FogWeaver: A Multi-Objective Optimization Strategy and Characterization of a Hybrid Internet of Things (IoT) Environment

James Olmsted

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in Computer Science and Systems

University of Washington

2020

Committee:

Eyhab Al-Masri, Chair

Orlando Baiocchi

Wei Cheng

Program Authorized to Offer Degree:
Computer Science and Systems

University of Washington

Abstract

FogWeaver: A Multi-Objective Optimization Strategy and Characterization of a Hybrid Internet of Things (IoT) Environment

James Olmsted

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

As the complexity and requirements of Internet of Things (IoT) systems evolve, there comes a need for adaptable optimization strategies that improve the process of task allocation, particularly across hybrid fog infrastructures. In this thesis, we introduce FogWeaver, an optimization strategy for efficiently allocating tasks across hybrid fog environments. FogWeaver employs a multi-objective optimization technique to create a metric of comparison between fog- and cloud-based environments while adapting to the needs of an IoT application. By identifying application requirements, it is then possible to determine the necessary resources required for completing computational tasks. To this extent, FogWeaver is designed such that it considers all available computational nodes that exist across both fog and cloud environments making it suitable for IoT applications that can operate within hybrid environments. In addition, FogWeaver employs an adaptable scoring method based on the particle swarm optimization technique to identify an optimal strategy for the allocation of fog- and cloud-based computational resources. Optimizing the allocation of resources helps reduce the costs associated with the execution of tasks or operations across fog environments. In addition, there may exist a number of distributed nodes across a fog environment that can perform a particular task. Hence, as part of our optimization strategy, we attempt to identify a number of computational nodes that can form a supernode that can perform a

particular task. By considering the computational capabilities of fog nodes, it is then possible to enhance the resilience and scalability of IoT applications while minimizing the costs associated with running advanced computational tasks such as Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL), among many others.

TABLE OF CONTENTS

Page

List of Figures iii

List of Tables iv

Chapter 1: Introduction 1

Chapter 2: Related Work 7
2.1 Characterization of IoT Environments 7
2.2 Resource Allocation 8
2.3 Optimization Methodology 9

Chapter 3: Hybrid System Model 10
3.1 Device Layer 11
3.2 Fog Layer 12
3.3 Cloud Layer 12
3.4 Cost Model 13

Chapter 4: Optimization Strategy 15
4.1 Metric of Comparison 15
4.2 Performance Thresholds 20
4.3 System Navigation 21
4.4 Implementation Details of Particle Swarm Optimization 22

Chapter 5: Evaluation and Assessment 23
5.1 Difference-Based Threshold 23
5.2 Percentage-Based Threshold 29
5.3 Two-Level Threshold Results 32
5.4 Summary of Optimization Strategy 33

Chapter 6: Conclusion and Future Work 34
6.1 Contributions and Findings of FogWeaver 34
6.2 Future Work 35

Bibliography 37

Appendix A: Additional Data 39

Appendix B: Early Energy Usage Model and Data 45

Appendix C: Code Demonstration 48
C.1 Data Collection 48
C.2 Visual Demonstration 82
C.3 Additional Files 98

Appendix D: Ant Colony Exploration of Optimization Across Hybrid Fog Environments110

LIST OF FIGURES

Figure Number Page

3.1 System Model Layer Diagram 11

4.1 UML Decision Diagram for Task Allocation Process 15

5.1 Percentage Threshold Accuracy Results 30

5.2 Percentage Threshold Cost Results 31

5.3 Two-Level Threshold Accuracy Results 32

LIST OF TABLES

Table Number Page

1.1 IoT Paradigm Comparison 4

3.1 Features Characterizing Computational Node Capabilities 13

5.1 $w_{tc} = 1$ Accuracy Results Execution Time-Driven Test Case: $w_{tc} = 1$ 24
5.2 $w_{tc} = 1$ Cost Results Execution Time-Driven Test Case: $w_{tc} = 1$ 25
5.3 $w_{tc} = 0.5$ Accuracy Results Execution Time and Cost Balance: $w_{tc} = 0.5$ . . 26
5.4 $w_{tc} = 0.5$ Cost Results Execution Time and Cost Balance: $w_{tc} = 0.5$ 27
5.5 $w_{tc} = 0$ Accuracy Results Execution Cost-Driven: $w_{tc} = 0$ 28
5.6 $w_{tc} = 0$ Cost Results Execution Cost Driven: $w_{tc} = 0$ 29

A.1 Metric of Comparison Weights 39
A.2 Particle Swarm Constants 39
A.3 Percentage Threshold $w_{tc} = 1$ Accuracy Results 40
A.4 Percentage Threshold $w_{tc} = 0.5$ Accuracy Results 40
A.5 Percentage Threshold $w_{tc} = 0$ Accuracy Results 41
A.6 Percentage Threshold $w_{tc} = 1$ cost Results 41
A.7 Percentage Threshold $w_{tc} = 0.5$ Cost Results 42
A.8 Percentage Threshold $w_{tc} = 0$ Cost Results 42
A.9 $w_{tc} = 1$ Two-Level Threshold Accuracy Results 43
A.10 $w_{tc} = 0.5$ Two-Level Threshold Accuracy Results 43
A.11 $w_{tc} = 0$ Two-Level Threshold Accuracy Results 44

B.1 Thermal Design Power Values 45
B.2 $w_{tc} = 1$ Energy (KJ) Results 46
B.3 $w_{tc} = 0.5$ Energy (KJ) Results 46
B.4 $w_{tc} = 0$ Energy (KJ) Results 47

ACKNOWLEDGMENTS

I am extremely grateful to my advisor, Eyhab Al-Masri, for his guidance and insight through the research process. Without him I would not have been able to have gotten through this research effort as helped me to guide intuition and illuminate paths of exploration that I had not seen.

I am indebted to my family for their support and being there for me to bounce ideas off of and making sure I stayed in contact with the outside world during the final stages of the research effort.

DEDICATION

To my friends and family, for their support and allowing me to bounce ideas off of them.

Chapter 1

INTRODUCTION

The deployment of Internet of Things (IoT) systems can be achieved through two major paradigms: (a) the cloud environment and (b) the fog environment. In order to fully understand the growing need for a hybrid environment, it is imperative to determine the suitability of the resources available in each environment for the purpose of allocating computational resources when completing computational tasks. That is, by considering the resources that exist in both environments, it is then possible to efficiently determine how to distribute computational resources more efficiently and in an optimal manner. While a majority of IoT systems are heavily dependent on cloud resources, fog computing has emerged in recent years as a suitable alternative that complements that of the cloud.

IoT systems that employ a cloud environment use IoT devices in order to collect and transfer data directly to the cloud without the need for processing data at a local scale. That is, these IoT devices generally do not perform local processing of the data which they collect, allowing for the ability to collect significant amounts of IoT data at a much faster pace. This data is then transferred to the cloud for further storage and processing. Allowing IoT devices to focus primarily on data collection increases the ability for these devices to be mobile and therefore offers a level of flexibility for accessing cloud resources ubiquitously. In addition, The scalability of cloud environments makes them suitable for supporting advanced or complex operations that require significant computational capabilities in terms of processing power, memory, and storage when compared to that of fog environments. However, as more cloud resources are required to perform advanced tasks, IoT systems may suffer from high latency rates which may not be suitable for IoT applications that are sensitive to low-latency.

While cloud environments offer versatile and effective methods for executing computational tasks, the need for processing nodes that reside in close proximity to where IoT devices are located becomes essential. That is, given that IoT data may need to travel long distances via a network, this often leads to latency issues in terms of executing tasks. As a result, when dealing with mission-critical tasks and those that require fast response time, depending solely on cloud environments is insufficient. Additionally, as IoT applications primarily rely on cloud service providers for allocating tasks, the costs associated with the execution of such tasks can be quite expensive.

Fog environments offer a way for IoT systems to execute tasks locally reducing the need for cloud resources. Through fog computing, IoT devices not only collect data via sensors but also can perform simple operations for processing tasks at a local level. By performing operations in close proximity to data sources, it is then possible to resolve latency issues that are often associated with transferring data to the cloud. This makes fog-based IoT systems extremely efficient for processing real-time operations and ones that require fast response time (e.g. mission-critical systems). Performing data analytics locally on fog environments translates into significant cost reductions compared to cloud environments.

However, the reliance of fog environments on IoT devices or nodes introduces additional challenges. Unlike cloud environments, fog environments often have very limited computational resources. This leads to operations competing for finding available resources in terms of processing, storage, and memory. Mobility also becomes a major challenge across fog environments since tasks need to migrate from one fog environment to another which often leads to interoperability issues. Furthermore, enhancing the fog computational capabilities translates into having additional hardware resources in terms of computational power, memory, and storage within devices placed nearby IoT devices. As a result, the scaling of fog environments becomes difficult due to space limitations, costs, energy consumption, among many others.

What is therefore desirable is to identify a well-balanced approach that can accommodate the strengths of the cloud and fog paradigms while considering their limitations. What we

mean by a hybrid environment is one that provides a balanced approach such that a fog environment can take full advantage of the localized computational capabilities of existing IoT devices while also considering cloud-based resources for complex or advanced computational processing. A hybrid system achieves a high level of maintainability by comparing the computational capabilities of processing nodes across both fog and cloud environments. Hence, it is then possible to identify an optimal computational node or nodes that can process a given task.

Through this hybrid approach, an IoT application can significantly adapt based on its requirements and the availability of existing resources. In addition, this hybrid approach can, for example, execute tasks that require a fast response within a fog environment (e.g. mission-critical tasks) while offloading to the cloud tasks that are less critical or may not be capable of running locally. As a result, this hybrid approach will be capable of maximizing the resources employed at the fog level while also minimizing costs and latency rate. Using this hybrid environment, it is then possible to partially complete a subset of tasks at the fog level while offloading more advanced ones to the cloud. However, this hybrid approach prompts the need for optimizing the process of offloading tasks between fog and cloud environments. In this research work, we investigate the usefulness of this hybrid approach propose FogWeaver, an optimization strategy that is capable of determining tasks that can execute at the fog level and those that require offloading to the cloud. Using this hybrid approach, the fog can complement that of the cloud environment while also increasing the resiliency and scalability of IoT applications particularly for ones involving mission-critical use cases.

To illustrate the application of a hybrid environment, consider, for example, an IoT hospital system for processing patient information. Due to the need for having quick access to patient information, this IoT system would be best developed using a fog environment because the latency of the cloud environment would be high which can act as a deterrent or cause delays when processing data involving life-threatening situations. As the number of operations involving patient information increases in this fog-based environment, the system may reach a saturation stage such that all of the available resources are being utilized which

may significantly introduce delays in processing pending or queued operations. In this scenario, the system can either reject processing new tasks, delay pending tasks, or attempt to offload tasks to other environments such as the cloud for further processing. In the case of mission-critical tasks, the system may then need to offload tasks to the cloud which often leads to high monetary costs in terms of cloud-based resources and network usage.

In addition, offloading tasks to the cloud introduces latency overhead depending on the distance between a local system and cloud data centers. Furthermore, the system may run into other complications such as in cases where patient information may need to be transferred between hospitals across heterogeneous environments. Privacy and security also become another major issue pertaining to the transfer of patient data to remote cloud data centers. Therefore, depending solely on fog resources may often lead to exhausting resources when processing data-intensive operations whereas offloading to the cloud all possible tasks often leads to high costs in terms of network and processing power. Therefore, a hybrid approach can optimize the processing of tasks at the local level while reducing the dependence on cloud resources as much as possible. Table 1.1 presents a feature comparison between fog, cloud, and hybrid environments.

Table 1.1: IoT Paradigm Comparison

Factor	**Fog**	**Cloud**	**Hybrid**
Support for Mobility	Low	High	High
Computational Power	Limited	Large	Moderate to Large
Storage	Limited	Large	Moderate to Large
Latency	Minimal	Varies depending on bandwidth and distance travelled	Requires Optimization
Monetary Cost	Minimal	Based on Service	Requires Optimization

Considering the IoT hospital system to be employed using a hybrid approach, we need to first determine the capabilities that exist across both environments, particularly computational capabilities, storage, and network bandwidth as outlined in Table 1.1. By considering the capabilities at the local level, it is then possible to identify tasks that can operate within a fog environment and those that require additional capabilities to be offloaded to the cloud. For example, in the IoT hospital system deployed using a hybrid approach, the IoT system would be able to run tasks in parallel, locally, and on the cloud. The cloud can be utilized for non-mission-critical tasks whereas the fog can execute tasks that require fast execution time. However, to achieve this, there may exist a trade-off in terms of the monetary costs associated with executing tasks locally or on the cloud. That is, there could be some benefits to having a task to be processed on the cloud (e.g. complex artificial intelligence operation) when compared to running it locally such as increased accuracy and execution time. As a result, optimizing the decision-making of when to send tasks to the cloud becomes necessary.

To allow the hybrid environment to operate optimally, we need to develop a strategy that optimizes the task allocation through a hybrid IoT system. To develop such a strategy, it is necessary to identify the factors that determine the performance of a task on a computational node and employ these factors to create a metric of comparison. This strategy needs to adapt to the needs of a system, while maintaining a high level of accuracy and how it allocates tasks. In addition, the strategy needs to be able to minimize the cost, while meeting the performance needs of a system. Finally, the strategy needs to ensure that tasks that are critical to the performance of a system, maintain the necessary performance levels for the system to operate, particularly during the stages under which the strategy is being fine-tuned. To meet these needs we propose FogWeaver: a multi-objective optimization methodology and strategy for task allocation across fog environments.

The main goal of FogWeaver is the ability to adapt to a wide variety of environments and needs while maintaining a well-balanced approach between the efficient handling of tasks, and the monetary costs for running them. Allowing IoT systems to scale dynamically to process complex operations while minimizing costs associated with processing tasks on a

cloud environment, the need for optimizing the allocation of resources at the local level becomes essential. As a result, there may exist a number of distributed resources residing on fog nodes that can form a supernode for completing a task. Hence, this introduces a hierarchy of nodes that can be utilized at the fog level for completing operations that requires significant optimization. In this thesis, we propose FogWeaver that serves as an optimization strategy that can support the formation of hierarchical levels within a fog environment to enhance the allocation of resources more efficiently across fog environments.

The rest of this thesis is organized as follows: Chapter 2 presents related work. Chapter 3 presents the proposed FogWeaver model and its architecture. Chapter 4 discusses the implementation of FogWeaver. Chapter 5 presents the particle swarm navigation model experimental results. Finally, Chapter 6 presents our conclusions and plans for future work.

Chapter 2

RELATED WORK

Optimally allocating resources within cloud environments is an active area of research that has been widely explored across the research community over the past few years. However, little research work has been conducted into the optimization of resources across fog environments [15]. In addition, the characterization and optimization of IoT systems that employ a hybrid approach utilizing both cloud and fog environments are often neglected in research. To this extent, in this section, we identify some of the common research efforts that have been conducted into the characterization and optimization of IoT systems across fog environments.

2.1 Characterization of IoT Environments

There have been numerous research studies that look into the characteristics that affect the performance of an IoT system in both fog and cloud environments [4, 6, 13, 16]. In [13, 16], the authors studied the performance of the fog infrastructure and factors that influence the resource provisioning process. Other research efforts have focused on examining the performance of cloud environments [4, 6]. The authors in [17] further illuminated this aspect by looking at the minimization of latency for a theoretical IoT application with a specific focus on smart city. While the research study in [17] explores the characterization of a hybrid IoT system, it is limited in terms of its applicability to all types of IoT systems. That is, the study focuses primarily on a specific IoT scenario (e.g. smart city) which could not be generalized for all types of IoT systems. In addition, the study lacks granularity in terms of the factors that may influence decision-making such as costs to be considered in the characterization of an IoT environment.

While existing research efforts provide insights into the overlap in terms of the performance factors that exist across both cloud and fog environments, considering the plurality and heterogeneity of computational capabilities that exist across hybrid fog environments is often neglected. In addition, identifying the characteristics that pertain to IoT systems running on cloud and fog environments has not been properly investigated. In this thesis, we will investigate the characteristics that differentiate between tasks that can be executed within cloud and fog environments and determine the feasibility of running tasks across both.

2.2 Resource Allocation

In addition to characterizing the factors of performance across cloud and fog environments, other research studies [4, 6, 13, 16] investigated methods for resource allocation within these environments. The authors in [13] introduced a mechanism for identifying the availability of resources that can be allocated into existing nodes while assigning these tasks appropriately to allocated nodes. The authors in [6] further investigated the use of minimizing deadline violations in the resource allocation process for improving the scheduling of tasks' execution. Another research effort examined the minimization of power consumption to create an energy-saving optimization strategy [4].

