability are initialized in the decision matrix. The dimensions of the decision matrix are 6 x M, which is the number of resources that meet the minimum threshold requirement for memory availability. M is bounded between 0 and 1,592 which is the maximum number of VMs available from the Materna dataset. If M is 0 then the algorithm stops since there are not available resources that meet the minimum threshold for memory availability. As shown below, the weight matrix has 5 attributes, of which are memory availability(MA), execution cost (ExC), energy consumption (EnC), predicted runtime (R), and CPU usage (CPU). The preference for each attribute is denoted with + or a –. Memory availability is the only beneficial attribute since it is preferred to have a higher memory availability. The remaining four attributes are non-beneficial, we want to minimize execution cost, energy consumption, predicted runtime, and CPU usage.

$$\boldsymbol{W} = [\boldsymbol{w}^{+}_{\boldsymbol{MA}}, \boldsymbol{w}^{-}_{\boldsymbol{ExC}} -, \boldsymbol{w}^{-}_{\boldsymbol{EnC}}, \boldsymbol{w}^{-}_{\boldsymbol{R}}, \boldsymbol{w}^{-}_{\boldsymbol{CPU}}]$$

The assignments for the test cases that we will explore as outlined in Table 5.1. We start with one attribute and transition the dominance to another and observe results for all cases except for the General Case which by definition is evenly weighted for all attributes.

Table 5.1: Weight Assignment Outline for Test Cases Explored in Sections 5.1-5.5

Treatment	Weight Dominance Transition
General Case	Even weight for all attributes
Memory Driven Use Case	Memory availability → Execution cost
Cost Driven Use Case	Execution cost → CPU usage
Compute Driven Use Case	Memory availability ↔ Execution consumption ↔ CPU usage

5.1 General case

The first use case we will consider is a general use case. In a general use case we consider resource allocation optimization for a task whose attributes are all equally weighted. The weights are evenly distributed at 0.2 each. Considering Task ID 108145 with the attributes in Table 5.2. The task is currently being executed in the cloud layer, it currently costs 2/1000 of a penny and will use 2,838 Joules of energy and will run for 33 seconds.

Table 5.2: Task ID 108145 Specifications

Used Memory	48612 KB	Location	cloud
Cost	$0.002	Run Time	33 seconds
Energy Consumption	2838 J	CPU Type	corei7-6700K

We provide details of Task 108145, weights and preferences to the OpERA framework. The output of the results are displayed in Table 5.3. With a general use case, the optimization model should not capitalize on any single attribute. Looking at the attribute of the top ranked alternative, the categories of memory availability, CPU usage, energy consumption, and execution time are not the optimal selections. As shown in Figure 5.1, for memory availability, m1062 is the optimal choice. For energy consumption and execution time, m1146 has the lowest value and provides the optimal choice. For CPU usage, m1 has the lowest observed values in that category. The order of the ranking alternatives in CPU usage, energy consumption, and execution time is not in order of smallest to largest. The order of the ranking alternatives in memory availability is not ranked largest to smallest. These outcomes verify that the general use case outputs a nondominant solution set and the weights are translated as expected in the optimization model.

Table 5.3: Optimization Results Task 108145

$$w^{+}_{MA} = 0.2, w^{-}_{ExC} = 0.2, w^{-}_{EnC} = 0.2, w^{-}_{R} = 0.2, w_{CPU}{}^{-} = 0.2$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank
m1012	1503239	0.001	412	816.86	21.50	fog	1
m1062	1778804	0.001	504	859.85	21.50	cloud	0.9
m1079	1255775	0.001	245	1031.82	21.50	cloud	0.8
m1146	1706243	0.001	1568	739.98	15.74	cloud	0.7
m1	1373635	0.001	95	1254.00	33.00	fog	0.6
m1070	763783	0.001	830	838.35	21.50	cloud	0.5
m1141	772172	0.001	944	988.83	21.50	cloud	0.4
m1061	1214251	0.001	1929	859.85	21.50	cloud	0.3
m1072	1716309	0.002	1731	1419.00	33.00	cloud	0.2
m113	1224946	0.001	5078	945.83	21.50	cloud	0.1