While these research efforts [4, 6, 13, 16] provide mechanisms for allocating resources, the granularity level offered by these studies for the decision making is very limited. Given that an IoT system includes many device types of heterogeneous nature, considering the computational capabilities of various device types that exist becomes crucial. In addition, fog and cloud environments may offer comparable resources that can perform tasks. Existing solutions either consider running tasks locally on fog or cloud environments. Little research has been conducted into support for the allocation of resources across both environments altogether. To this extent, some tasks can partially execute on one environment (e.g. cloud) whereas the remaining parts can be executed on another environment (e.g. fog). This research work investigates a fine-grained level of task allocation that is not offered by existing research studies.

2.3 Optimization Methodology

Allocating resources requires identifying the computational capabilities of existing devices or computational nodes to efficiently optimize the workflow of scheduling tasks. Hence, there may exist multiple conflicting objectives when resources are allocated. Generally, this problem can be solved through the use of a multi-objective optimization [2, 8, 11, 14, 20, 21]. There exists a wide range of optimization techniques that can be employed to support resource allocation across hybrid environments. For example, existing research efforts have focused on the use of the particle swarm optimization technique in an attempt to determine an optimal solution for the task allocation process [2, 8, 11, 20]. Other research studies employed a similar method utilizing the ant colony population-based optimization technique to determine an optimal schedule that can be utilized for executing tasks [1, 9, 21]. Other research studies such as the ones in [12, 14, 19] employed genetic algorithm strategies of a time and space-optimized Non-dominated Sorting Genetic Algorithm II (NSGA-II) for comparing response time and storage of cloud resources using a set of heuristics to minimize the execution time across a multi-cloud environment. The notion was extended using a hybrid load utilizing a push-pull algorithm to minimize the execution time across network clusters [12, 14, 19].

While [20] uses a particle swarm algorithm to balance both execution time and cost for task scheduling, the research study focuses primarily on the cloud environment and does not explore the application of the particle swarm to balance between the cost and performance of a fog or a hybrid environment. Unlike existing research efforts, in this thesis, we will investigate the use of optimization techniques to support hybrid fog environments.

Chapter 3
HYBRID SYSTEM MODEL

In this chapter, we introduce FogWeaver, a hybrid system model that can reside on fog nodes or gateways within fog environments. A hybrid IoT system supports the allocation of tasks locally (e.g. fog) as well as the cloud. We incorporate an optimization strategy that considers the plurality of application requirements and the capabilities of resources that exist across both environments. By considering the capabilities of available resources and the requirements of a task, it is then possible to find the best matching resource that can be allocated for executing the task. As part of our proposed hybrid system model, we identify the best allocation plan for an encountered task. That is, there may exist a number of resources that can perform a particular task. However, identifying an optimal resource or selecting the best alternative is a major component of our proposed FogWeaver optimization strategy. We first identify key terminologies that we will use throughout the remainder of this thesis.

- Node: A node is defined as an entity that represents an IoT device or a containerized cloud resource running as a service residing in the hybrid environment.

- Resource: A computing entity that represents a combination of physical hardware components such as memory, processor, storage to which is responsible for executing software operations or tasks. Resources vary in terms of the degree they offer the hardware components which are considered resource attributes. Table 3.1 presents a list of possible attributes that a resource may possess.

- Task: A running process or executable piece of software that must be executed as part of a system or service request. An example of a task is an artificial intelligence (AI)

service, a Machine Learning (ML) service, an image processing request for identifying objects within images, computing the average of elements in a vector or an array, among many others.

Using these definitions we formulate a model for our hybrid IoT systems employing a layered structure as shown in Figure 3.1. These layers and the cost model are explored in this section.

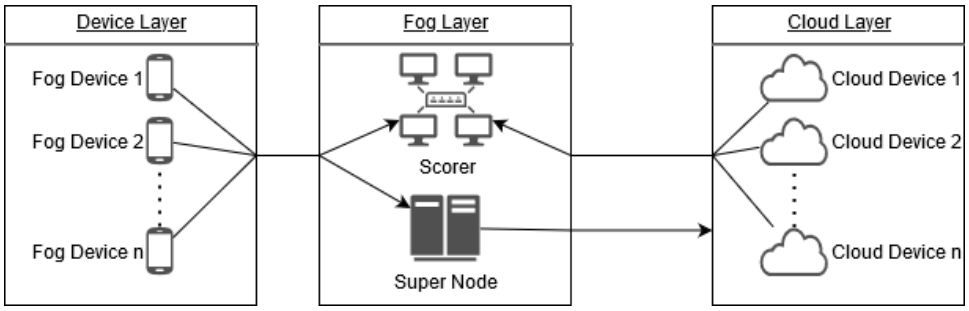

Figure 3.1: System Model Layer Diagram

3.1 Device Layer

The device layer consists of a number of fog devices (n). These fog devices can be in the form of computing devices such as laptops, cellphones, Raspberry Pi, Arduino, among many others. The device layer represents the basic layer in an IoT system for data collection where devices are often equipped with sensors that perform some measurements and collect data. Hence, we consider fog devices that exist at this layer as fog nodes represented as a node in a directed graph where each node may possess the attributes defined in Table 3.1. The nodes of the graph are scored and the model is navigated to check for a node that can be allocated to perform a given task. We discuss the resource allocation process further in the next chapter. Given that the device layer may contain multiple different fog devices that vary in the magnitude of physical hardware attributes (e.g. processor, memory, storage, among others), this layer is generally responsible for the initiation of the plurality of tasks.

Our optimization model identifies the capabilities possessed by the devices that exist in this layer to determine whether a task can run optimally or requires offloading to other layers in the fog hierarchy.

3.2 Fog Layer

The fog layer serves as the first point of contact between fog devices and the back-end of IoT systems. It is responsible for the communication between multiple fog nodes and is equipped with hardware characteristics that enable it to run or process tasks at the fog level. Examples of entities that can exist at this layer include fog gateways, micro data centers, or other types of devices that are powerful enough to execute and coordinate tasks. A fog layer can consist of a number of fog nodes where each node possesses physical attributes that make it act as an available resource. A number of fog nodes can form a cluster of nodes or what we refer to as a supernode (which is described below).

3.2.1 Scorer

The scorer is the component that reads in the user's preferences, takes in the data from the directed graph nodes, and computes a node score for each encountered node using a weighted-sum optimization technique. This process is described in Chapter 4.

3.2.2 Supernode

Supernode: a construct of taking an optimal node that can be discovered across the device or fog layers and combining it with a complimentary node to handle a task, allowing the system to avoid having to offload fully to the cloud layer. In some circumstances, it outperforms what is achieved in the cloud layer, while helping to keep the monetary cost low.

3.3 Cloud Layer

The cloud layer consists of a number of cloud devices (m) representing containerized services or virtual machine instances. As with the other layers, we assume that each cloud device as

a node in a directed graph where each node has the attributes in Table 3.1. The nodes of the graph are scored and the model is navigated to check for an available node to allocate the given task to. This process is described in Chapter 4.

Table 3.1: Features Characterizing Computational Node Capabilities

Attribute	**Description**
CPU Cores	The number of cores allocated to the node
CPU Capacity	The provisioned CPU available to the node in MHz
CPU Usage	The CPU used by the task in MHz
CPU Percentage	The percentage of the CPU used
Memory Capacity	Allocated memory on the node in KB
Memory Usage	Allocated memory used by the task in KB
Disk Read Throughput	The disk read throughput of the node in KB/s
Disk Write Throughput	The disk write throughput of the node in KB/s
Network Receive Throughput	The network receive throughput of the node in KB/s
Network Transmit Throughput	The network transmit throughput of the node in KB/s

3.4 Cost Model

To enhance the optimization process through our FogWeaver strategy, we designed a cost model that aims to reduce the costs associated with running a task. The costs associated with the various layers may vary. For example, at the device layer, the costs associated with running a task on a device or fog node is significantly smaller compared to the nodes residing on the cloud. To compare the trade-offs between fog and cloud layers, we assume that the cost associated with running a task locally on the fog or device layers is zero. For the costs associated with running tasks on cloud nodes, we used available calculators offered by existing cloud service platform providers such as the Microsoft Azure Price Calculator.

Using these calculators, we are able to create an estimate based on the resources consumed on a cloud layer including, for example, CPU cores, memory, network bandwidth, and other types of attributes that characterize a cloud node. For our cost model, we based the costs on the total number of CPU cores estimated by the price calculators.

To optimize the task allocation within our proposed hybrid IoT system, we propose FogWeaver as a multi-objective optimization strategy that is capable of adapting to the needs of users or application requirements. In addition, FogWeaver considers the plurality of node capabilities to determine a suitable layer for executing a task. FogWeaver is designed such that it will first consume all available resources at the fog layer, then offload tasks to the cloud in cases resources are not available locally (e.g. on fog or device layers). Through this approach, it is then possible to build a well-balanced approach that minimizes the execution costs associated with running tasks.

Chapter 4

OPTIMIZATION STRATEGY

Using the models discussed in Chapter 3, we designed and developed an optimization strategy called FogWeaver. FogWeaver employs a multi-objective particle swarm population-based optimization technique that is incorporated into the decision process as shown in Figure 4.1. FogWeaver takes into consideration the availability of existing resources and their capabilities in addition to considering the application requirements or user preferences. The implementation of FogWeaver is divided into three main components: (a) metric of comparison, (b) performance threshold, and (c) system navigation.

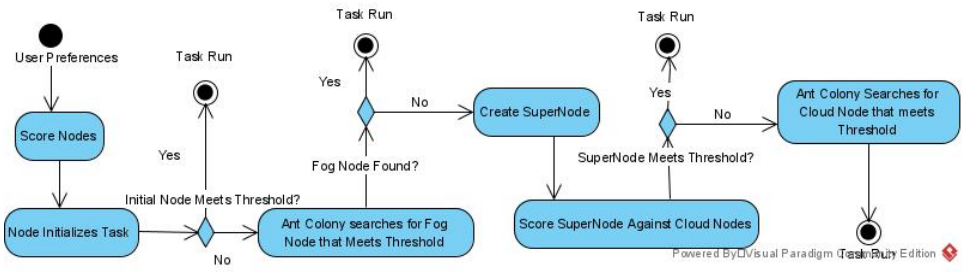

Figure 4.1: UML Decision Diagram for Task Allocation Process

4.1 Metric of Comparison

The metric of comparison for FogWeaver uses a multi-objective weighted-sum methodology that retrieves users' preferences or application requirements and scores the nodes that make up the system. The decision to use a multi-objective weighted sum is based on the need to adapt to a wide range of IoT application types. By employing a weighted sum, it is then possible to identify priorities on specific attributes or characteristics. The metric of

comparison is calculated using the following equations:

4.1.1 Fog Environment

Using the device layer model we used data from the GWA-T-12 fastStorage containing trace data on 1,250 virtual machines to fast storage area network (SAN) devices to fill in the resource attributes of each node [18]. This data was also used to guide the development of the following equations for use by the scorer of the fog layer.

$$F_{wsE}(X) = \sum_{i=1}^{k} w_{iE} \frac{f_{iE}(X) - f_{iE}^{min}}{f_{iE}^{max} - f_{iE}^{min}} \tag{4.1}$$

where $F_{wsE}(X)$ represents the metric of comparison for fog device X that exists on a fog environment, and k represents the number of objective functions for the fog environment $f_{iE}(X)$, f_{iE}^{min} is the minimum value for all nodes in the fog environment for the given objective, f_{iE}^{max} is the maximum value for all nodes in the fog environment for the given objective, and weights $w_{iE} \in [0, 1]$ where $\sum_{i=1}^{k} w_{iE} = 1$ [12]. For our purposes k is 7, and the objective functions are as follows:

$$f_{1E}(X) = \frac{C_{CPU}(X)}{C_{CPU}^{max}} * (1 - P_{CPU}(X)) \tag{4.2}$$

where $C_{CPU}(X)$ is the CPU capacity for fog device X, C_{CPU}^{Max} is the maximum CPU capacity among all of the fog devices and $P_{CPU}(X)$ is the percentage of the CPU used by the fog device.

$$f_{2E}(X) = \frac{C_{mem}(X)}{C_{mem}^{max}} * (1 - P_{mem}(X)) \tag{4.3}$$

where $C_{mem}(X)$ is the memory capacity for fog device X, C_{mem}^{Max} is the maximum memory capacity among all of the fog devices and $P_{mem}(X)$ is the percentage of the memory used by the fog device:

$$P_{mem}(X) = \frac{C_{mem}(X)}{U_{mem}(X)} \tag{4.4}$$

where $U_{mem}(X)$ is the memory used for the fog device.

$$f_{3E}(X) = \frac{T_{read}(X)}{T_{read}^{max}} + \frac{T_{write}(X)}{T_{write}^{max}} \tag{4.5}$$

where $T_{read}(X)$ is the disk read throughput for fog device X, T_{read}^{Max} is the maximum disk read throughput among all of the fog devices, $T_{write}(X)$ is the disk write throughput for fog device X, T_{write}^{Max} is the maximum disk write throughput among all of the fog devices.

$$f_{4E}(X) = \frac{T_{rec}(X)}{T_{rec}^{max}} + \frac{T_{trans}(X)}{T_{trans}^{max}} \tag{4.6}$$

where $T_{rec}(X)$ is the network receive throughput for fog device X, T_{rec}^{Max} is the maximum network receive throughput among all of the fog devices, $T_{trans}(X)$ is the network transmit throughput for fog device X, T_{trans}^{Max} is the maximum network transmit throughput among all of the fog devices.

$$f_{5E}(X) = 1 - \frac{t(X) - t_{min}}{t_{min}} \tag{4.7}$$

where $t_(X)$ is the estimate execution time for fog device X, C_{min} is the minimum estimate execution time among all of the fog devices.

$$f_{6E}(X) = r(X) \tag{4.8}$$

where $r(X)$ is the reliability for a given fog device, it is a value between 0 and 1 that corresponds to how often the fog device completes a task successfully.

$$f_{7E}(X) = (1 - w_{tc}); \quad 0 \leq w_{tc} \leq 1 \tag{4.9}$$

where w_{tc} is a trade-off weight between the minimization of the execution time and execution cost of a task. For our implementation, we worked with the assumption that the cost score for all fog devices is treated as 1, as we assume the cost to be 0 as given by the cost model.

4.1.2 Cloud Environment

As with the fog environment, we used data from GWA-T 12 trace data to identify the values of the attributes for the cloud nodes and develop the metric of comparison equation and objective functions for the cloud environment. We used the rnd trace data from 2013-08 instead of the fastStorage trace data [18] for the cloud environment simulation. The following equation represents the metric of comparison.

$$F_{wsC}(X) = \sum_{i=1}^{k} w_{iC} \frac{f_{iC}(X) - f_{iC}^{min}}{f_{iC}^{max} - f_{iC}^{min}} \tag{4.10}$$

where $F_{wsC}(X)$ is the metric of comparison for cloud node X the cloud environment, and k is the number of objective functions for the cloud environment $f_{iC}(X)$, f_{iC}^{min} is the minimum value for all nodes in the cloud environment for the given objective, f_{iC}^{max} is the maximum value for all nodes in the cloud environment for the given objective, and weights $w_{iC} \in [0, 1]$ where $\sum_{i=1}^{k} w_{iC} = 1$ [12]. For our purposes k is 7, and the objective functions are as follows:

$$f_{1C}(X) = \frac{C_{CPU}(X)}{C_{CPU}^{max}} * (1 - P_{CPU}(X)) \tag{4.11}$$

where $C_{CPU}(X)$ is the CPU capacity for cloud node X, C_{CPU}^{Max} is the maximum CPU capacity among all of the cloud nodes and $P_{CPU}(X)$ is the percentage of the CPU used by the cloud node.

$$f_{2C}(X) = \frac{C_{mem}(X)}{C_{mem}^{max}} * (1 - P_{mem}(X)) \tag{4.12}$$

where $C_{mem}(X)$ is the memory capacity for cloud node X, C_{mem}^{Max} is the maximum memory capacity among all of the cloud nodes and $P_{mem}(X)$ is the percentage of the memory used by the cloud node.

$$P_{mem}(X) = \frac{C_{mem}(X)}{U_{mem}(X)} \tag{4.13}$$

where $U_{mem}(X)$ is the memory used for the cloud node.

$$f_{3C}(X) = \frac{T_{read}(X)}{T_{read}^{max}} + \frac{T_{write}(X)}{T_{write}^{max}} \tag{4.14}$$

where $T_{read}(X)$ is the disk read throughput for cloud node X, T_{read}^{Max} is the maximum disk read throughput among all of the cloud nodes, $T_{write}(X)$ is the disk write throughput for cloud node X, T_{write}^{Max} is the maximum disk write throughput among all of the cloud nodes.

$$f_{4C}(X) = \frac{T_{rec}(X)}{T_{rec}^{max}} + \frac{T_{trans}(X)}{T_{trans}^{max}} \tag{4.15}$$

where $T_{rec}(X)$ is the network receive throughput for cloud node X, T_{rec}^{Max} is the maximum network receive throughput among all of the cloud nodes, $T_{trans}(X)$ is the network transmit throughput for cloud node X, T_{trans}^{Max} is the maximum network transmit throughput among all of the cloud nodes.

$$f_{5C}(X) = 1 - \frac{t(X) - t_{min}}{t_{min}} \tag{4.16}$$

where $t_(X)$ is the estimated latency/execution time for cloud node X, C_{min} is the minimum estimate latency/execution time among all of the cloud nodes.

$$f_{6C}(X) = r(X) \tag{4.17}$$

where $r(X)$ is the reliability for a given cloud node, it is a value between 0 and 1 that corresponds to how often the cloud node completes a task successfully.

$$f_{7C}(X) = 0 \tag{4.18}$$

This function is set as 0 as the cost for running on cloud nodes. To this extent, the cost score for all cloud nodes is modeled as 0. This equation is included in the objective functions even though it is assumed to be 0 for all of the cloud nodes, as we need to factor in the loss of the cost weight preference, and to allow for future refinement of the cost minimization.

The estimation is based on the cost per core per hour of the D1 v2 virtual machine which comes out to be $0.126 per core per hour, a value that leads to cost estimates approaching $3.56 per hour. These values come from the Microsoft Azure pricing calculator.

4.2 Performance Thresholds

This section shows the methods used for determining the performance threshold or node score that a node needs to have for the system navigation to stop and return the given node as a possible solution.

4.2.1 Difference-Based Threshold

The following equation is used to compute the difference-based threshold.

$$\lambda = os - ((os - ms) * (1 - w_{th}); \quad 0 \leq w_{th} \leq 1, ms \neq 0 \tag{4.19}$$

where os is the optimal score amongst all cloud and fog nodes, ms is the minimum non-trivial score among all cloud and fog nodes, and w_{th} is the weight threshold set by the user that represents the preference between the allocation time and the finding the optimal score.

4.2.2 Percentage-Based Threshold

The following equation is used to compute the percentage-based threshold.

$$\lambda = os * p; \quad 0 \leq p \leq 1 \tag{4.20}$$

where os is the optimal score among all cloud and fog devices, and p is a percentage value set by the user.

4.2.3 Two-Level Threshold

The two-level threshold employs the difference-based threshold as the initial check for the performance of a node and then applies the percentage-based threshold. This is designed to catch critical systems and ensure that they operate at the correct level.

4.3 System Navigation

The system navigation method that is employed by FogWeaver is based on the particle swarm optimization methodology. This is based on the most common methods found in the literature for task placement in IoT systems as discussed in Chapter 2. A working implementation of FogWeaver was developed using a modified ant colony, modified to look at the score of an individual node instead of the path is taken, before transitioning into a particle swarm methodology, after tests showed that the particle swarm achieved comparable accuracy in a third of the allocation time. Results from this version of FogWeaver are further described in detail in a peer-reviewed paper published in the proceedings of the 3rd IEEE International Conference on Knowledge Innovation and Invention 2020 found in Appendix D.

The particle swarm methodology begins with creating n number of particles that move m times continuously through the solution space. For the purpose of our application of FogWeaver, the solution space is a directed-graph representing either the fog or cloud environments. Each of the n particles is placed randomly throughout the nodes of the graph and move at most m times. In some cases, a particle may move less than m times if a particle achieves a score that meets the desired performance threshold. The n and m variables are set by finding the factorization of the total number of nodes in the graph for which q is minimum for:

$$q = |f_1 - f_2| \tag{4.21}$$

where f_1 is one number in the factor pair, and f_2 is the other. For instance, if we have a graph with 250 nodes, $f_1 = 10$ and $f_2 = 25$. After finding the factorization for the given number of nodes, the larger value is assigned to n, and the smaller is assigned to m. This is based on early tests into the optimal navigation method, based on the allocation time and allocation accuracy.

4.3.1 Particle Velocity

The following equation computes the velocity of the particle:

$$v_{i+1}^k = \omega v_i^k + c_1 r_1 \frac{pb^k - x_i^k}{\Delta t} + c_2 r_2 \frac{gb_i - x_i^k}{\Delta t} \tag{4.22}$$

where v_{i+1}^k is the velocity the particle will have for the next iteration, ω is the inertia constant and is a constant value between 0.4 and 1.4 each iteration it is multiplied by a damping factor to slow particles as they get closer to the solution, v_i^k is the velocity of particle k at iteration i, c_1 is the self-confidence constant and influences how much the particle relies on its own motion and is a constant between 1.5 and 2, c_2 is the swarm confidence constant which influence how much the particle relies on the swarm for its own motion and is a constant between 2 and 2.5, r_1 and r_2 are random numbers between 0 and 1, pb^k is the best score for particle k, gb_i is the best score among all particles at iteration i, Δt is the amount of time between each velocity change, and x_i^k is the position of particle k at iteration i.

$$x_{i+1}^k = x_i^k + v_{i+1}^k \Delta t \tag{4.23}$$

where x_{i+1}^k is the position of particle k the next iteration.

4.4 Implementation Details of Particle Swarm Optimization

We used MATLAB for the implementation of the major components that compose Fog-Weaver including the system's navigation, optimization strategy and constructs needed for the system model. We incorporated the Find Closest Factorization library developed by Luke Gane [10] for finding the values used for the particle population size and maximum iterations. The complete source code implementation can be found in Appendix C.

Chapter 5
EVALUATION AND ASSESSMENT

To reduce the effect of the random nature of the navigation caused by the particle swarm, the data presented in this section consists of an average of 100 groups of 1000 runs. To test the algorithm under a wide variety of different situations, we applied various w_{tc} and w_{th} under the three threshold methods shown in the previous chapter. We also tested with systems of varying numbers of fog nodes between 10 and 1000.

5.1 Difference-Based Threshold

The following test cases represent scenarios for applying the difference-based threshold as shown in Equation 4.19. The data is divided into three sections: the first is with $w_{tc} = 1$, then $w_{tc} = 0.5$ and finally $w_{tc} = 0$ for equation 4.9.

5.1.1 Execution Time-Driven: $w_{tc} = 1$

In this test case, we would like to determine an optimal solution that primarily focuses on the execution time. Hence, this test case is execution time-driven. We set the trade-off weight value to 1. That is, the results are determined when the optimization strategy is instructed to entirely focus on the execution time regardless of the cost of executing a task. As a result, the outcome of this test case may yield resources that have varying magnitudes of costs. As we increase the performance threshold, the time for identifying an optimal solution in the solution space will likely increase. Hence, we refer to this time as the allocation time or the time it takes for the system to navigate through to find a possible acceptable solution. Table 5.1 presents the results from executing this test case with varying degrees of the weight threshold (w_{th}) that is set by the user representing the preference between the allocation

time and finding an optimal score.

Table 5.1: $w_{tc} = 1$ Accuracy Results Execution Time-Driven Test Case: $w_{tc} = 1$

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	98.2858%	98.3150%	89.8958%	87.2449%	84.27310%
25	97.9543%	97.9920%	86.4282%	74.4375%	65.62250%
50	97.9543%	97.9374%	79.1906%	70.4880%	58.13920%
75	97.9180%	97.9316%	78.3866%	70.0696%	54.25990%
100	97.8536%	97.7936%	76.8391%	66.1291%	52.08240%
250	94.5771%	92.8136%	80.6034%	69.2650%	39.71940%
500	94.5078%	92.7967%	78.2691%	61.7899%	35.80850%
750	94.5310%	91.9236%	76.2498%	61.6769%	34.06630%
1000	94.5367%	92.0832%	75.7009%	63.5431%	33.16380%

As can be seen from the results in Table 5.1, as we increase the weight threshold, the accuracy of FogWeaver finding an optimal solution increases. For example, when $w_{th} = 0.25$, the accuracy for 10 nodes decreases to 87.15%. In addition, the accuracy degrades significantly as there exist more nodes as in the case of 1000 nodes representing larger systems. That is, increasing the number of nodes is likely to cause the optimization algorithm to require more allocation time for finding an optimal solution in the search space. However, reducing the weight threshold, significantly impacts the accuracy reaching 61.67% in some cases. Results from Table 5.1 also suggest that as we increase the search space representing more nodes, the overall scores for each node begins to drop since the range of the values within the data becomes much larger.

Table 5.2 presents the cost results for the execution time-driven test case. As can be seen from Table 5.2, as the performance threshold is lowered, the cost significantly drops. This is because as the performance threshold decreases, the system utilizes the device and fog layer

Table 5.2: $w_{tc} = 1$ Cost Results Execution Time-Driven Test Case: $w_{tc} = 1$

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	$ 1.0382	$ 1.0361	$ 0.33881	$ 0.2960	$ 0
25	$ 1.2738	$ 1.2846	$ 0.17432	$ 0.0440	$ 0
50	$ 1.2870	$ 1.2947	$ 0.03074	$ 0	$ 0
75	$ 1.3275	$ 1.3264	$ 0.01573	$ 0	$ 0
100	$ 1.3640	$ 1.3555	$ 0.00748	$ 0	$ 0
250	$ 3.5549	$ 0.0062	$ 0.00153	$ 0	$ 0
500	$ 3.5444	$ 0.0636	$ 0.00171	$ 0	$ 0
750	$ 3.5521	$ 0.0766	$ 0.00022	$ 0	$ 0
1000	$ 3.5542	$ 0.3212	$ 0.00025	$ 0	$ 0

resources more extensively. Hence, the FogWeaver strategy would not be able to identify an optimal solution that can meet the requirements or user preferences. Furthermore, results from Table 5.2 demonstrate that in this test case the strategy functions more as a cloud environment rather than a true hybrid one. This is evident by the costs accrued for a task for optimal solutions which indicates that the majority of tasks are offloaded to the cloud.

5.1.2 Execution Time and Cost Balanced: $w_{tc} = 0.5$

In this test case, we would like to determine an optimal solution that strikes a balance between the execution time and execution cost. Hence, this test case is equally important to both execution time and cost. We set the trade-off weight value to 0.5. That is, the results are determined when the optimization strategy is instructed to balance minimizing both the execution time and cost when considering the execution of a task. As a result, we can formulate supernodes for large systems through the strategy to fully utilize the resources available at the device and fog layers. Therefore, in this hybrid model, the cost for running

a task would drop whereas the performance of the task is expected to increase allowing supernodes to be utilized as resources for completing tasks.

Table 5.3: $w_{tc} = 0.5$ Accuracy Results Execution Time and Cost Balance: $w_{tc} = 0.5$

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	95.1976%	90.1711%	86.0004%	86.0054%	17.9306%
25	94.1515%	87.5009%	69.3955%	69.4656%	30.5967%
50	94.0972%	84.0379%	62.8597%	62.7943%	42.7036%
75	93.8918%	83.3503%	59.4350%	59.3938%	44.1180%
100	93.7365%	81.7976%	57.4999%	57.4615%	44.0759%
250	100.3943%	91.9418%	68.3681%	49.6310%	35.6205%
500	99.9877%	88.4249%	63.2410%	45.7213%	37.6205%
750	99.9682%	84.5562%	64.2163%	44.4822%	38.1396%
1000	99.2824%	84.0976%	64.8679%	43.6683%	37.9624%

Table 5.3 shows interesting results in the accuracy values for $w_th = 1$ for systems with having at least 250 fog nodes. In these systems, it is apparent that the accuracy values are slightly higher or lower than 100%. These results demonstrate that the system is fully utilizing the resources available in both the device and fog layers through the creation of supernodes in an attempt to meet the performance threshold. In this case, if supernodes are not available or meeting the optimal score required, the task would then be offloaded to the cloud for further processing.

The relatively low accuracy seen in Table 5.3 is due to the fact that we set $w_{th} = 0$ which instructors the strategy to identify any available resource to run the task. This result is most apparent for the 10 fog node system where the task is run on a device layer resource when the cost is not considered is not marked as available.

Table 5.4 shows that the cost results reflecting the accuracy results increase as the system

Table 5.4: $w_{tc} = 0.5$ Cost Results Execution Time and Cost Balance: $w_{tc} = 0.5$

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	$ 1.0435	$ 0.3921	$ 0	$ 0	$ 0
25	$ 1.2824	$ 0.2023	$ 0	$ 0	$ 0
50	$ 1.2904	$ 0.1026	$ 0	$ 0	$ 0
75	$ 1.3289	$ 0.0527	$ 0	$ 0	$ 0
100	$ 1.3698	$ 0.2473	$ 0	$ 0	$ 0
250	$ 0.0104	$ 0.0049	$ 0	$ 0	$ 0
500	$ 0.0691	$ 0.0019	$ 0	$ 0	$ 0
750	$ 0.0898	$ 0.0001	$ 0	$ 0	$ 0
1000	$ 0.3353	$ 0.0001	$ 0	$ 0	$ 0

utilizes the device and fog layer resources to their full capacity. As a result, tasks are not likely to be offloaded to the cloud. That is, as the performance threshold decreases, the determination to run entirely on the device or fog layers is achieved at a fast rate.

5.1.3 Execution Cost-Driven: $w_{tc} = 0$

In this test case, we would like to determine an optimal solution that primarily focuses on the execution cost. Hence, this test case is execution cost-driven. We set the trade-off weight value to 0. That is, the results are determined when the optimization strategy is instructed to entirely focus on the execution cost regardless of the time for executing a task. As a result, the outcome of this test case may yield resources that would otherwise be ignored as they take too long to execute a task. As in the balanced test case, we identify a heavy utilization of device and fog layer resources including the implementation of supernodes to ensure that costs are minimized when executing a task. However, this comes at the trade-off of lower accuracy as the performance threshold is lowered and the execution time is ignored allowing

resources that would not be used due to the amount of time they take to attempt executing a task.

Table 5.5: $w_{tc} = 0$ Accuracy Results Execution Cost-Driven: $w_{tc} = 0$

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	92.7054%	90.0172%	87.3091%	87.3819%	18.1145%
25	91.0598%	85.0283%	72.4548%	72.5060%	32.4031%
50	91.0063%	83.2514%	66.6337%	66.5658%	45.9235%
75	90.6752%	82.6922%	63.5667%	63.5701%	47.7389%
100	90.5189%	81.4584%	61.8722%	61.7890%	47.7456%
250	100.3138%	84.3111%	54.7206%	54.6908%	40.2504%
500	99.7862%	83.1309%	51.5066%	51.4745%	42.9511%
750	99.7345%	82.2166%	50.4408%	50.5000%	43.8464%
1000	98.3235%	81.6470%	49.7516%	49.7356%	43.7136%

Table 5.5 shows a pattern of resource utilization that follows a similar one seen in the balanced test case. The key difference is that because the cost is factored with high weight or priority into the metric of comparison, results from 5.5 demonstrate that the resources utilized that previously would not meet the performance threshold. As a result the $w_{th} = 0$ accuracy results are higher as the score for these resources due to the weight assigned by the cost trade-off. In addition, results from Table 5.5 also demonstrate a drop to slightly below 60% accuracy in terms of task performance as the performance threshold is determined by less optimal resources since their metric of comparison scores are improved further by the cost factor.

Table 5.6 shows that for the highest level of the performance threshold. Results from Table 5.6 demonstrate that for the optimal solutions determined by the strategy, the cost levels are mirrored between the cost-driven and balanced results. This is an indicator that

Table 5.6: $w_{tc} = 0$ Cost Results Execution Cost Driven: $w_{tc} = 0$

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	$ 1.0349	$ 0.3951	$ 0	$ 0	$ 0
25	$ 1.2812	$ 0.1006	$ 0	$ 0	$ 0
50	$ 1.2904	$ 0.0362	$ 0	$ 0	$ 0
75	$ 1.3359	$ 0.0188	$ 0	$ 0	$ 0
100	$ 1.3650	$ 0.0105	$ 0	$ 0	$ 0
250	$ 0.0109	$ 0	$ 0	$ 0	$ 0
500	$ 0.00695	$ 0	$ 0	$ 0	$ 0
750	$ 0.0839	$ 0	$ 0	$ 0	$ 0
1000	$ 0.3389	$ 0	$ 0	$ 0	$ 0

when attempting to find a truly optimal resource, it is possible to find a balance between the execution time and execution cost while achieving a minimal cost. In addition, results from Table 5.6 show that as the performance threshold decreases, the focus on minimizing the cost yields lower values for the device layer resources since they are scored highly enough to be used. It is worth noting that the cost difference for these results is very minimal and is most likely the difference in offloading to the cloud once or twice in the thousands of trials for the test cases.