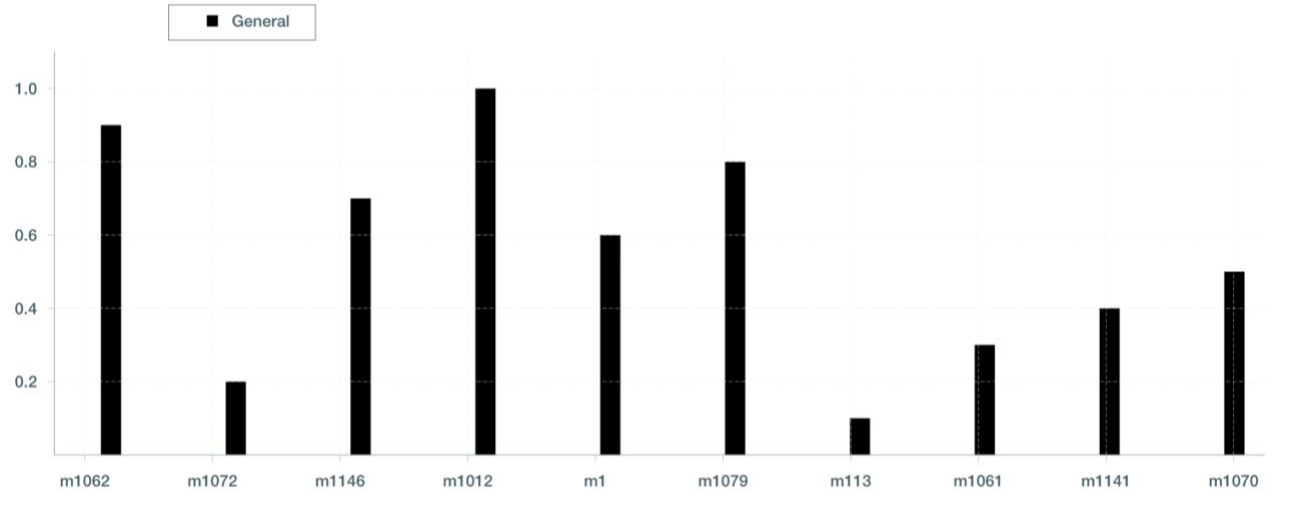

Figure 5.1 Rank of Top 10 Resources Selected for Task 108145

The values from energy consumption, cost and estimate execution time in Table 5.3 are derived from our energy , cost and runtime prediction models. To derive the predicted runtime for m1012, we take the number of CPUs that the virtual machine has and compare it to the number of CPUs that the Task 108145 ran on, it should be noted that all of the tasks in the Auvergrid dataset ran on single CPU machines as noted in Section 3.1. m1012 contains 2 CPUs, given the p value of 69.2% for Task 108145, we calculate the speedup factor to be 1.53. Using Equation 5.3 if we divide 33 seconds (observed runtime) by 1.53 (speedup factor) and get approximately 21.5 seconds. The cost is derived from multiplying the cost basis from Table 3.2 and the energy consumption model is dependent on the predicted runtime multiplied by the base TDP that is noted in Appendix A for each CPU type.

Based on the results presented in Figure 5.1 and Table 5.3, m1012, the top selected resource to offload Task 108145, is predicted to perform the task in 21.5 seconds as compared to the original runtime of 33 seconds which is about a 35% reduction in runtime. Furthermore, m1012 performs the task with an estimated energy consumption of 816.86 Joules which is a 71% reduction as compared to an energy consumption of 2,838 Joules. The cost of the task on m1012 is reduced by 50% from 2/100 of a penny to 1/100 of a penny. Our analysis of the performance score indicates that m1012 is an optimal resource for executing the offloaded task with performance gains, which confirms that our hypothesis in Section 1 is true.

5.2 Memory Driven Use Case

In the memory driven use case we begin with assigning a weight of 1 to memory availability and assign weights of 0 at all other attributes. We will then transition the weight dominance from memory availability to execution cost and observe the outcomes. The weight treatments we'll explore are:

a. $\boldsymbol{w^{+}_{MA} = 1, w^{-}_{ExC} = 0, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0}$
b. $\boldsymbol{w^{+}_{MA} = 0.7, w^{-}_{ExC} = 0.3, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0}$
c. $\boldsymbol{w^{+}_{MA} = 0.3, w^{-}_{ExC} = 0.7, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0}$

Considering Task ID 224334 with the attributes in Table 5.4. The task is currently being executed in the fog layer, it currently costs about $0.12, will use 187,222 Joules of energy, 1,494,608 KB of memory and will run for 2,177 seconds which is about 36 minutes.

Table 5.4: Task ID 224334 Specifications

Used Memory	1494608 KB	Location	fog
Cost	$0.122	Run Time	2177 seconds
Energy Consumption	187222 J	CPU Type	corei7-6700K

We will observe the outputs of the OpERA framework while we transition the weights. We provide details of Task 224334's weights and preferences to the OpERA framework. The output of the results are displayed in Table 5.5. The outcomes reinforce our expectation that top ranking resource m1062 which has the highest memory availability. Since the weight has been skewed to

favor memory availability the resulting outcome is a dominant set. As we transition the weight dominance away from memory availability, we should see these rankings change.

Table 5.5: Optimization Results Task 224334

$$w^{+}_{MA} = 1, w^{-}_{ExC} = 0, w^{-}_{EnC} = 0, w^{-}_{PR} = 0, w_{CPU}{}^{-} = 0$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank
m1062	1778804	0.091	504	65505.93	1637.65	cloud	1
m113	1775868	0.091	3360	72056.52	1637.65	cloud	0.9
m1143	1717148	0.061	16	87080.00	2177.00	fog	0.8
m1072	1716309	0.122	1731	93611.00	2177.00	cloud	0.7

m1146	1706243	0.076	1568	64294.70	1367.97	cloud	0.6
m1	1644587	0.061	347	82726.00	2177.00	fog	0.5
m1144	1623405	0.061	15	71841.00	2177.00	fog	0.4
m1038	1572025	0.076	2289	56086.87	1367.97	cloud	0.3
m1070	1516241	0.091	113	63868.28	1637.65	cloud	0.2
m1012	1503239	0.046	412	62230.63	1637.65	fog	0.1

Observed from Table 5.6, as we transition the weight to execution cost we notice that m1143 is chosen at the top ranking resource. Even though m1143 does not have the highest memory availability, its cost has decreased by 33% as compared to a cost usage of 0.091 with m1062 when the feasible solution set was memory dominant. The second choice with this weight treatment is m1 which has the same cost as the third choice m1144 but within the same cost bracket it is still ordering memory from highest available to lowest memory availability since this set is more dominated by memory availability than it is cost.

Table 5.6: Optimization Results Task 224334

$$w^{+}_{MA} = 0.7, w^{-}_{ExC} = 0.3, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank
m1143	1717148	0.061	16	87080.00	2177.00	fog	1

m1	1644587	0.061	347	82726.00	2177.00	fog	0.9
m1144	1623405	0.061	15	71841.00	2177.00	fog	0.8
m1012	1503239	0.046	412	62230.63	1637.65	fog	0.7
m1146	1706243	0.076	1568	64294.70	1367.97	cloud	0.6
m1038	1572025	0.076	2289	56086.87	1367.97	cloud	0.5
m1062	1778804	0.091	504	65505.93	1637.65	cloud	0.4
m113	1775868	0.091	3360	72056.52	1637.65	cloud	0.3
m1070	1516241	0.091	113	63868.28	1637.65	cloud	0.2
m1072	1716309	0.122	1731	93611.00	2177.00	cloud	0.1

As we transition the weight to cost reduction we can observe in Table 5.7 that the top chosen alternative, m1012, has the lowest cost. Furthermore, the next highest ranking group of alternatives have the second lowest value for cost and descending order of highest to lowest for memory availability.