5.2 Percentage-Based Threshold

After testing with the performance threshold calculated using Equation 4.19, we conducted a number of experiments using the performance threshold from Equation 4.20. This was achieved to determine the outcome of applying a simpler performance threshold determination method and compare the results with the difference-based methodology. Appendix A provides detailed results of the tests that we conducted. For all experiments conducted, they

applied the same execution time and cost trade-off values as in the difference-based methodology tests. Because the performance threshold is based entirely on the optimal score and does not vary with the minimum score, lower accuracy is expected as the performance threshold level is reduced.

Figure 5.1 shows a scatter plot of the accuracy results for all of the test cases of the execution cost and time trade-off. Appendix A provides more detailed tables of the data generated as a result of this test case. As can be seen in Figure 5.1, when the cost is factored into the metric of comparison and the performance threshold is at its highest value, the extreme utilization of supernodes allows resources to achieve maximum levels of performance not otherwise possible. Figure 5.1 also shows that it is possible to achieve the best results for large systems by implementing a balance between the execution cost and time for a given task.

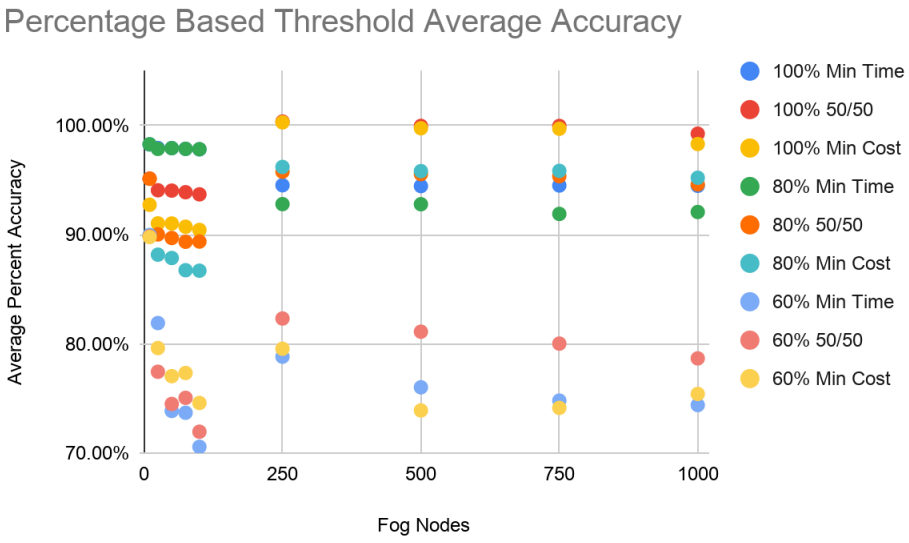

Figure 5.1: Percentage Threshold Accuracy Results

Figure 5.2, shows the percentage-based threshold average cost. As can be seen in Figure 5.2, there exists a drop in the cost resulting from the full utilization of the fog and device

layers. This also confirms that the strategy is able to minimize the cost. Results from Figure 5.2 also demonstrate that for larger systems that are able to properly utilize the supernodes and hybrid resource allocation, the balanced and execution cost driven results mirror each other.

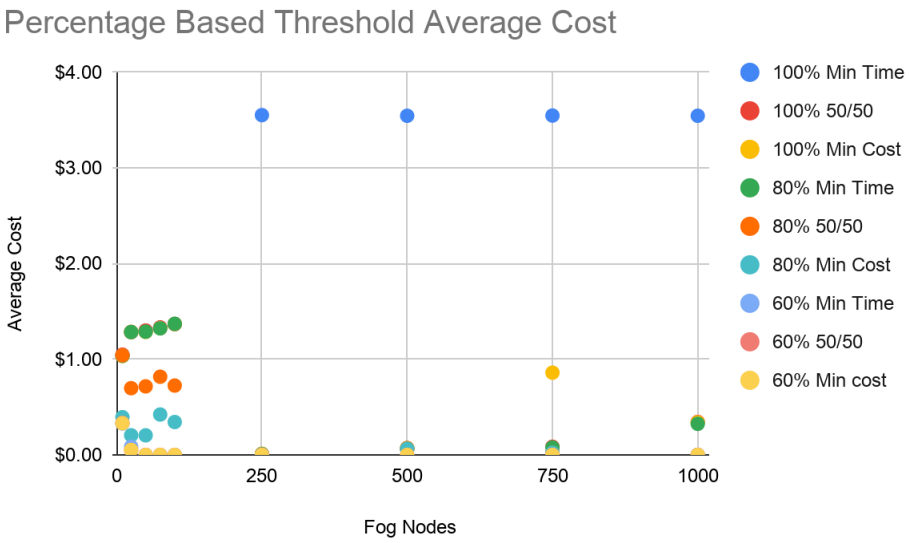

Figure 5.2: Percentage Threshold Cost Results

While this method has benefits over the difference-based threshold methodology, it suffers from limitations in the adaptability of having more control over the users' preferences between allocation time and optimizing the score determined by the difference-based threshold. As such, it is recommended to use the difference-based threshold methodology as the main cutoff level and to save the percentage-based threshold method as a catch-for-all for the two threshold methodology involving mission-critical tasks. This will be explored further in the next section.

5.3 Two-Level Threshold Results

To test the two-level threshold result, we set the percentage-based catch threshold as 60%, as this is the minimum level of performance that would be allowed for system operation, in particular for mission-critical tasks. The two-level threshold system begins by applying the difference-based threshold for a task, and then as the back-up catch checks against the percentage-based threshold to ensure that the system runs at the necessary level. We further experimented on the test cases for which the difference-based threshold accuracy is below 60% since other cases would not show any impact with the use of the two-level threshold. The data for these tests are found in Appendix A.

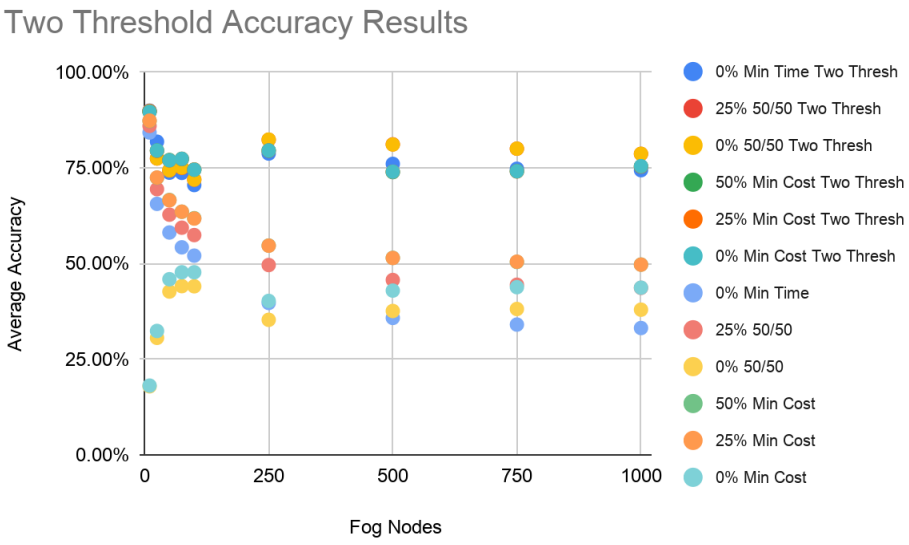

Figure 5.3: Two-Level Threshold Accuracy Results

In order to demonstrate the improvement of the resource performance in the system Figure 5.3 shows the two-level accuracy results for the difference-based threshold without the two-level threshold for those cases that are below 60% and the accuracy results found when the two-level threshold is implemented. Results from Figure 5.3 show significant improvement in the resource performance as the percentage-based threshold of the two-level threshold

places the performance threshold above the 60%. More detailed data from these tests can be found in Appendix A.

5.4 Summary of Optimization Strategy

By testing FogWeaver on a large range of w_{tc} and w_{th} as well as varying numbers of fog nodes, the performance of the optimization strategy for most performance thresholds can achieve scores of at least 60%. Tables 5.3 to 5.6 show the ability of the optimization strategy to minimize the cost while still maintaining task allocation accuracy by employing supernodes which allows the system to achieve a performance that it could not without using supernodes. Therefore, this allows for the placement of tasks in the fog environment that would otherwise be offloaded to the cloud.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Contributions and Findings of FogWeaver

By using a multi-objective weighted-sum scoring methodology, FogWeaver is able to create an adaptable metric of comparison that allows for efficient resource allocation in a wide variety of IoT systems. In addition, this enables IoT systems to maintain high levels of accuracy when the strategy is designed to focus on finding an optimal score. FogWeaver is also able to use multiple methods for determining the performance threshold that enables users to adapt the threshold based on their needs while working with multiple navigation algorithms. We demonstrated this feature through the application of the optimization strategy using both the particle swarm and modified ant colony algorithms.

Through this research, we identified the key factors for comparing the performance of a task across a hybrid IoT environment and developed a metric of comparison for scoring the computational nodes across the environment for use in optimization of resource allocation. By employing a multi-objective weighted-sum methodology for scoring, we developed an adaptable method that is able to adjust to the needs of the user. In addition, by employing the performance cutoff thresholds, it is then possible for users to further fine-tune the strategy to their needs while being able to maintain an accuracy of 60% or more when the system is not pushed to extreme levels.

Through the use of the execution time-Cost trade-off constant, w_{tc}, FogWeaver allows the user to further fine-tune how heavily the cost of running a task is employed in scoring the computation nodes of the system. Throughout the thesis, we demonstrated how FogWeaver is able to achieve allocation accuracy beyond what could be achieved otherwise by using supernodes. The strategy has the greatest benefit when searching for optimal scores while

evenly balancing the performance with the monetary cost. Through the use of the two-level threshold catch method, we demonstrate that FogWeaver can identify and greatly improve the performance of mission-critical tasks particularly in cases when the first performance threshold passes through a computational node that does not meet the needs of the system.

The contributions of FogWeaver and our research work are as follows:

- We created an adaptable metric of comparison across hybrid environments.
- We designed and developed an optimization strategy that is able to adapt to the needs of users while maintaining an adequate level of accuracy of at least 60%.
- We extended the optimization strategy to consider the use of supernodes and the ability to balance between the performance and cost minimization.
- We further enhanced the optimization strategy with the ability to catch and maintain the performance levels of mission-critical tasks through the use of a two-level threshold methodology.

Based on the above research contributions, FogWeaver provides a very promising foundation for further exploration of hybrid IoT resource allocation for tasks across fog environments that can be smoothly adapted to meet the needs of an IoT system and various methods of modeling.

6.2 Future Work

While FogWeaver has been developed to optimize resources across hybrid fog-environments, there is room for improvements. It would be desirable to extend this work to be deployed on a real-world IoT system involving actual fog-based nodes and IoT devices. In addition, more advanced cost model analysis for cloud-based computations can help further the accuracy of the hybrid optimization strategy. Another possible improvement is to run the optimization strategy to navigate through the solution search space for both cloud and fog environments

in parallel. In addition, there is room for exploration of other optimization considerations, such as the energy usage for running a task (more on this topic can be found in Appendix B).

BIBLIOGRAPHY

[1] H. Alayed, F. Dahan, T. Alfakih, H. Mathkour, and M. Arafah. Enhancement of ant colony optimization for qos-aware web service selection. *IEEE Acess*, 7:97041–97051, 2019.

[2] E. S. Alkayal, N. R. Jennings, and M. F. Abulkhair. Efficient task scheduling multi-objective particle swarm optimization in cloud computing. *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*, pages 17–24, 2016.

[3] R. Arora. *Optimization: Algorithms and Applications*. CRC Press, 2015.

[4] M. Barcelo, A. Correa, J. Llorca, A. M. Tulino, J. L. Vicario, and A. Morell. Iot-cloud service optimization in next generation smart environments. *IEEE Journal on Selected Areas in Communications*, 34(12):4077–4090, 2016.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. *MCC Workshop on Mobile Cloud Computing*, pages 13–16, 2012.

[6] Y. Choi and Y. Lim. Optimization approach for resource allocation on cloud computing for iot. *International Journal of Distributed Sensor Networks*, 2016:1–6, 2016.

[7] H. Ding, V. Guo, S. Qin, Z. Zhang, R. Dautov, and Z. Chang. Alibaba cluster trace program. https://github.com/alibaba/clusterdata. Accessed: 2020-06-25.

[8] J. J. Durillo, J. García-Nieto, A. J. Nebro, C. A. C. Coello, F. Luna, and E. Alba. Multi-objective particle swarm optimizers: An experimental comparison. In M. Ehrgott, C. M. Fonseca, X. Gandibleux, JK. Hao, and M. Sevaux, editors, *Evolutionary Multi-Criterion Optimization*, pages 495–509, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[9] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(6):911–924, 2010.

[10] L. Gane. Find closest factorization. https://www.mathworks.com/matlabcentral/fileexchange/57255-find-closest-factorization, MATLAB Central File Exchange. Accessed: 2020-06-25.

[11] L. Guo, G. Shao, and S. Zhao. Multi-objective task assignment in cloud computing by particle swarm optimization. *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4, 2012.

[12] M. Hu and B. Veeravalli. Dynamic scheduling of hybrid real-time tasks on clusters. *IEEE Transactions on Computers*, 63(12):2988–2997, 2014.

[13] N. Kherraf, H. A. Alameddine, C. M. Sharafeddine, C. M. Assi, and A. Ghrayeb. Optimized provisioning of edge computing resources with heterogeneous workload in iot networks. *IEEE Transactions on Network and Service Management*, 16(2):459–474, 2019.

[14] Q. Liu, W. Cai, J. Shen, Z. Fu, X. Liu, and Linge N. A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environments. *Security Comm. Networks*, 9:4002–4012, 2016.

[15] J. Olmsted and E. Al-Masri. Fogweaver: Task allocation optimization strategy across hybrid fog environments. *2020 3rd IEEE International Conference on Knowledge Innovation and Invention*, 2020.

[16] H. N. Pham-Nguyen and Q. Tan-Mih. Dynamic resource provisioning on fog landscapes. *Security and Communication Networks*, 2019:1–5, 2019.

[17] J. Santos, T. Wauters, B. Volckaert, and F. De Turck. Resource provisioning for iot application services in smart cities. *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9, 2017.

[18] S. Shen, V. V. Beek, and A. Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015.

[19] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 320–327, 2011.

[20] A. Verma and S. Kaushal. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. *Parallel Computing*, 62:1–19, 2017.

[21] L. Zuo, S. Shu, S. Dong, C. Zhu, and T. Hara. A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. *IEEE Access*, 3:2687–2699, 2015.