Table 5.7: Optimization Results Task 224334

$$w^{+}_{MA} = 0.3, w^{-}_{ExC} = 0.7, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank
m1012	1503239	0.046	412	62230.63	1637.65	fog	1

m1143	1717148	0.061	16	87080.00	2177.00	fog	0.9
m1	1644587	0.061	347	82726.00	2177.00	fog	0.8
m1144	1623405	0.061	15	71841.00	2177.00	fog	0.7
m1146	1706243	0.076	1568	64294.70	1367.97	cloud	0.6
m1038	1572025	0.076	2289	56086.87	1367.97	cloud	0.5
m1062	1778804	0.091	504	65505.93	1637.65	cloud	0.4
m113	1775868	0.091	3360	72056.52	1637.65	cloud	0.3
m1070	1516241	0.091	113	63868.28	1637.65	cloud	0.2
m1072	1716309	0.122	1731	93611.00	2177.00	cloud	0.1

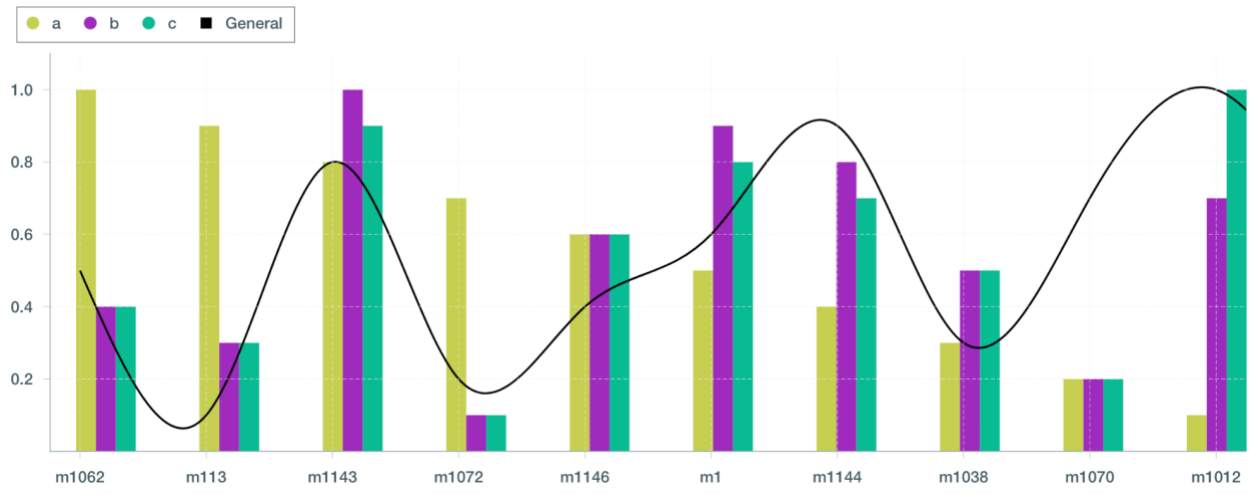

Figure 5.2 Rank of Top 10 Resources Selected for Task 224334 With Varying Weight Treatments

Based on the results, presented in Table 5.5-5.7 and Figure 5.2, the top selected resources to offload Task 224334 are m1062, m1143, and m1012. All three of these resources possess the memory required to perform Task 224334. As shown in Tables 5.5-5.7, offloading Task 224334 on any of alternatives produces a reduction in cost, energy consumption, and runtime. However, m1012 achieves the most gains in all three categories; m1012 reduces the runtime by shortening it by 25% from 2177 seconds to 1637.65 seconds. Furthermore, as shown in Table 5.4 and Table 5.7, offloading Task 224334 to m1012 also reduces the energy consumption by 67% and cost by 62%. The performance score analysis indicates that all the top 10 chosen resources, especially, m1062, m1143, and m1012 are optimal resources for executing the offloaded task with a performance boost, which supports our hypothesis described in Section 1.

5.3 Cost Savings Driven Use Case

In the cost savings driven use case we begin with assigning a weight of 1 to cost and assign weights of 0 at all other attributes. We will then transition the weight dominance from cost to CPU usage and observe the outcomes. The weight treatments we'll explore will be:

a. $\boldsymbol{w^{+}_{MA} = 0, w^{-}_{ExC} = 1, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0}$

b. $\boldsymbol{w^{+}_{MA} = 0, w^{-}_{ExC} = 0.6, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0.4}$

c. $\boldsymbol{w^{+}_{MA} = 0, w^{-}_{ExC} = 0.3, w^{-}_{EnC} = 0, w^{-}_{R} =, w_{CPU}{}^{-} = 0.7}$

Considering Task ID 318037 with the attributes in Table 5.8. The task is currently being executed in the cloud layer, it currently costs about $6.31, will use 7,717,828 Joules of energy, 70,104 KB of memory and will run for 112,998 seconds which is about 31.39 hours.

Table 5.8: Task ID 318037 Specifications

Used Memory	70104 KB	Location	cloud
Cost	$6.311	Run Time	112998 seconds
Energy Consumption	7717828 J	CPU Type	corei7-6700K

We first analyze the outcomes of the weight distribution skewed towards cost. As we can see from Table 5.9, OpERA framework has placed m1015 at the top of the result set which has the lowest cost. The results in the result set are all cost dominant in ranking from the lowest cost to highest cost alternatives being at the end.

Table 5.9: Optimization Results Task 318037

$$w^{+}_{MA} = 0, w^{-}_{ExC} = 1, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank
m1015	1089470	1.555	340	1188106.17	74256.64	edge	1
m1001	1144206	2.074	267	2673238.89	74256.64	fog	0.7
m106	717436	2.074	439	2896008.79	74256.64	fog	0.7
m111	948122	2.074	13	2450468.98	74256.64	fog	0.7

m1007	1684013	2.074	634	2673238.89	74256.64	fog	0.7
m1121	1088422	2.074	702	2524725.61	74256.64	fog	0.7
m1061	1409706	4.147	124	2970265.43	74256.64	cloud	0.3
m1033	1560700	4.147	1955	2747495.52	74256.64	cloud	0.3
m113	1426273	4.147	3563	3267291.97	74256.64	cloud	0.3
m1123	1426483	6.311	841	5084910.00	112998.00	cloud	0.1

As we transition the weights dominance away from cost and towards CPU usage minimization, the m111 is selected at the top resource. The second best choice is m1015 which is a cheaper alternative but as shown in Table 5.10 the reduction in CPU usage is almost 2600% as compared to an optimization loss in cost of 25%.