Appendix A

ADDITIONAL DATA

Table A.1: Metric of Comparison Weights

Factor	**Value**
Power (w_1)	0.2
Memory (w_2)	0.2
Disk (w_3)	0.2
Network	0.1
Cost	0.15
Time	0.15
Reliability	0.05

Table A.2: Particle Swarm Constants

Constant	**Value**
ω	1
ω Damping	0.99
c_1	2
c_2	2
Δt	1

Table A.3: Percentage Threshold $w_{tc} = 1$ Accuracy Results

Nodes	$w_{th} = 1$	$w_{th} = 0.8$	$w_{th} = 0.6$
10	98.2864%	98.3029%	89.9689%
25	97.9610%	97.8770%	81.9172%
50	97.9523%	97.9468%	73.8627%
75	97.8849%	97.8811%	73.6967%
100	97.8399%	97.8510%	70.5758%
250	94.5465%	92.8174%	78.8346%
500	94.4719%	92.8067%	76.0287%
750	94.5235%	91.9208%	74.8096%
1000	94.4953%	92.0944%	74.4012%

Table A.4: Percentage Threshold $w_{tc} = 0.5$ Accuracy Results

Nodes	$w_{th} = 1$	$w_{th} = 0.8$	$w_{th} = 0.6$
10	95.1575%	95.1385%	89.8587%
25	94.0857%	90.0404%	77.4545%
50	94.0559%	89.7132%	74.5031%
75	93.9091%	89.3683%	71.9584%
100	93.7123%	89.3869%	71.9584%
250	100.3840%	95.7852%	82.3331%
500	99.9834%	95.5695%	81.1263%
750	99.9809%	95.3781%	80.0339%
1000	99.2710%	94.6247%	78.6803%

Table A.5: Percentage Threshold $w_{tc} = 0$ Accuracy Results

Nodes	$w_{th} = 1$	$w_{th} = 0.8$	$w_{th} = 0.6$
10	92.7558%	89.9830%	89.7998%
25	91.0519%	88.1843%	79.6263%
50	91.0425%	87.8794%	77.0499%
75	90.7438%	86.7600%	77.3400%
100	90.4497%	86.7163%	74.5937%
250	100.3102%	96.2222%	79.5608%
500	99.7732%	95.8403%	73.9090%
750	99.7279%	95.8640%	74.1429%
1000	98.3286%	95.2321%	75.4217%

Table A.6: Percentage Threshold $w_{tc} = 1$ cost Results

Nodes	$w_{th} = 1$	$w_{th} = 0.8$	$w_{th} = 0.6$
10	\$1.0362	\$1.0380	\$0.3284
25	\$1.2824	\$1.2851	\$0.0863
50	\$1.2989	\$1.2882	\$0
75	\$1.3307	\$1.3231	\$0
100	\$1.3700	\$1.3699	\$0.0002
250	\$3.5529	\$0.0085	\$0
500	\$3.5459	\$0.0638	\$0.0007
750	\$3.5482	\$0.0776	\$0
1000	\$3.5461	\$0.3252	\$0

Table A.7: Percentage Threshold $w_{tc} = 0.5$ Cost Results

Nodes	$w_{th} = 1$	$w_{th} = 0.8$	$w_{th} = 0.6$
10	\$1.0480	\$1.0419	\$0.3343
25	\$1.2855	\$0.6977	\$0.0495
50	\$1.3000	\$0.7167	\$0
75	\$1.3331	\$0.8176	\$0
100	\$1.3671	\$0.7521	\$0.0001
250	\$0.0100	\$0.0055	\$0
500	\$0.0725	\$0.0597	\$0
750	\$0.0851	\$0.0218	\$0
1000	\$0.3449	\$0.0004	\$0

Table A.8: Percentage Threshold $w_{tc} = 0$ Cost Results

Nodes	$w_{th} = 1$	$w_{th} = 0.8$	$w_{th} = 0.6$
10	\$1.0340	\$0.3942	\$0.3314
25	\$1.2848	\$0.2051	\$0.0518
50	\$1.2863	\$0.2046	\$0
75	\$1.3252	\$0.4224	\$0
100	\$3693	\$0.3437	\$0
250	\$0.0117	\$0.0053	\$0
500	\$0.0694	\$0.0627	\$0
750	\$0.8599	\$0.0229	\$0
1000	\$0.3400	\$0.0003	\$0

Table A.9: $w_{tc} = 1$ Two-Level Threshold Accuracy Results

Nodes	$w_{th} = 0$
10	89.9952%
25	81.8771%
50	73.654%
75	73.7201%
100	70.5426%
250	78.7830%
500	76.0777%
750	74.7996%
1000	74.4274%

Table A.10: $w_{tc} = 0.5$ Two-Level Threshold Accuracy Results

Nodes	$w_{th} = 0.25$	$w_{th} = 0$
10	89.9050%	89.8406%
25	77.4927%	77.4981%
50	74.3876%	74.4043%
75	75.0265%	75.0952%
100	71.9650%	71.9655%
250	82.3777%	82.4191%
500	81.1852%	81.1375%
750	80.0469%	80.0531%
1000	78.6831%	78.7081%

Table A.11: $w_{tc} = 0$ Two-Level Threshold Accuracy Results

Nodes	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	98.2858%	98.3150%	89.8958%
25	97.9543%	97.9920%	86.4282%
50	97.9543%	97.9374%	79.1906%
75	97.9180%	97.9316%	78.3866%
100	97.8536%	97.7936%	76.8391%
250	94.5771%	92.8136%	80.6034%
500	94.5078%	92.7967%	78.2691%
750	94.5310%	91.9236%	76.2498%
1000	94.5367%	92.0832%	75.7009%

Appendix B

EARLY ENERGY USAGE MODEL AND DATA

In order to estimate the energy usage for a given task we used the following equation:

$$E(X) = \frac{U_{mem}(X)}{T_{read}(X) + T_{write}(X)}\Gamma(X) \tag{B.1}$$

where $U_{mem}(X)$ is the memory usage in KB of the given computation node, from the Bitbrains data, $T_{read}(X)$ and $T_{write}(X)$ is the disk read and write throughput for the given computation node in KB/s, $\Gamma(X)$ is the thermal design power or the highest load that can be safely put on a device, for the given computational node in W or J/s.

For our model estimation the thermal design power values used are as follows:

Table B.1: Thermal Design Power Values

Node Type	**Value**
Fog Node	6.4 W
Super Node	12.8 W
Cloud Node	135 W

The device or fog node value is based on the value for a Raspberry Pi, thus making the supernode made of two Raspberry Pis simply double that value, and the cloud node is based on the drive that the D series virtual machines for Microsoft Azure are based on.

Table B.2: $w_{tc} = 1$ Energy (KJ) Results

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	10.0993	12.5117	169.8194	821.1099	821.1099
25	14.7651	15.1242	136.5136	620.2496	620.2496
50	17.1297	13.9137	440.6000	381.7943	381.7943
75	17.0826	15.7268	432.9548	356.3176	359.3176
100	14.6856	17.5547	414.2508	396.3559	396.3559
250	40.2329	0.0051	0.1951	201.1224	201.1224
500	37.0957	0.4205	0.1734	801.8347	801.8347
750	39.0564	1.1487	266.3971	812.1038	812.1038
1000	39.2174	2.5876	283.5099	196.3952	196.3952‘

Table B.3: $w_{tc} = 0.5$ Energy (KJ) Results

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	12.9381	170.0227	165.3737	165.7696	45.227
25	15.1147	136.7175	456.8694	454.0569	214.4668
50	16.5641	127.3061	377.6786	373.0254	289.7642
75	15.582	126.1386	321.689	321.2173	251.4573
100	15.4505	113.6892	346.4838	344.4424	273.8379
250	0.52684	0.00054367	704.7337	1763.5855	1675.1087
500	0.37892	0.19359	1152.5598	2431.6761	2366.3766
750	1.5772	270.984	1131.9977	8362.0884	8260.682
1000	5.2162	289.1315	781.905	6305.5147	6530.542

Table B.4: $w_{tc} = 0$ Energy (KJ) Results

Nodes	$w_{th} = 1$	$w_{th} = 0.75$	$w_{th} = 0.5$	$w_{th} = 0.25$	$w_{th} = 0$
10	12.4021	170.7098	167.2524	166.254	45.257
25	12.1644	479.7341	4585.3286	452.9033	214.3384
50	19.5942	436.5089	377.7801	380.6452	287.5699
75	15.3377	429.4731	319.9501	318.8343	253.5171
100	13.5923	418.1339	339.8786	343.4912	270.6524
250	0.47413	0.47913	1765.6317	1728.4356	1616.9008
500	1.2225	0.15205	2432.2541	2505.8426	2334.8112
750	1.3986	219.9253	8237.7034	8098.5255	8260.0161
1000	3.0613	240.7243	5997.5088	6460.7696	5842.0816

Appendix C

CODE DEMONSTRATION

C.1 Data Collection

C.1.1 Models

Create Fog Model

```
function model = CreatePSModel(task, powPref, memPref, diskPref, netPref, costPref,...
                               timPref, relPref,data,numNodes, timCostTrade)


x = zeros(numNodes,1);
y = zeros(numNodes,1);


q = FindClosestFactorization(numNodes);
w = 100/q(1);
v = 100/q(2);


for i = 1:q(1)
for j = 1:q(2)
x((i-1)*q(2)+j) = (i-1)*w;
y((i-1)*q(2)+j) = (j-1)*v;
end
end
```

```
n = numel(x);

empty_node.Name = [];
empty_node.Time = [];
empty_node.Cores = [];
empty_node.CPUCap = [];
empty_node.CPUUse = [];
empty_node.CPUPerc = [];
empty_node.MemCap = [];
empty_node.MemUse = [];
empty_node.ReadTP = [];
empty_node.WriteTP = [];
empty_node.RecTP = [];
empty_node.TransTP = [];
empty_node.Rel = [];
empty_node.Score = [];

best_val.Time = inf;
best_val.CPU = -inf;
best_val.Mem = -inf;
best_val.ReadTP = -inf;
best_val.WriteTP = -inf;
best_val.RecTP = -inf;
best_val.TransTP = -inf;

nodes = repmat(empty_node,n,1);

for i = 1:n
```

```
nodes(i).Name = i;
nodes(i).Cores = data(i).Entry.CPUCores(task);
nodes(i).CPUCap = data(i).Entry.CPUCapacity(task);
nodes(i).CPUUse = data(i).Entry.CPUUsageMHz(task);
nodes(i).CPUPerc = data(i).Entry.CPUUsagePerc(task);
nodes(i).MemCap = data(i).Entry.MemoryProvisionedKB(task);
nodes(i).MemUse = data(i).Entry.MemoryUsageKB(task);
nodes(i).ReadTP = data(i).Entry.DiskReadThroughputKB_s(task);
nodes(i).WriteTP = data(i).Entry.DiskWriteThroughputKB_s(task);
nodes(i).RecTP = data(i).Entry.NetworkRecievedThroughputKB_s(task);
nodes(i).TransTP = data(i).Entry.NetworkTransmittedThroughputKB_s(task);
nodes(i).Time=data(i).Entry.TimeStamp(task+1) - data(i).Entry.TimeStamp(task);
nodes(i).Rel=1;


if nodes(i).Time < best_val.Time
best_val.Time = nodes(i).Time;
end

if nodes(i).CPUCap > best_val.CPU
best_val.CPU = nodes(i).CPUCap;
end

if nodes(i).MemCap > best_val.Mem
best_val.Mem = nodes(i).MemCap;
end

if nodes(i).ReadTP > best_val.ReadTP
```

```
best_val.ReadTP = nodes(i).ReadTP;
end

if nodes(i).WriteTP > best_val.WriteTP
best_val.WriteTP = nodes(i).WriteTP;
end

if nodes(i).RecTP > best_val.RecTP
best_val.RecTP = nodes(i).RecTP;
end

if nodes(i).TransTP > best_val.TransTP
best_val.TransTP = nodes(i).TransTP;
end

end


scores = CalculateScore(nodes,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,best_val,timCostTrade);

for i = 1:n
nodes(i).Score = scores(i);
end


model.x = x;
model.y = y;
```

```
model.n = n;
model.nodes = nodes;
end
```

Create Cloud Model

```
function model = CreatePSCLDModel(task, powPref, memPref, diskPref, netPref,...
costPref,timPref, relPref,data,numNodes, timCostTrade)

x = zeros(numNodes,1);
y = zeros(numNodes,1);

q = FindClosestFactorization(numNodes);
w = 100/q(1);
v = 100/q(2);

for i = 1:q(1)
for j = 1:q(2)
x((i-1)*q(2)+j) = (i-1)*w;
y((i-1)*q(2)+j) = (j-1)*v;
end
end


n = numel(x);

empty_node.Name = [];
empty_node.Children = [];
empty_node.Time = [];
```

```
empty_node.Cores = [];
empty_node.CPUCap = [];
empty_node.CPUUse = [];
empty_node.CPUPerc = [];
empty_node.MemCap = [];
empty_node.MemUse = [];
empty_node.ReadTP = [];
empty_node.WriteTP = [];
empty_node.RecTP = [];
empty_node.TransTP = [];
empty_node.Rel = [];
empty_node.Score = [];

best_val.Time = inf;
best_val.CPU = -inf;
best_val.Mem = -inf;
best_val.ReadTP = -inf;
best_val.WriteTP = -inf;
best_val.RecTP = -inf;
best_val.TransTP = -inf;

nodes = repmat(empty_node,n,1);

for i = 1:n
nodes(i).Name = i;
for j = 1:n
if i ~= j
nodes(i).Children = [nodes(i).Children j];
```

```
end
end

nodes(i).Cores = data(i).Entry.CPUCores(task);
nodes(i).CPUCap = data(i).Entry.CPUCapacity(task);
nodes(i).CPUUse = data(i).Entry.CPUUsageMHz(task);
nodes(i).CPUPerc = data(i).Entry.CPUUsagePerc(task);
nodes(i).MemCap = data(i).Entry.MemoryProvisionedKB(task);
nodes(i).MemUse = data(i).Entry.MemoryUsageKB(task);
nodes(i).ReadTP = data(i).Entry.DiskReadThroughputKB_s(task);
nodes(i).WriteTP = data(i).Entry.DiskWriteThroughputKB_s(task);
nodes(i).RecTP = data(i).Entry.NetworkRecievedThroughputKB_s(task);
nodes(i).TransTP = data(i).Entry.NetworkTransmittedThroughputKB_s(task);
nodes(i).Time=data(i).Entry.TimeStamp(task+1) - data(i).Entry.TimeStamp(task);
nodes(i).Rel=1;


if nodes(i).Time < best_val.Time
best_val.Time = nodes(i).Time;
end

if nodes(i).CPUCap > best_val.CPU
best_val.CPU = nodes(i).CPUCap;
end

if nodes(i).MemCap > best_val.Mem
best_val.Mem = nodes(i).MemCap;
end
```

```
if nodes(i).ReadTP > best_val.ReadTP
best_val.ReadTP = nodes(i).ReadTP;
end

if nodes(i).WriteTP > best_val.WriteTP
best_val.WriteTP = nodes(i).WriteTP;
end

if nodes(i).RecTP > best_val.RecTP
best_val.RecTP = nodes(i).RecTP;
end

if nodes(i).TransTP > best_val.TransTP
best_val.TransTP = nodes(i).TransTP;
end

end


scores = CalculateScore(nodes,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,best_val,timCostTrade);

for i = 1:n
nodes(i).Score = scores(i)-(costPref * (1-timCostTrade));
end
```

```
model.x = x;
model.y = y;
model.n = n;
model.nodes = nodes;

end
```