Table 5.10: Optimization Results Task 318037

$$w^{+}_{MA} = 0, w^{-}_{ExC} = 0.6, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0.4$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank
m111	948122	2.074	13	2450468.98	74256.64	fog	1
m1015	1089470	1.555	340	1188106.17	74256.64	edge	0.9
m1001	1144206	2.074	267	2673238.89	74256.64	fog	0.8

m106	717436	2.074	439	2896008.79	74256.64	fog	0.7
m1007	1684013	2.074	634	2673238.89	74256.64	fog	0.6
m1121	1088422	2.074	702	2524725.61	74256.64	fog	0.5
m1061	1409706	4.147	124	2970265.43	74256.64	cloud	0.4
m1123	1426483	6.311	841	5084910.00	112998.00	cloud	0.3
m1033	1560700	4.147	1955	2747495.52	74256.64	cloud	0.2
m113	1426273	4.147	3563	3267291.97	74256.64	cloud	0.1

As we shift further over to CPU usage minimization weighted, m111 is still selected as the most optimal resource for offloading and the feasible solution set becomes heavy CPU usage minimization dominant as shown in Table 5.11.

Table 5.11: Optimization Results Task 318037

$$w^{+}_{MA} = 0, w^{-}_{ExC} = 0.3, w^{-}_{EnC} = 0, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0.7$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank
m111	948122	2.074	13	2450468.98	74256.64	fog	1
m1001	1144206	2.074	267	2673238.89	74256.64	fog	0.9

m1015	1089470	1.555	340	1188106.17	74256.64	edge	0.8
m1061	1409706	4.147	124	2970265.43	74256.64	cloud	0.7
m106	717436	2.074	439	2896008.79	74256.64	fog	0.6
m1007	1684013	2.074	634	2673238.89	74256.64	fog	0.5
m1121	1088422	2.074	702	2524725.61	74256.64	fog	0.4
m1123	1426483	6.311	841	5084910.00	112998.00	cloud	0.3
m1033	1560700	4.147	1955	2747495.52	74256.64	cloud	0.2
m113	1426273	4.147	3563	3267291.97	74256.64	cloud	0.1

From use cases 5.2 and 5.3 we can observe and verify that the optimization framework is able to choose the optimal solution with mixed proportion weights or a single dominant weight. MCDM comes into play for mixing often opposing attributes to produce the optimal outcomes in a nondominated set.

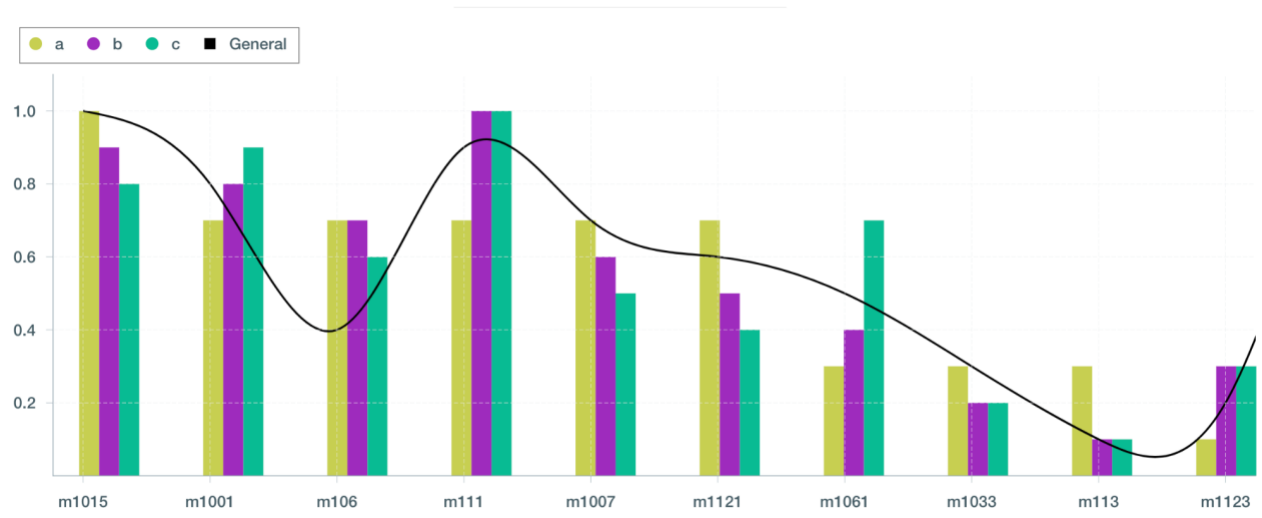

Figure 5.3 Rank of Top 10 Resources Selected for Task 318037 With Varying Weight Treatments

Based on the results, presented in Table 5.9-5.11 and Figure 5.3, the top selected resources to offload Task 318037 are m111 and m1015. Both of these resources possess the memory required to perform Task 318037. As shown in Tables 5.9-5.11, offloading Task 318037 on any of alternatives produces a reduction in cost, energy consumption, and runtime. However, m1015 achieves the most gains in all three categories; m1015 reduces the runtime by shortening it by 34% from a little over 31 hours to a little over 20 hours. Furthermore, as shown in Table 5.8 and Table 5.9, offloading Task 318037 to m1015 also reduces the energy consumption by 85% and cost by 75% from $6.31 to $1.55. Our analysis of the performance score indicates that all 10 resources chosen for all of the weight assignments, especially, m1015 is an optimal resource for performing the offloaded task with performance gains, which is in line with our hypothesis in Section 1.