C.1.2 Particle Swarm

Single Threshold Main File

```
clc;
clear;
close all;

%% Problem Definition
powPref = 0.2;
memPref = 0.2;
diskPref = 0.2;
netPref = 0.1;
costPref = 0.15;
timPref = 0.1;
relPref = 0.05;

tasks = 1;

total = powPref + memPref + diskPref + netPref + costPref + timPref + relPref;
if total ~= 1
disp('The chosen preferences are not valid they must total 1');
```

```
return;
end

timCostTrade = 0.5;
threshWeight = 1.0;

numTest = 1000;

numNodes = [10 25 50 75 100 250 500 750 1000];

currentDrive = pwd;

cldData.Entry = [];
CLDData = repmat(cldData,250,1);
SupData = repmat(cldData,251,1);

for i = 1:250
CLDData(i).Entry = FileRead([currentDrive '\rnd\2013-8\' num2str(i) '.csv'],tasks);
end

cldModel = CreatePSCLDModel(1,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,CLDData,250,timCostTrade);

for q = 1:numel(numNodes)
data.Entry = [];
Data = repmat(data,numNodes(q),1);
factor = FindClosestFactorization(numNodes(q));
```

```
for i = 1:numNodes(q)
Data(i).Entry = FileRead([currentDrive '\FastStorage\2013-8\' num2str(i)...
'.csv'],tasks);
end

optimal = 0;
minScore = inf;

model = CreatePSModel(1,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,Data,numNodes(q), timCostTrade);
disp('');
disp("Model Ready");
for z = 1:cldModel.n
if optimal < cldModel.nodes(z).Score
optimal = cldModel.nodes(z).Score;
end
if minScore > cldModel.nodes(z).Score && cldModel.nodes(z).Score ~= 0
minScore = cldModel.nodes(z).Score;
end
end

for z = 1:model.n
if optimal < model.nodes(z).Score
optimal = model.nodes(z).Score;
end
if minScore > model.nodes(z).Score && model.nodes(z).Score ~= 0
minScore = model.nodes(z).Score;
end
```

```
end

threshold = optimal-((optimal-minScore)*(1- threshWeight));

%threshold = optimal * threshWeight;

loopTim = zeros(numTest,1);
t = zeros(100,1);

best.Score = [];
best.Position=[];

Best = repmat(best,tasks,1);

%% Parameters of Optimization

maxIt = factor(1);

nPop=factor(2);

wdamp = 0.99;
c1 = 2;
c2 = 2;

%% Initilization

empty_particle.Position = [];
empty_particle.Velocity = [];
```

```
empty_particle.Score = [];
empty_particle.Best.Position = [];
empty_particle.Best.Score = [];

particle = repmat(empty_particle, nPop, 1);

%% Run Algorithm

starttime = tic;
for task = 1:tasks
Perc = 0;
Cost = 0;
Energy=0;
for z = 1:100
avgCost = 0;
avgPerc = 0;
avgEnergy = 0;
threshCount = 0;
for n = 1:numTest
solutionFound = false;

VarMin = 1;
VarMax = model.n;

tic;
i = randi(VarMax);
if model.nodes(i).Score >= threshold
GlobalBest.Score = model.nodes(i).Score;
```

```
GlobalBest.Position = i;
solutionFound = true;
loopTim(n) = toc;
threshCount = threshCount+1;
end

if not(solutionFound)
tic
w = 1;
GlobalBest.Score = -inf;

for k=1:nPop

particle(k).Position = randi(VarMax);

particle(k).Velocity = zeros(1);

particle(k).Score = model.nodes(particle(k).Position).Score;

particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end
end
```

```
for it = 1:maxIt
for k = 1:nPop
particle(k).Velocity = w*particle(k).Velocity+ c1*rand(1)...
*(particle(k).Best.Position - particle(k).Position) + c2*rand(1)*...
(GlobalBest.Position - particle(k).Position);

particle(k).Position = particle(k).Position + round(particle(k).Velocity);

while particle(k).Position > model.n ||...
particle(k).Position <= 0
if particle(k).Position > model.n
particle(k).Position = particle(k).Position - ...
model.n;
elseif particle(k).Position <= 0
particle(k).Position = particle(k).Position +...
model.n;
end
end

particle(k).Score = model.nodes(particle(k).Position).Score;

if particle(k).Score > particle(k).Best.Score
particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;
end

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
```

```
end

if GlobalBest.Score >= threshold
break;
end

w = w*wdamp;
end

% Store the Best Cost Value
if GlobalBest.Score >= threshold
threshCount = threshCount+1;
break;
end

end

Best(task).Score = GlobalBest.Score;
Best(task).Position = GlobalBest.Position;

end

if GlobalBest.Score < threshold
node1= model.nodes(GlobalBest.Position);
node2.Score = 0;
for k=1:nPop
if k~=GlobalBest.Position
if model.nodes(k).Score>node2.Score
```

```
node2 = model.nodes(k);
end
end
end
SuperNode.CPUCores = node1.Cores+node2.Cores;
SuperNode.CPUCapacity = node1.CPUCap+node2.CPUCap;
SuperNode.CPUUsageMHz = node1.CPUUse+node2.CPUUse;
SuperNode.CPUUsagePerc = SuperNode.CPUUsageMHz/...
SuperNode.CPUCapacity;
SuperNode.MemoryProvisionedKB = node1.MemCap+node2.MemCap;
SuperNode.MemoryUsageKB = node1.MemUse+node2.MemUse;
SuperNode.DiskReadThroughputKB_s = node1.ReadTP+node2.ReadTP;
SuperNode.DiskWriteThroughputKB_s = node1.WriteTP+node2.WriteTP;
SuperNode.NetworkRecievedThroughputKB_s = node1.RecTP+node2.RecTP;
SuperNode.NetworkTransmittedThroughputKB_s = node1.TransTP+...
node2.TransTP;
SupData(1).Entry = SuperNode;
SupData(2:251) = CLDData;
end

if GlobalBest.Score >= threshold
accPerc = GlobalBest.Score/optimal*100;
time = model.nodes(GlobalBest.Position).MemUse/...
(model.nodes(GlobalBest.Position).ReadTP +...
model.nodes(GlobalBest.Position).WriteTP);
if isnan(time) || time==inf
time = 300;
end
```

```
avgEnergy = avgEnergy+(6.4*time);
else
SupModel = CreateSupModel(1,powPref,memPref,diskPref,netPref,...
costPref,timPref,relPref,SupData,251, timCostTrade);
if SupModel.nodes(1).Score>=threshold
accPerc = SupModel.nodes(1).Score/optimal*100;
time = SupModel.nodes(1).MemUse/(SupModel.nodes(1).ReadTP...
+ SupModel.nodes(1).WriteTP);
if isnan(time) || time==inf
time = 300;
end
avgEnergy = avgEnergy+(12.8*time);
else
time = SupModel.nodes(1).MemUse/(SupModel.nodes(1).ReadTP...
+ SupModel.nodes(1).WriteTP);
if isnan(time) || time==inf
time = 300;
end
cld = CloudPS(cldModel,threshold,optimal,...
SupModel.nodes(1).Score,time);
accPerc = cld(1);
avgCost = avgCost + cld(2);
avgEnergy = avgEnergy+cld(3);
end
end
loopTim(n)=toc;
avgPerc = avgPerc+accPerc;
end
```

```
t(z) = sum(loopTim)/numTest;
avgPerc = avgPerc/numTest;
avgCost = avgCost/numTest;
avgEnergy = avgEnergy/numTest;
Perc = Perc + avgPerc;
Cost = Cost + avgCost;
Energy = Energy + avgEnergy;
end
avgTim = sum(t)/100;
Perc = Perc/100;
Cost = Cost/100;
Energy = Energy/100/1000;
disp("Task : " + num2str(task) + " Nodes: " + numNodes(q) + "...
Accuracy: " + Perc);
disp("Average Execution Time: " + avgTim);
disp("Estimated Avergage Monetary Cost: \$" + Cost);
disp("Estimated Average Energy Usage: " + Energy + " kJ");
end

finish = toc(starttime);

disp("Full Execution Time: " + finish);
end
```

Two Threshold Main File

```
clc;
clear;
close all;
```

```
%% Problem Definition
powPref = 0.2;
memPref = 0.2;
diskPref = 0.2;
netPref = 0.1;
costPref = 0.15;
timPref = 0.1;
relPref = 0.05;

tasks = 1;

total = powPref + memPref + diskPref + netPref + costPref + timPref + relPref;
if total ~= 1
disp('The chosen preferences are not valid they must total 1');
return;
end

timCostTrade = 0.0;
threshWeight = 0.0;
minPerc = 0.6;

numTest = 1000;

numNodes = [10 25 50 75 100 250 500 750 1000];

currentDrive = pwd;
```

```
cldData.Entry = [];
CLDData = repmat(cldData,250,1);
SupData = repmat(cldData,251,1);

for i = 1:250
CLDData(i).Entry = FileRead([currentDrive '\rnd\2013-8\' num2str(i) '.csv'],tasks);
end

cldModel = CreatePSCLDModel(1,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,CLDData,250,timCostTrade);

for q = 1:numel(numNodes)
data.Entry = [];
Data = repmat(data,numNodes(q),1);
factor = FindClosestFactorization(numNodes(q));

for i = 1:numNodes(q)
Data(i).Entry = FileRead([currentDrive '\FastStorage\2013-8\'...
num2str(i) '.csv'],tasks);
end


optimal = 0;
minScore = inf;

model = CreatePSModel(1,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,Data,numNodes(q), timCostTrade);
disp('');
```

```
disp("Model Ready");
for z = 1:cldModel.n
if optimal < cldModel.nodes(z).Score
optimal = cldModel.nodes(z).Score;
end
if minScore > cldModel.nodes(z).Score && cldModel.nodes(z).Score ~= 0
minScore = cldModel.nodes(z).Score;
end
end

for z = 1:model.n
if optimal < model.nodes(z).Score
optimal = model.nodes(z).Score;
end
if minScore > model.nodes(z).Score && model.nodes(z).Score ~= 0
minScore = model.nodes(z).Score;
end
end

threshold = optimal-((optimal-minScore)*(1- threshWeight));

minThresh = optimal * minPerc;


loopTim = zeros(numTest,1);
t = zeros(100,1);

best.Score = [];
```

```
best.Position=[];

Best = repmat(best,tasks,1);

%% Parameters of Optimization

maxIt = factor(1);

nPop=factor(2);


wdamp = 0.99;
c1 = 2;
c2 = 2;

%% Initilization

empty_particle.Position = [];
empty_particle.Velocity = [];
empty_particle.Score = [];
empty_particle.Best.Position = [];
empty_particle.Best.Score = [];

particle = repmat(empty_particle, nPop, 1);

%% Run Algorithm

starttime = tic;
```

```
for task = 1:tasks
Perc = 0;
Cost = 0;
Energy=0;
for z = 1:100
avgCost = 0;
avgPerc = 0;
avgEnergy = 0;
threshCount = 0;
for n = 1:numTest
solutionFound = false;

VarMin = 1;
VarMax = model.n;

tic;
i = randi(VarMax);
if model.nodes(i).Score >= threshold &&...
model.nodes(i).Score >= minThresh
GlobalBest.Score = model.nodes(i).Score;
GlobalBest.Position = i;
solutionFound = true;
loopTim(n) = toc;
threshCount = threshCount+1;
end

if not(solutionFound)
tic
```

```
w = 1;
GlobalBest.Score = -inf;

for k=1:nPop

particle(k).Position = randi(VarMax);

particle(k).Velocity = zeros(1);

particle(k).Score = model.nodes(particle(k).Position).Score;

particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end
end



for it = 1:maxIt
for k = 1:nPop
particle(k).Velocity = w*particle(k).Velocity...
+ c1*rand(1).*(particle(k).Best.Position...
- particle(k).Position + ...
c2*rand(1).*(GlobalBest.Position -...
particle(k).Position);
```

```
particle(k).Position = particle(k).Position +...
round(particle(k).Velocity);

while particle(k).Position > model.n ||...
particle(k).Position <= 0
if particle(k).Position > model.n
particle(k).Position = particle(k).Position -...
model.n;
elseif particle(k).Position <= 0
particle(k).Position = particle(k).Position +...
model.n;
end
end

particle(k).Score = model.nodes(particle(k).Position).Score;


if particle(k).Score > particle(k).Best.Score
particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;
end

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end

if GlobalBest.Score >= threshold && GlobalBest.Score >= minThresh
break;
```

```
end

w = w*wdamp;
end

% Store the Best Cost Value
if GlobalBest.Score >= threshold && GlobalBest.Score >= minThresh
threshCount = threshCount+1;
break;
end

end

Best(task).Score = GlobalBest.Score;
Best(task).Position = GlobalBest.Position;

end

if GlobalBest.Score < threshold || GlobalBest.Score < minThresh
node1= model.nodes(GlobalBest.Position);
node2.Score = 0;
for k=1:numNodes(q)
if k~=GlobalBest.Position
if model.nodes(k).Score>node2.Score
node2 = model.nodes(k);
end
end
end
```

```
SuperNode.CPUCores = node1.Cores+node2.Cores;
SuperNode.CPUCapacity = node1.CPUCap+node2.CPUCap;
SuperNode.CPUUsageMHz = node1.CPUUse+node2.CPUUse;
SuperNode.CPUUsagePerc = SuperNode.CPUUsageMHz/SuperNode.CPUCapacity;
SuperNode.MemoryProvisionedKB = node1.MemCap+node2.MemCap;
SuperNode.MemoryUsageKB = node1.MemUse+node2.MemUse;
SuperNode.DiskReadThroughputKB_s = node1.ReadTP+node2.ReadTP;
SuperNode.DiskWriteThroughputKB_s = node1.WriteTP+node2.WriteTP;
SuperNode.NetworkRecievedThroughputKB_s = node1.RecTP+node2.RecTP;
SuperNode.NetworkTransmittedThroughputKB_s = node1.TransTP+node2.TransTP;
SupData(1).Entry = SuperNode;
SupData(2:251) = CLDData;
end

if GlobalBest.Score >= threshold && GlobalBest.Score >= minThresh
accPerc = GlobalBest.Score/optimal*100;
time = model.nodes(GlobalBest.Position).MemUse/...
(model.nodes(GlobalBest.Position).ReadTP + model.nodes(GlobalBest.Position).WriteTP);
if isnan(time) || time==inf
time = 300;
end
avgEnergy = avgEnergy+(6.4*time);
else
SupModel = CreateSupModel(1,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,SupData,251, timCostTrade);
if SupModel.nodes(1).Score>=threshold && SupModel.nodes(1).Score >= minThresh
accPerc = SupModel.nodes(1).Score/optimal*100;
time = SupModel.nodes(1).MemUse/...
```

```
(SupModel.nodes(1).ReadTP + SupModel.nodes(1).WriteTP);
if isnan(time) || time==inf
time = 300;
end
avgEnergy = avgEnergy+(12.8*time);
else
cld = TwoThreshCloudPS(cldModel,threshold, minThresh, optimal);
accPerc = cld(1);
avgCost = avgCost + cld(2);
avgEnergy = avgEnergy+cld(3);

end
end
loopTim(n)=toc;
avgPerc = avgPerc+accPerc;
end
t(z) = sum(loopTim)/numTest;
avgPerc = avgPerc/numTest;
avgCost = avgCost/numTest;
avgEnergy = avgEnergy/numTest;
Perc = Perc + avgPerc;
Cost = Cost + avgCost;
Energy = Energy + avgEnergy;
end
avgTim = sum(t)/100;
Perc = Perc/100;
Cost = Cost/100;
Energy = Energy/100/1000;
```

```
disp("Task : " + num2str(task) + " Nodes: " + numNodes(q) + " Accuracy: " + Perc);
disp("Average Execution Time: " + avgTim);
disp("Estimated Avergage Monetary Cost: $" + Cost);
disp("Estimated Average Energy Usage: " + Energy + " kJ");
end

finish = toc(starttime);

disp("Full Execution Time: " + finish);
end
```

Cloud File

```
function cld=CloudPS(model, threshold, optimal,superSCR,tme)

numTest = 1;

%% Parameters of Optimization

factor = FindClosestFactorization(model.n);

maxIt = factor(1);

nPop=factor(2);


wdamp = 0.99;
c1 = 2;
c2 = 2;
```

```
%% Initilization

empty_particle.Position = [];
empty_particle.Score = [];
empty_particle.Velocity = [];
empty_particle.Best.Position = [];
empty_particle.Best.Score = [];
empty_particle.Worst.Score = [];

particle = repmat(empty_particle, nPop, 1);

%% Run Algorithm
for n = 1:numTest
solutionFound = false;
VarMax = model.n;

i = randi(VarMax);
if model.nodes(i).Score >= threshold
GlobalBest.Score = model.nodes(i).Score;
GlobalBest.Position = i;
solutionFound = true;
end

if not(solutionFound)
w = 1;
GlobalBest.Score = -inf;
```

```
for k=1:nPop

particle(k).Position = randi(VarMax);

particle(k).Velocity = zeros(1);

particle(k).Score = model.nodes(particle(k).Position).Score;

particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end
end

for it = 1:maxIt
for k = 1:nPop
particle(k).Velocity = w*particle(k).Velocity...
+ c1*rand(1).*(particle(k).Best.Position - particle(k).Position)....
+ c2*rand(1).*(GlobalBest.Position - particle(k).Position);

particle(k).Position = particle(k).Position + round(particle(k).Velocity);

while particle(k).Position > model.n || particle(k).Position <= 0
if particle(k).Position > model.n
particle(k).Position = particle(k).Position - model.n;
elseif particle(k).Position <= 0
```

```
particle(k).Position = particle(k).Position + model.n;
end
end

particle(k).Score = model.nodes(particle(k).Position).Score;


if particle(k).Score > particle(k).Best.Score
particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;
end

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end

if GlobalBest.Score >= threshold
break;
end
end

% Store the Best Cost Value
if GlobalBest.Score >= threshold
break;
end
w = w*wdamp;
end
end
```

```
if GlobalBest.Score > superSCR
accPerc = GlobalBest.Score/optimal*100;
cost = model.nodes(GlobalBest.Position).Cores*0.126+0.05;
time = (model.nodes(GlobalBest.Position).MemUse/...
(model.nodes(GlobalBest.Position).ReadTP + model.nodes(GlobalBest.Position).WriteTP));
if isnan(time) || time==inf
time = 300;
end
cpu = 135*time;
disk = 10*time;
energy = cpu + disk;
else
accPerc = superSCR/optimal*100;
GlobalBest.Score = superSCR;
GlobalBest.Position = -1;
energy = 12.8*tme;
cost = 0;
end
cld(1) = accPerc;
cld(2) = cost;
cld(3) = energy;
cld(4) = GlobalBest.Score;
cld(5) = GlobalBest.Position;
end
end
```

C.2 Visual Demonstration

C.2.1 Particle Swarm

Demonstration Main

```
clc;
clear;
close all;

%% Problem Definition
powPref = 0.2;
memPref = 0.2;
diskPref = 0.2;
netPref = 0.1;
costPref = 0.15;
timPref = 0.1;
relPref = 0.05;

tasks = 1;

total = powPref + memPref + diskPref + netPref + costPref + timPref + relPref;
if total ~= 1
disp('The chosen preferences are not valid they must total 1');
return;
end


disp('Enter your weight towards execution time versus cost.');
```

```
disp('0 puts highest priority on minumum cost, while 1 puts the highest ...
priority on exectution time.');
timCostTrade = input('Enter a value between 0 and 1:');

disp('Enter your weight towards accuracy versus allocation time.');
disp('0 puts highest priority on allocation time , while 1 puts the highest ...
priority on accuracy.');
threshWeight = input('Enter a value between 0 and 1:');

numNodes = 250;

currentDrive = pwd;

cldData.Entry = [];
CLDData = repmat(cldData,250,1);
SupData = repmat(cldData,251,1);

for i = 1:250
CLDData(i).Entry = FileRead([currentDrive '\rnd\2013-8\' num2str(i) '.csv'],tasks);
end

cldModel = CreatePSCLDModel(1,powPref,memPref,diskPref,netPref,costPref,timPref...
,relPref,CLDData,250,timCostTrade);

data.Entry = [];
Data = repmat(data,numNodes,1);
factor = FindClosestFactorization(numNodes);
bestTour = [];
```

```
for i = 1:numNodes
Data(i).Entry = FileRead([currentDrive '\FastStorage\2013-8\'...
num2str(i) '.csv'],tasks);
end


optimal = 0;
minScore = inf;

model = CreatePSModel(1,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,Data,numNodes, timCostTrade);

starttime = tic;
for z = 1:cldModel.n
if optimal < cldModel.nodes(z).Score
optimal = cldModel.nodes(z).Score;
end
if minScore > cldModel.nodes(z).Score && cldModel.nodes(z).Score ~= 0
minScore = cldModel.nodes(z).Score;
end
end

for z = 1:model.n
if optimal < model.nodes(z).Score
optimal = model.nodes(z).Score;
end
if minScore > model.nodes(z).Score && model.nodes(z).Score ~= 0
```

```
minScore = model.nodes(z).Score;
end
end

threshold = optimal-((optimal-minScore)*(1- threshWeight));

best.Score = [];
best.Position=[];

Best = repmat(best,tasks,1);

%% Parameters of Optimization

maxIt = factor(1);

nPop=factor(2);

wdamp = 0.99;
c1 = 2;
c2 = 2;

%% Initilization

empty_particle.Position = [];
empty_particle.Velocity = [];
empty_particle.Score = [];
empty_particle.Best.Position = [];
empty_particle.Best.Score = [];
```

```
empty_particle.Tour = [];

particle = repmat(empty_particle, nPop, 1);

%% Run Algorithm
Cost = 0;
supPos = 0;

solutionFound = false;

VarMin = 1;
VarMax = model.n;

i = randi(VarMax);
if model.nodes(i).Score >= threshold
GlobalBest.Score = model.nodes(i).Score;
GlobalBest.Position = i;
solutionFound = true;
Finish= toc(starttime);
end

if not(solutionFound)
w = 1;
GlobalBest.Score = -inf;

for k=1:nPop

particle(k).Position = randi(VarMax);
```

```
particle(k).Tour = particle(k).Position;

particle(k).Velocity = zeros(1);

particle(k).Score = model.nodes(particle(k).Position).Score;

particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end
end

bestTour = GlobalBest.Position;

for it = 1:maxIt
for k = 1:nPop
particle(k).Velocity = w*particle(k).Velocity...
+ c1*rand(1).*(particle(k).Best.Position - particle(k).Position)....
+ c2*rand(1).*(GlobalBest.Position - particle(k).Position);

particle(k).Position = particle(k).Position + round(particle(k).Velocity);


while particle(k).Position > model.n || particle(k).Position <= 0
if particle(k).Position > model.n
particle(k).Position = particle(k).Position - model.n;
```

```
elseif particle(k).Position <= 0
particle(k).Position = particle(k).Position + model.n;
end
end

particle(k).Tour = [particle(k).Tour particle(k).Position];

particle(k).Score = model.nodes(particle(k).Position).Score;


if particle(k).Score > particle(k).Best.Score
particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;
end

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end

if GlobalBest.Score >= threshold
break;
end

w = w*wdamp;
end

bestTour = [bestTour GlobalBest.Position];
```

```
f=figure(1);
f.Name = ['Best Ant Path Iteration: ' num2str(it) ' Score: '...
num2str(GlobalBest.Score) ' Position: ' num2str(GlobalBest.Position)];
if(GlobalBest.Score >= threshold)
f.Name = ['Threshold Best Ant Paths Iteration: ' num2str(it) ' Score: '...
num2str(GlobalBest.Score) ' Position: ' num2str(GlobalBest.Position)];
end
f.NumberTitle = 'off';
f.OuterPosition = [0 864/2-10 768*2 864/2+10];
PlotMovement(bestTour,model,it+1);

f=figure(2);
f.Name = ['Ant Paths Iteration: ' num2str(it) ' Best Score: '...
num2str(GlobalBest.Score) ' Best Position: ' num2str(GlobalBest.Position)];
if(GlobalBest.Score >= threshold)
f.Name = ['Threshold Ant Paths Iteration: ' num2str(it) ' Best Score: '...
num2str(GlobalBest.Score) ' Best Position: ' num2str(GlobalBest.Position)];
end
f.NumberTitle = 'off';
f.OuterPosition = [0 10 768*2 864/2];

for k=1:2.5:nPop
val = round(k);
subplot(2,5,round((val-1)/2.5+1));
PlotMovement(particle(val).Tour,model,it+1);
title(['Ant: ' num2str(val)]);
end
```

```
% Store the Best Cost Value
if GlobalBest.Score >= threshold
finish = toc(starttime);
break;
end

end
end

disp("");

if GlobalBest.Score < threshold
node1= model.nodes(GlobalBest.Position);
node2.Score = 0;
for k=1:nPop
if k~=GlobalBest.Position
if model.nodes(k).Score>node2.Score
node2 = model.nodes(k);
supPos = k;
end
end
end
SuperNode.CPUCores = node1.Cores+node2.Cores;
SuperNode.CPUCapacity = node1.CPUCap+node2.CPUCap;
SuperNode.CPUUsageMHz = node1.CPUUse+node2.CPUUse;
SuperNode.CPUUsagePerc = SuperNode.CPUUsageMHz/SuperNode.CPUCapacity;
SuperNode.MemoryProvisionedKB = node1.MemCap+node2.MemCap;
SuperNode.MemoryUsageKB = node1.MemUse+node2.MemUse;
```

```
SuperNode.DiskReadThroughputKB_s = node1.ReadTP+node2.ReadTP;
SuperNode.DiskWriteThroughputKB_s = node1.WriteTP+node2.WriteTP;
SuperNode.NetworkRecievedThroughputKB_s = node1.RecTP+node2.RecTP;
SuperNode.NetworkTransmittedThroughputKB_s = node1.TransTP+node2.TransTP;
SupData(1).Entry = SuperNode;
SupData(2:251) = CLDData;
end

if GlobalBest.Score >= threshold
Perc = GlobalBest.Score/optimal*100;
time = model.nodes(GlobalBest.Position).MemUse/...
(model.nodes(GlobalBest.Position).ReadTP + model.nodes(GlobalBest.Position).WriteTP);
if isnan(time) || time==inf
time = 300;
end
Energy = 6.4*time;
disp("Nodes: " + numNodes + " Accuracy: " + Perc + "%");
disp("Solution Score: " + GlobalBest.Score);
disp("Fog Position: " + GlobalBest.Position);
finish = toc(starttime);
else
SupModel = CreateSupModel(1,powPref,memPref,diskPref,netPref,costPref,timPref,...
relPref,SupData,251, timCostTrade);
if SupModel.nodes(1).Score>=threshold
Perc = SupModel.nodes(1).Score/optimal*100;
time = SupModel.nodes(1).MemUse/...
(SupModel.nodes(1).ReadTP + SupModel.nodes(1).WriteTP);
if isnan(time) || time==inf
```

```
time = 300;
end
Energy = 12.8*time;
disp("Nodes: " + numNodes + " Accuracy: " + Perc + "%");
disp("Solution Score: " + SupModel.nodes(1).Score);
disp("Fog Nodes Used: " + GlobalBest.Position + " and " + supPos);
finish = toc(starttime);
else
cld = DispCloudPS(cldModel,threshold,optimal,SupModel.nodes(1).Score);
Perc = cld(1);
Cost = cld(2);
Energy = cld(3);
disp("Nodes: " + numNodes + " Accuracy: " + Perc + "%");
disp("Solution Score: " + cld(4));
disp("Cloud Position: " + cld(5));
finish = toc(starttime);
end
end

% disp("Allocation Time: " + finish);
disp("Estimated Monetary Cost: $" + Cost);
disp("Estimated Energy Usage: " + Energy + " kJ");
```

Cloud Demonstration File

```
function cld=DispCloudPS(model, threshold, optimal, superScore)
numTest = 1;
cost = 0;
```

```
%% Parameters of Optimization

factor = FindClosestFactorization(model.n);

maxIt = factor(1);

nPop=factor(2);

wdamp = 0.99;
c1 = 2;
c2 = 2;

%% Initilization

empty_particle.Position = [];
empty_particle.Score = [];
empty_particle.Velocity = [];
empty_particle.Best.Position = [];
empty_particle.Best.Score = [];
empty_particle.Worst.Score = [];
empty_particle.Tour = [];

particle = repmat(empty_particle, nPop, 1);

%% Run Algorithm
for n = 1:numTest
solutionFound = false;
VarMax = model.n;
```

```
i = randi(VarMax);
if model.nodes(i).Score >= threshold
GlobalBest.Score = model.nodes(i).Score;
GlobalBest.Position = i;
solutionFound = true;
end

if not(solutionFound)
w = 1;
GlobalBest.Score = -inf;

for k=1:nPop

particle(k).Position = randi(VarMax);
particle(k).Tour = particle(k).Position;

particle(k).Velocity = zeros(1);

particle(k).Score = model.nodes(particle(k).Position).Score;

particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;

if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end
end
```

```
for it = 1:maxIt
for k = 1:nPop
particle(k).Velocity = w*particle(k).Velocity...
+ c1*rand(1).*(particle(k).Best.Position - particle(k).Position)....
+ c2*rand(1).*(GlobalBest.Position - particle(k).Position);

particle(k).Position = particle(k).Position + round(particle(k).Velocity);

while particle(k).Position > model.n || particle(k).Position <= 0
if particle(k).Position > model.n
particle(k).Position = particle(k).Position - model.n;
elseif particle(k).Position <= 0
particle(k).Position = particle(k).Position + model.n;
end
end

particle(k).Tour = [particle(k).Tour particle(k).Position];

particle(k).Score = model.nodes(particle(k).Position).Score;


if particle(k).Score > particle(k).Best.Score
particle(k).Best.Position = particle(k).Position;
particle(k).Best.Score = particle(k).Score;
end
```

```
if particle(k).Best.Score > GlobalBest.Score
GlobalBest = particle(k).Best;
end

if GlobalBest.Score >= threshold
break;
end
end

f=figure(3);
f.Name = ['Cloud Ant Paths Iteration: ' num2str(it) ' Best Score: '...
num2str(GlobalBest.Score) ' Best Position: ' num2str(GlobalBest.Position)];
if(GlobalBest.Score >= threshold)
f.Name = ['Threshold Cloud Ant Paths Iteration: ' num2str(it) ' Best Score: '...
num2str(GlobalBest.Score) ' Best Position: ' num2str(GlobalBest.Position)];
end
f.NumberTitle = 'off';
f.OuterPosition = [0 0 768*2 864];

for k=1:nPop
subplot(5,5,k);
PlotMovement(particle(k).Tour,model,it+1);
title(['Ant: ' num2str(k)]);
end

% Store the Best Cost Value
if GlobalBest.Score >= threshold
break;
```

```
end
w = w*wdamp;
end
end
if GlobalBest.Score > superScore
accPerc = GlobalBest.Score/optimal*100;
cost = model.nodes(GlobalBest.Position).Cores*0.126+0.05;
time = (model.nodes(GlobalBest.Position).MemUse/...
(model.nodes(GlobalBest.Position).ReadTP + model.nodes(GlobalBest.Position).WriteTP));
if isnan(time) || time==inf
time = 300;
end
else
disp("SuperNode Used")
accPerc = superScore/optimal*100;
GlobalBest.Score = superScore;
GlobalBest.Position = -1;
time = 300;
end
cpu = 135*time;
disk = 10*time;
energy = cpu + disk;
cld(1) = accPerc;
cld(2) = cost;
cld(3) = energy;
cld(4) = GlobalBest.Score;
cld(5) = GlobalBest.Position;
end
```

```
end
```

C.3 Additional Files

Calculate Score

```
function score = CalculateScore(nodes,powPref,memPref,diskPref,netPref,costPref,...
timPref,relPref,best,timCostTrade)

n = numel(nodes);

score = zeros(n,1);

CPU = zeros(n,1);
Mem = zeros(n,1);
Read = zeros(n,1);
Write = zeros(n,1);
Rec = zeros(n,1);
Trans = zeros(n,1);

pow = zeros(n,1);
mem = zeros(n,1);
disk = zeros(n,1);
net = zeros(n,1);
time = zeros(n,1);
rel = zeros(n,1);


for i = 1:n
```

```
CPU(i) = nodes(i).CPUCap;
Mem(i) = nodes(i).MemCap;
Read(i) = nodes(i).ReadTP;
Write(i) = nodes(i).WriteTP;
Rec(i) = nodes(i).RecTP;
Trans(i) = nodes(i).TransTP;


if isempty(CPU(i))
CPU(i)=0;
end

if isempty(Mem(i))
Mem(i)=0;
end

if isempty(Read(i))
Read(i) = 0;
end

if isempty(Write(i))
Write(i) = 0;
end

if isempty(Rec(i))
Rec(i) = 0;
end
```

```
if isempty(Trans(i))
Trans(i) = 0;
end

pow(i) = (CPU(i) / best.CPU * (100-nodes(i).CPUPerc)/100);
mem(i) = (Mem(i)/best.Mem * (1 - (nodes(i).MemUse/nodes(i).MemCap)));
disk(i) = (Read(i)/best.ReadTP + Write(i)/best.WriteTP);
net(i) = (Rec(i)/best.RecTP + Trans(i)/best.TransTP);
time(i) = (1-((nodes(i).Time-best.Time)/best.Time));
rel(i) = nodes(i).Rel;
end

for i = 1:n

if isnan(pow(i)) || isempty(pow(i)) || pow(i) == 0
score(i) = 0;
continue;
end

if isnan(mem(i)) || isempty(mem(i)) || mem(i) == 0
score(i) = 0;
continue;
end

if isnan(disk(i))
disk(i) = 0;
end
```

```
if isnan(net(i))
net(i) = 0;
end

if isnan(time(i))
time(i) = 0;
end

p = powPref *((pow(i) - min(pow))/(max(pow)-min(pow)));
m = memPref *((mem(i) - min(mem))/(max(mem)-min(mem)));
d = diskPref *((disk(i) - min(disk))/(max(disk)-min(disk)));
n = netPref *((net(i) - min(net))/(max(net)-min(net)));
%         t = timPref *((time(i) - min(time))/(max(time)-min(time)));
t = timPref * 1;
%         r = relPref *((rel(i) - min(rel))/(max(rel)-min(rel)));
r = relPref *1;

if isnan(p)
score(i) = 0;
continue;
end

if isnan(m)
score(i) = 0;
continue;
end

if isnan(d)
```

```
d = 0;
end

if isnan(n)
n = 0;
end

if isnan(r)
r = 0;
end

score(i) = p + m + d + n + t + r;

if isempty(score(i))
score(i) = 0;
end

if isnan(score(i))
score(i) = 0;
end

if score(i)~=0
score(i) = score(i)+(costPref * (1-timCostTrade));
end

end
end
```