5.4 Compute Driven Use Case

In a compute driven use case we will explore memory availability, CPU usage and energy consumption attributes. We begin with assigning a weight of 0.495 to memory availability, 0.495 to CPU usage, 0.01 to energy consumption and assign weights of 0 at all other attributes. We will then transition the weight dominance between memory availability, CPU usage and energy consumption while observing the outcomes. The weight treatments we'll explore will be:

a. $\boldsymbol{w^+_{MA} = 0.495, w^-_{ExC} = 0, w^-_{EnC} = 0.01, w^-_R = 0, w_{CPU}{}^- = 0.495}$
b. $\boldsymbol{w^+_{MA} = 0.495, w^-_{ExC} = 0, w^-_{EnC} = 0.495, w^-_R = 0, w_{CPU}{}^- = 0.01}$
c. $\boldsymbol{w^+_{MA} = 0.01, w^-_{ExC} = 0, w^-_{EnC} = 0.495, w^-_R =, w_{CPU}{}^- = 0.495}$

Considering Task ID 334016 with the attributes in Table 5.12. The task is currently being executed in the cloud layer, it currently costs about $0.19, will use 294,034 Joules of energy, 732,832 KB of memory and will run for 3,419 seconds which is about 56 minutes.

Table 5.12: Task ID 334016 Specifications

Used Memory	732832 KB	Location	cloud
Cost	$0.191	Run Time	3419 seconds
Energy Consumption	294034 J	CPU Type	AMD-3201

We first analyze the outcomes of the weight distribution skewed towards memory usage and CPU usage. As we can see from Table 5.13, OpERA framework has placed m106 at the top of the result set which has the highest memory availability and the second lowest CPU usage. The weight added to the energy consumption did not factor into this decision much since the third energy depleting resource also accounts for the third highest cost.

Table 5.13: Optimization Results Task 334016

$$w^{+}_{MA} = 0.495, w^{-}_{ExC} = 0, w^{-}_{EnC} = 0.01, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0.495$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank
m106	1773142	0.063	40	88078.40	2258.42	fog	1

m1084	1363358	0.063	377	74527.87	2258.42	fog	0.9
m1001	1144206	0.063	267	81303.14	2258.42	fog	0.8
m1007	1684013	0.063	634	81303.14	2258.42	fog	0.7
m111	948122	0.063	13	74527.87	2258.42	fog	0.6
m1015	1089470	0.047	340	36134.73	2258.42	edge	0.5
m1123	1426483	0.191	841	153855.00	3419.00	cloud	0.4
m1121	1088422	0.063	702	76786.30	2258.42	fog	0.3
m1033	1560700	0.126	1955	83561.56	2258.42	cloud	0.2
m113	1426273	0.126	3563	99370.50	2258.42	cloud	0.1

As we transition the weights dominance away from CPU usage and towards energy consumption minimization, m1015 is selected as the top resource. Even though m1015 does not have the highest memory availability, the loss in memory availability as compared to the second choice is about 36%. As shown in Table 5.14, the savings in energy consumption account for more than a 56% reduction as compared to the second choice m1007.

Table 5.14: Optimization Results Task 334016

$$w^{+}_{MA} = 0.495, w^{-}_{ExC} = 0, w^{-}_{EnC} = 0.495, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0.01$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank

m1015	1089470	0.047	340	36134.73	2258.42	edge	1
m1007	1684013	0.063	634	81303.14	2258.42	fog	0.9
m1084	1363358	0.063	377	74527.87	2258.42	fog	0.8
m106	1773142	0.063	40	88078.40	2258.42	fog	0.7
m1033	1560700	0.126	1955	83561.56	2258.42	cloud	0.6
m1121	1088422	0.063	702	76786.30	2258.42	fog	0.5
m1001	1144206	0.063	267	81303.14	2258.42	fog	0.4
m111	948122	0.063	13	74527.87	2258.42	fog	0.3
m113	1426273	0.126	3563	99370.50	2258.42	cloud	0.2
m1123	1426483	0.191	841	153855.00	3419.00	cloud	0.1

As we shift back to CPU usage and away from memory availability, Table 5.15 shows that m1015 is still selected as the most optimal resource for offloading. Even though the weight is split evenly between cost and CPU usage, the feasible solution becomes more weighted towards energy consumption.

Table 5.15: Optimization Results Task 334016

$$w^{+}_{MA} = 0.01, w^{-}_{ExC} = 0, w^{-}_{EnC} = 0.495, w^{-}_{R} = 0, w_{CPU}{}^{-} = 0.495$$

VM	Available Memory	Cost	CPU Usage	Energy Consumption	Estimated Execution Time	Location	Rank

m1015	1089470	0.047	340	36134.73	2258.42	edge	1
m111	948122	0.063	13	74527.87	2258.42	fog	0.9
m1084	1363358	0.063	377	74527.87	2258.42	fog	0.8
m1001	1144206	0.063	267	81303.14	2258.42	fog	0.7
m106	1773142	0.063	40	88078.40	2258.42	fog	0.6
m1007	1684013	0.063	634	81303.14	2258.42	fog	0.5
m1121	1088422	0.063	702	76786.30	2258.42	fog	0.4
m1123	1426483	0.191	841	153855.00	3419.00	cloud	0.3
m1033	1560700	0.126	1955	83561.56	2258.42	cloud	0.2
m113	1426273	0.126	3563	99370.50	2258.42	cloud	0.1

By employing OpERA on a variety of edge based operation types we've demonstrated that the performance of the optimization strategy for most performance thresholds can achieve scores of at least 30%. Users can use this framework to select the optimal set of resources for tasks offloading and experiment with different combinations of weights to determine which composition would optimize resource allocation. The evaluation of the test cases confirmed the hypothesis that by using a resource allocation optimization model, we can identify resources to offload a task with gains in performance.

Figure 5.4 Rank of Top 10 Resources Selected for Task 334016 With Varying Weight Treatments

Based on the results, presented in Table 5.13-5.15 and Figure 5.4, the top selected resources to offload Task 334016 are m106 and m1015. Both of these resources possess the memory required to perform Task 334016. As shown in Tables 5.9-5.11, offloading Task 334016 on any of alternatives produces a reduction in cost, energy consumption, and runtime. However, m1015 achieves the most gains in all three categories; m1015 reduces the runtime by shortening it by 34% from a little over 56 minutes to a little over 37 minutes. Furthermore, as shown in Table 5.12 and Table 5.14, offloading Task 334016 to m1015 also reduces the energy consumption by 88% from 294034 Joules to 36134 Joules and cost by 76%. Albeit, all of the top alternatives in each category are optimal resources for executing the offloaded task, m1015 and m106, are the top performant resources with performance gains, which stand to reinforce the hypothesis in Section 1 is true.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

OpERA, an offloading framework that allows for optimized resource allocation which can be implemented across a wide range of IoT systems that use a multi-objective decision making methodology. In addition, OpERA is able to navigate multiple computing layers to access resources for an optimization decision making matrix. This feature was demonstrated through the application of an optimization strategy based on algorithms and prediction models.

We investigated the application of an MCDM optimization method TOPSIS to task offloading, more specifically the aspect of resource allocation for task offloading. The problem area we addressed is the lack of optimized offloading strategies for edge based task offloading, reliance on the cloud for task offloading is not optimal for all use cases such as compute centric and time sensitive devices. There is a need to develop an optimization strategy for resource allocation as the sophistication and computation complexity of applications grow in the IoT realm. We developed OpERA, a resource allocation optimization model that can ingest and calculate attribute metrics from available resources in multi-layer environments. We created a framework that allows users to deduce and choose an alternative with ample resources to execute the task.

The contributions of OpERA and the research are as follows:

- An edge-based resource allocation optimization strategy is investigated to optimize task offloading.
- We create OpERA, a multi-objective optimization model that combines runtime prediction, cost derivation, and energy consumption objectives.

- A series of experiments were conducted to determine the usefulness of multi-objective optimization strategies such as TOPSIS MCDM methods.
- Using real world clustering datasets, we evaluate the performance of the OpERA optimization model.
- For the purpose of offloading operations, we provide insights into the feasibility of MCDM methods.

The work presented above provides a strong case for further consideration of resource allocation optimization for task offloading. The utilization of a MCDM optimization model for task offloading provides a niche opportunity to address edge resource allocation impartial to the offloading strategy as shown in Figure 1.1.

6.2 Future Work

To further improve upon the optimization framework the use of manual verification should be replaced with a statistical measure. Furthermore, to more closely reflect offloading strategies for highly compute centric workloads such as DNN workloads, trace data that includes GPU should be considered. The use of GPU in a workload would require benchmarking in order to predict speedup factors for predicted runtime. The employment of a more complex energy dissipation model can be used in order to consider energy dissipation with data transfer and other factors. The trace data contained 13.9 million traces which often varied slightly for each metric attribute, the use of a graph structure such as an entity graph, that can greatly characterize and detect the small differences but increase the selection of the decision matrix.

References

[1] Jiang, Congfeng, Xiaolan Cheng, Honghao Gao, Xin Zhou, and Jian Wan. ”Toward computation offloading in edge computing: A survey.” IEEE Access 7 (2019): 131543-131558.

[2] D. Majumder, S. M. Kumar, D. V. Ashoka and A. S. Nargunam, ”Resource Allocation Techniques in Edge/Fog Computing,” 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), 2021.

[3] Studying and developing a resource allocation algorithm in Fog computing by Nguyen Quang-Hung, Truong Pham Thanh An, 2018 International Conference on Advanced Computing and Applications (ACOMP).

[4] Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms by CHEOL-HO HONG and BLESSON VARGHESE, ACM Computing Surveys, Vol. 1, No. 1, Article 1, Publication date: September 2019.

[5] Aazam, Mohammad , S. Zeadally , and K. A. Harras . ”Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities.” Future Generation Computer Systems (2018): S0167739X18301973.

[6] A. Ealiyas and S. P. Jeno Lovesum, ”Resource Allocation and Scheduling Methods in Cloud-A Survey,” 2018 Second International Conference on Computing Methodologies and Communication (ICCMC), 2018.

[7] Satyanarayanan, Mahadev. ”A Brief History of Cloud Offload.” Acm Sigmobile Mobile Computing Communications Review 18.4(2015):19-23.

[8] Z. Wu and H. Chen, "Design and Implementation of TCP/IP Offload Engine System over Gigabit Ethernet," Proceedings of 15th International Conference on Computer Communications and Networks, 2006, pp. 245-250.

[9] E. Al-Masri, "An Edge-Based Resource Allocation Optimization for the Internet of Medical Things (IoMT)," 2021 IEEE 3rd Eurasia Conference on Biomedical Engineering, Healthcare and Sustainability (ECBIOS), 2021, pp. 143-147.

[10]Kaxiras S. and Martonosi M.. 2008. Computer Architecture Techniques for Power-Efficiency. DOI: DOI: https://doi.org/10.2200/S00119ED1V01Y200805CAC004

[11] “Pricing Calculator: Microsoft Azure.” Pricing Calculator | Microsoft Azure. Accessed March 12, 2022. https://azure.microsoft.com/en-us/pricing/calculator/.

[12] Details of Grid Workload Archive, www.st.ewi.tudelft.nl/~iosup/project_grid_gwa.html. Accessed February 12, 2022.

[13] Details of AuverGrid workload, gwa.ewi.tudelft.nl/datasets/gwa-t-4-auvergrid. Accessed February 12, 2022.

[14] Details of Materna workload, gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna. Accessed February 12, 2022.

[15] Null, Linda, and Julia Lobur. 2018. The essentials of computer organization and architecture. Sudbury, Mass: Jones and Bartlett Publishers.

[16] “Mathematical Optimization for Business Problems” Mathematical Optimization for Business Problems. Accessed March 3, 2022. https://cognitiveclass.ai/courses/mathematical-optimization-for-business-problems.

[17] Hwang, Ching-Lai, Young-Jou Lai, and Ting-Yun Liu. "A new approach for multiple objective decision making." Computers & operations research 20, no. 8, 1993.

[18] Şahin, Mehmet. "A comprehensive analysis of weighting and multicriteria methods in the context of sustainable energy." International Journal of Environmental Science and Technology 18, no. 6, 2021.

[19]Mach, Pavel, and Zdenek Becvar. "Mobile edge computing: A survey on architecture and computation offloading." IEEE Communications Surveys & Tutorials 19, no. 3 2017.

[20] Rodriguez, Maria Alejandra, and Rajkumar Buyya. "A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments." Concurrency and Computation: Practice and Experience 29, no. 8, 2017.

[21] W. -Z. Zhang et al., "Secure and Optimized Load Balancing for Multitier IoT and Edge-Cloud Computing Systems," in IEEE Internet of Things Journal, vol. 8, no. 10, pp. 8119-8132, 15 May 15, 2021

[22]Sun, Zhenfeng, and Mohammad Reza Nakhai. "An online learning algorithm for distributed task offloading in multi-access edge computing." IEEE Transactions on Signal Processing 68, 2020.

[23] H. Guo, J. Liu and J. Zhang, "Computation Offloading for Multi-Access Mobile Edge Computing in Ultra-Dense Networks," in IEEE Communications Magazine, vol. 56, no. 8, pp. 14-19, August 2018.

[24] Tran, Tuyen X., and Dario Pompili. "Joint task offloading and resource allocation for multi-server mobile-edge computing networks." IEEE Transactions on Vehicular Technology 68.1, 2018.

[25] F. Wei, S. Chen and W. Zou, "A greedy algorithm for task offloading in mobile edge computing system," in China Communications, vol. 15, no. 11, pp. 149-157, Nov. 2018.

[26] Islam, Akhirul, Arindam Debnath, Manojit Ghose, and Suchetana Chakraborty. "A survey on task offloading in multi-access edge computing." Journal of Systems Architecture 118 (2021).

[27] Hou, Xiangwang, Zhiyuan Ren, Jingjing Wang, Wenchi Cheng, Yong Ren, Kwang-Cheng Chen, and Hailin Zhang. "Reliable computation offloading for edge-computing-enabled software-defined IoV." IEEE Internet of Things Journal 7, no. 8, 2020.

[28] Khayyat, Mashael, Ibrahim A. Elgendy, Ammar Muthanna, Abdullah S. Alshahrani, Soltan Alharbi, and Andrey Koucheryavy. "Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks." IEEE Access 8, 2020.

[29] Lee, Seung-seob, and SuKyoung Lee. "Resource allocation for vehicular fog computing using reinforcement learning combined with heuristic information." IEEE Internet of Things Journal 7, no. 10, 2020.

Appendix A - CPU Hardware Specification

Model	Family	Base TDP (W)	Max TDP (W)	CPU Base Freq.	CPU Max Freq.
corei7-6700K	6th Generation Intel® Core™ i7 Processors	88	91	4.00 GHz	4.20 GHz
AMD-3451	AMD EPYC™ Embedded Processors	85	90	2.14 GHz	3 GHz
AMD-3351	AMD EPYC™ Embedded Processors	70	70	1.9 GHz	3 GHz
corei7-6700	6th Generation Intel® Core™ i7 Processors	64	65	3.40 GHz	4.00 GHz
corei7-6785R	6th Generation Intel® Core™ i7 Processors	61	65	3.30 GHz	3.90 GHz
D-1513N	Intel® Xeon® D Processor	59	60	2.20 GHz	3.00 GHz
AMD-3251	AMD EPYC™ Embedded Processors	50	55	2.5 GHz	3.1 GHz
D-2123IT	Intel® Xeon® D Processor	52	55	2.90 GHz	3.20 GHz
AMD-3151	AMD EPYC™ Embedded Processors	42	45	2.7 GHz	2.9 GHz
corei7-6700HQ	6th Generation Intel® Core™ i7 Processors	42	45	2.60 GHz	3.50 GHz
corei7-6820HK	6th Generation Intel® Core™ i7 Processors	45	45	2.70 GHz	3.60 GHz
corei7-6820HQ	6th Generation Intel® Core™ i7 Processors	43	45	2.70 GHz	3.60 GHz
corei7-6920HQ	6th Generation Intel® Core™ i7 Processors	40	45	2.90 GHz	3.80 GHz
corei7-6820EQ	6th Generation Intel® Core™ i7 Processors	43	45	2.80 GHz	3.50 GHz
corei7-6970HQ	6th Generation Intel® Core™ i7 Processors	40	45	2.80 GHz	3.70 GHz
corei7-6870HQ	6th Generation Intel® Core™ i7 Processors	45	45	2.70 GHz	3.60 GHz
corei7-6770HQ	6th Generation Intel® Core™ i7 Processors	41	45	2.60 GHz	3.50 GHz
AMD-3255	AMD EPYC™ Embedded Processors	38	40	2.5 GHz	3.1 GHz
AMD-3101	AMD EPYC™ Embedded Processors	35	35	2.1 GHz	2.9 GHz
corei7-6700T	6th Generation Intel® Core™ i7 Processors	33	35	2.80 GHz	3.60 GHz
corei7-6700TE	6th Generation Intel® Core™ i7 Processors	30	35	2.40 GHz	3.40 GHz

Appendix B - Plots of Optimization Rank of Additional Tasks

Table B.1 Task ID 160573

Used Memory	43152 KB	Location	edge
Cost	$0.007	Run Time	129 seconds
Energy Consumption	11094 J	CPU Type	AMD-3201

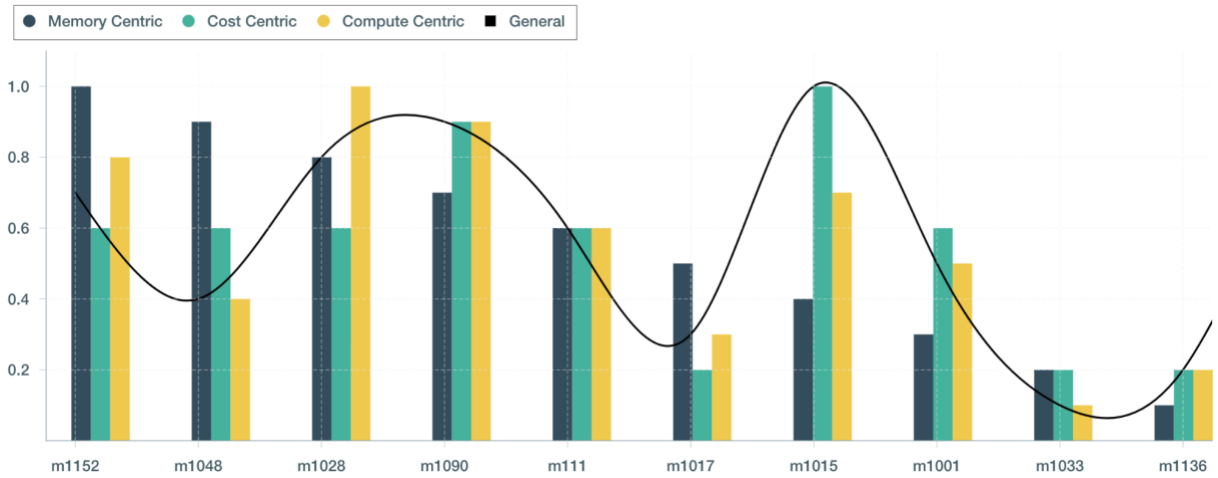

Figure B.1 Top 10 Alternatives For Task ID 160573 With Various Preference Weights

Table B.2 Task ID 401503

Used Memory	1730832 KB	Location	edge
Cost	$0.106	Run Time	5068 seconds
Energy Consumption	126700 J	CPU Type	AMD-3201

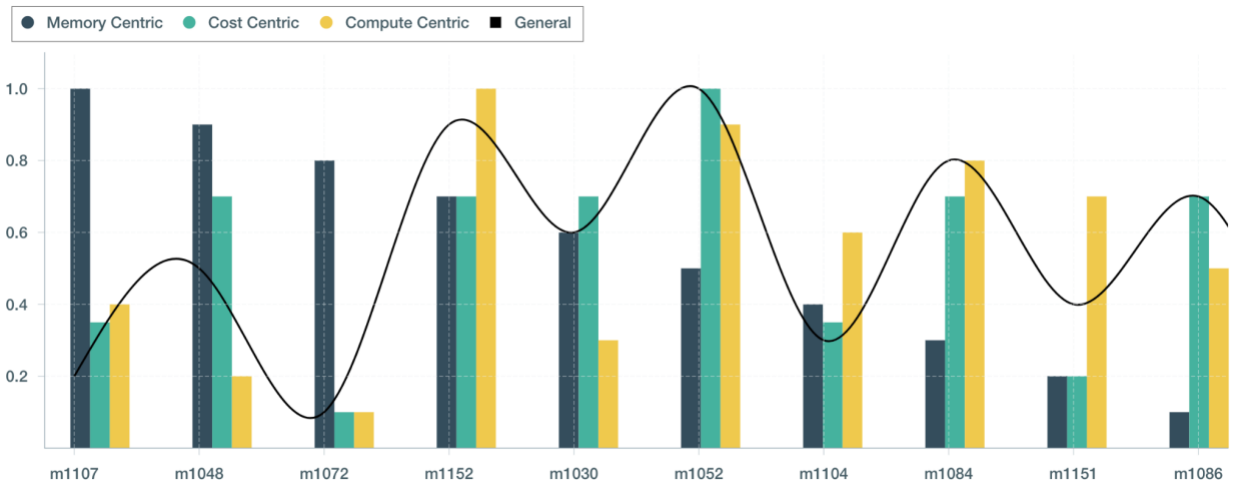

Figure B.2 Top 10 Alternatives For Task ID 401503 With Various Preference Weights

Table B.3 Task ID 39808

Used Memory	582788 KB	Location	fog
Cost	$0.183	Run Time	6568 seconds
Energy Consumption	269288 J	CPU Type	corei7-6820HK

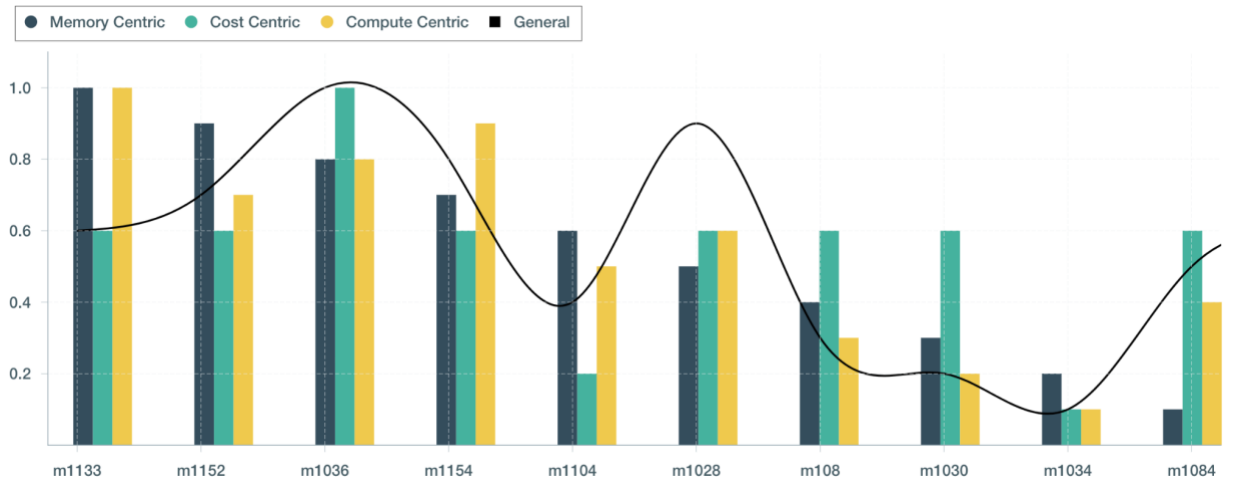

Figure B.3 Top 10 Alternatives For Task ID 39808 With Various Preference Weights