Create Super Node Model

```
function model = CreateSupModel(task, powPref, memPref, diskPref, netPref,...
costPref, timPref, relPref,data,numNodes, timCostTrade)

x = zeros(numNodes,1);
y = zeros(numNodes,1);

q = FindClosestFactorization(numNodes);
w = 100/q(1);
v = 100/q(2);

for i = 1:q(1)
for j = 1:q(2)
x((i-1)*q(2)+j) = (i-1)*w;
y((i-1)*q(2)+j) = (j-1)*v;
end
end


n = numel(x);

empty_node.Name = [];
empty_node.Time = [];
empty_node.Cores = [];
empty_node.CPUCap = [];
empty_node.CPUUse = [];
empty_node.CPUPerc = [];
```

```
empty_node.MemCap = [];
empty_node.MemUse = [];
empty_node.ReadTP = [];
empty_node.WriteTP = [];
empty_node.RecTP = [];
empty_node.TransTP = [];
empty_node.Rel = [];
empty_node.Score = [];

best_val.Time = inf;
best_val.CPU = -inf;
best_val.Mem = -inf;
best_val.ReadTP = -inf;
best_val.WriteTP = -inf;
best_val.RecTP = -inf;
best_val.TransTP = -inf;

nodes = repmat(empty_node,n,1);

for i = 1:n
nodes(i).Name = i;
if i~=1
nodes(i).Cores = data(i).Entry.CPUCores(task);
nodes(i).CPUCap = data(i).Entry.CPUCapacity(task);
nodes(i).CPUUse = data(i).Entry.CPUUsageMHz(task);
nodes(i).CPUPerc = data(i).Entry.CPUUsagePerc(task);
nodes(i).MemCap = data(i).Entry.MemoryProvisionedKB(task);
nodes(i).MemUse = data(i).Entry.MemoryUsageKB(task);
```

```
nodes(i).ReadTP = data(i).Entry.DiskReadThroughputKB_s(task);
nodes(i).WriteTP = data(i).Entry.DiskWriteThroughputKB_s(task);
nodes(i).RecTP = data(i).Entry.NetworkRecievedThroughputKB_s(task);
nodes(i).TransTP = data(i).Entry.NetworkTransmittedThroughputKB_s(task);
nodes(i).Time = data(i).Entry.TimeStamp(task+1) - data(i).Entry.TimeStamp(task);
else
nodes(i).Cores = data(i).Entry.CPUCores;
nodes(i).CPUCap = data(i).Entry.CPUCapacity;
nodes(i).CPUUse = data(i).Entry.CPUUsageMHz;
nodes(i).CPUPerc = data(i).Entry.CPUUsagePerc;
nodes(i).MemCap = data(i).Entry.MemoryProvisionedKB;
nodes(i).MemUse = data(i).Entry.MemoryUsageKB;
nodes(i).ReadTP = data(i).Entry.DiskReadThroughputKB_s;
nodes(i).WriteTP = data(i).Entry.DiskWriteThroughputKB_s;
nodes(i).RecTP = data(i).Entry.NetworkRecievedThroughputKB_s;
nodes(i).TransTP = data(i).Entry.NetworkTransmittedThroughputKB_s;
nodes(i).Time = 300;
end
nodes(i).Rel=1;


if nodes(i).Time < best_val.Time
best_val.Time = nodes(i).Time;
end

if nodes(i).CPUCap > best_val.CPU
best_val.CPU = nodes(i).CPUCap;
end
```

```
if nodes(i).MemCap > best_val.Mem
best_val.Mem = nodes(i).MemCap;
end

if nodes(i).ReadTP > best_val.ReadTP
best_val.ReadTP = nodes(i).ReadTP;
end

if nodes(i).WriteTP > best_val.WriteTP
best_val.WriteTP = nodes(i).WriteTP;
end

if nodes(i).RecTP > best_val.RecTP
best_val.RecTP = nodes(i).RecTP;
end

if nodes(i).TransTP > best_val.TransTP
best_val.TransTP = nodes(i).TransTP;
end

end

scores = CalculateScore(nodes,powPref,memPref,diskPref,netPref,costPref,...
timPref,relPref,best_val,timCostTrade);

for i = 1:n
nodes(i).Score = scores(i);
```

```
end

model.x = x;
model.y = y;
model.n = n;
model.nodes = nodes;

end
```

Read in File

```
function data = FileRead(file,tasks)
    warning('off')
    opts = detectImportOptions(file);
    opts.VariableNames = {'TimeStamp','CPU Cores','CPU Capacity',...
    'CPU Usage MHz', 'CPU Usage Perc', 'Memory Provisioned KB',...
    'Memory Usage KB', 'Disk Read Throughput KB/s', 'Disk Write' ...
    'Throughput KB/s','Network Recieved Throughput KB/s',...
    'Network Transmitted Throughput KB/s'};
    opts.DataLines = [2 tasks+3];
    opts.Delimiter = ';';


    data = readtable(file,opts);

    warning('on');

end
```

Plot Particle Movement

```
function PlotMovement(tour,model,length)
    warning('off');

    plot(model.x(tour), model.y(tour),'k-o',...
        'MarkerSize',10,...
        'MarkerFaceColor','y',...
        'MarkerIndices', length,...
        'LineWidth',1.5);

    xlabel('x');
    ylabel('y');

    axis equal;
    grid on;

alpha = 0.1;

    xmin  = min(model.x);
    xmax = max(model.x);
    dx = xmax -xmin;
    xmin = floor((xmin - alpha*dx)/10)*10;
    xmax = ceil((xmax + alpha*dx)/10)*10;
    xlim([xmin xmax]);

        ymin = min(model.y);
        ymax = max(model.y);
```

```
        dy = ymax - ymin;
        ymin = floor((ymin - alpha*dy)/10)*10;
        ymax = ceil((ymax + alpha*dy)/10)*10;
        ylim([ymin ymax]);

end
```

Appendix D

ANT COLONY EXPLORATION OF OPTIMIZATION ACROSS HYBRID FOG ENVIRONMENTS

FogWeaver: Task Allocation Optimization Strategy Across Hybrid Fog Environments

James Olmsted *School of Engineering and Technology*

University of Washington, Tacoma

jolmsted@alumni.mines.edu

Eyhab Al-Masri *School of Engineering and Technology*

University of Washington, Tacoma

ealmasri@uw.edu

IEEE International Conference on Knowledge Innovation and Invention

2020

FogWeaver: Task Allocation Optimization Strategy across Hybrid Fog Environments

James Olmsted*, Eyhab Al-Masri**

School of Engineering and Technology, University of Washington, Tacoma, WA, USA
*Corresponding Author: Email: *jmeo@uw.edu, **ealmasri@uw.edu

Abstract

As the complexity and requirements of IoT systems evolve, there comes a need for adaptable optimization algorithms that improve the process of task allocation, particularly across fog infrastructures. In this paper, we introduce FogWeaver, a strategy for optimizing task allocation within hybrid fog environments by integrating particle swarm optimization into the task allocation process. FogWeaver determines tasks that can efficiently run within fog environments, while reducing the latency associated with offloading tasks to cloud resources. Throughout this paper, we present experimental validation results, and analysis of the proposed optimization approach.

Keywords: internet of things, real-time data, fog computing, cloud computing, optimization, particle swarm

Introduction

As IoT systems evolve, cloud and fog infrastructures are becoming integral deployment strategies for IoT service provisioning. Since each have unique advantages and disadvantages, there is a need for a hybrid environment, where the cloud complements that of the fog environment. Specifically, a strategy is needed for placing and migrating computational tasks optimally while adapting to the dynamic needs and constraints of the system.

Developing hybrid IoT environments allows for the creation and adaptation of IoT systems to support a wider range of scenarios and processes. However, it is necessary to develop and improve up on existing techniques for optimizing the resource utilization across fog environments. To this extent, we propose FogWeaver, a multi-objective optimization method that optimizes an execution plan for tasks involving computational fog nodes that exist across both cloud and fog infrastructures. Through FogWeaver, it is possible to enhance computation offloading by identifying when a task can execute locally within a fog infrastructure or require migration to remote cloud servers.

To illustrate the need for a hybrid environment, consider a fog-based IoT hospital system which is composed of several fog gateways and localized medical IoT devices that collect patients' data in real-time for further processing. The rate at which this data is generated is likely to increase as the number of patients increases. As a result, dynamically allocating resources at runtime for faster processing of advanced operations such as image processing, artificial intelligence and machine learning becomes necessary. Healthcare or mission-critical IoT systems cannot afford time delays introduced by network traffic congestions when migrating tasks to remote servers on the cloud for fast processing.

In a typical cloud-based IoT system, data migrates from devices to remote servers on the cloud, which is often associated with high latency rates [1]. Because processing computationally-intensive operations traditionally depends on cloud resources, this process often yields to long execution times [1, 20]. That is, latency-sensitive IoT applications require efficient techniques for utilizing resources more efficiently for enhancing the processing of advanced operations at a local scale. Hence, processing computational operations locally in a distributed manner across fog infrastructures is desirable.

Given that not all operations can be executed locally at fog infrastructures due a wide variety of reasons most notably the lack of sufficient compute resources, the dependency on cloud resources is crucial. The decision of when to migrate tasks to the cloud involves a tradeoff in terms of reducing energy consumption while improving the accuracy of the output. As a result, some tasks can execute locally while other tasks can only be executed on cloud servers. Hence, the difficulty in determining when to execute a task locally requires efficient metrics of comparison between cloud- and fog-based task executions. To overcome this challenge, we introduce FogWeaver which considers fundamental task characteristics as decision variables for optimizing the task allocation process. This allows FogWeaver to be adapt dynamically to a large range of tasks running on constrained IoT environments.

Related Work

The characterization and optimization of both cloud and fog IoT environments have been explored extensively in literature. Several research studies have examined the optimization of resource allocation across both cloud and fog computing environments [2, 3, 4, 5, 6]. All of these research studies examined the required resource characteristics for optimizing performance. In [2, 3, 4], the authors focus their studies primarily on fog infrastructures, while in [3] the authors provide an optimization strategy based on minimizing latency.

The authors in [4] divide the resource allocation problem into two phases: (a) identify the correct number of nodes needed, and (b) assign tasks as needed to existing nodes. In [5, 6], the authors examine the resource allocation within a cloud environment where each study had a different primary focus for applying optimization. The authors in [5] build an optimization strategy that focuses on minimizing deadline violations, while in [6], the authors develop an energy-saving optimization strategy that focuses on reducing power consumption.

The optimization of workflow scheduling has typically been solved using multi-objective optimization algorithms [7, 8, 9, 10, 11, 12]. The majority of existing research studies on optimizing scheduling workflows apply a multi-objective particle swarm algorithm [8, 9, 10, 11]. Other studies employ methods such as: (a) ant colony optimization [7, 15, 16], (b) time- and space-optimized NSGA-II [12], (c) applying a set of heuristics to minimize execution times across a multi-cloud

environment [13] and (d) using a hybrid load push-pull algorithm to minimize execution time across network clusters [14].

While existing research studies provide solutions for optimizing the allocation of resources and improving the task execution process, their primary focus is on the operability of a single running environment. What is therefore needed is a mechanism for optimally allocating resources across multiple heterogeneous cloud and fog environments. We need to create a metric of comparison that supports task allocation across a hybrid environment to ensure that tasks are executed using an optimal number of resources while providing a balance between minimizing total execution time and clients' operating costs.

FogWeaver: A Fog-Based Optimization Strategy

Given that the availability of fog resources is limited [20], application tasks or workloads may need to be queued in a waiting queue until resources become available. In some cases, instead of workloads or tasks being queued, IoT applications offload these tasks to the cloud. However, the total execution time for migrating tasks to a remote environment (i.e. cloud) could be longer when compared to the overall duration it may take for tasks to wait for resources to become available on a local environment (i.e. fog). To this extent, we need a strategy that identifies whether a task can execute locally or remotely based on the availability of resources.

The goal of FogWeaver is to dynamically examine task requirements and given available resources, identify an optimal solution or a computing resource that can execute the task. Through FogWeaver, the task allocation process becomes more dynamic since it incorporates runtime factors (e.g. CPU, memory and storage) into the selection process. By considering these differentiating factors among resources within fog and cloud environments, FogWeaver enhances the task execution process while improving the agility of IoT applications.

The methodology for implementing FogWeaver comprises of three phases: (1) investigating the features of comparison for running workloads on cloud and fog environments, (2) collecting measurements for these features or attributes and (3) developing an optimization technique that can identify an optimal solution from a list of possible alternatives. An optimal solution represents an available computational resource that can execute the task across a hybrid fog environment.

In order to develop a truly hybrid computing environment, it is necessary to compare between the cloud and fog environments such that the optimization technique can consistently and accurately identify optimal resources for executing tasks. To this end, we use multiple criteria to compute the performance scores of each environment in terms of having available resources that can execute a particular task. By measuring these performance scores, it is then possible to develop metrics to bridge the gap in the characterization of each environment.

For testing FogWeaver, we used two real-world datasets that contain traces of actual workloads running on cloud environments. We used a dataset from Bitbrains (GWA-T-12) which contains tracing data consisting of 1,250 VMs connected to storage area network (SAN) devices [17]. We also used the 2017 tracing data from the Alibaba Cluster Trace Program [18]. This dataset contains tracing data for attributes such as CPU usage, memory provisioned, memory usage, and disk read/write throughput, among others. We use these attributes as decision variables for our optimization technique.

To identify available resources on a fog environment for executing a particular task, we employed the Bitbrains dataset [17] to develop a metric of comparison based on the weighted sum method. Then, we identify whether there exist fog resources that can execute a given task. We compute a score for each fog node using:

$$F_{wsE}(x) = \sum_{i=1}^{k} w_{iE} \frac{f_{iE}(x) - f_{iE}^{min}(x)}{f_{iE}^{max}(x) - f_{iE}^{min}(x)} \quad (1)$$

where $f_{wsE}(x)$ is the objective function for the fog environment, x is the fog node or device, k is the number of objective functions for the fog environment $f_{iE}(x)$ and relative importance using weight factors $w_{iE} \in [0, 1]$ such that $\sum_{i=1}^{k} w_{iE} = 1$ [19]. We employ seven fog-based objective functions (i.e. k = 7) for our multi-objective optimization technique. An example an objective function for CPU usage is defined as:

$$F_{1E}(x) = \frac{C(x)_{CPU}}{C_{CPU}^{max}} \times (1 - P(x)_{CPU}) \quad (2)$$

where $P(x)_{CPU}$ is the percentage of the CPU used on the fog device, $C(x)_{CPU}$ is the available CPU on the fog device and C_{CPU}^{max} represents the maximum available CPU among all fog devices

For considering cloud resources, we used Alibaba's tracing dataset [18] and employed the weighted sum method for determining optimal resources for task placement. The objective function for a cloud resource is defined as:

$$F_{wsC}(x) = \sum_{i=1}^{k} w_{iC} \frac{f_{iC}(x) - f_{iC}^{min}(x)}{f_{iC}^{max}(x) - f_{iC}^{min}(x)} \quad (3)$$

where $f_{wsC}(x)$ is the objective function for the cloud environment, k is the number of objective functions for the cloud environment $f_{iC}(x)$ and weights $w_{iC} \in [0, 1]$ such that $\sum_{i=1}^{k} w_{iC} = 1$ [19]. For our implementation, we set k = 6 and the cloud-based objective functions, other than $f_{4C}(x)$ which represents the cloud network response, follow those of the fog environment. To calculate $f_{4C}(x)$, the cloud network response, we use the following equation:

$$F_{4C}(x) = \frac{n(x)}{n_{max}} \quad (4)$$

where n(x) is the network score of a cloud resource. A cloud network score is defined as:

$$n(x) = w_{1C} \times \frac{C_{contCPU}(x) - C_{contCPU}^{min}}{C_{contCPU}^{max} - C_{contCPU}^{min}} + w_{2C} \times \frac{C_{contMem}(x) - C_{contMem}^{min}}{C_{contMem}^{max} - C_{contMem}^{min}} + w_{3C} \times \frac{C_{contDisk}(x) - C_{contDisk}^{min}}{C_{contDisk}^{max} - C_{contDisk}^{min}} \quad (5)$$

where $C_{contMem}(x)$ represents the available memory on a running cloud container and $C_{contDisk}(x)$ represents the available disk space on a cloud container.

Based on the computed scores, a threshold value can be devised based on end-users' preferences. The threshold value is determined by assigning a priority or weight, defined as w_{th},

using the following equation:

$$\lambda = F_{wsH}^{max} - \left((F_{wsH}^{max} - F_{wsH}^{min}) \times (1 - w_{th})\right) \quad (6)$$

where λ is the threshold score, F_{wsH}^{max} is the maximum score across both the fog and cloud nodes as determined by equations 1 and 3 respectively, and F_{wsH}^{min} is the minimum score. The optimization algorithm has been applied and tested using the same dataset from which our metric of comparison is developed.

The response of the algorithm is divided into layers based on the required task. The first level treats an IoT system as a fog environment with no cloud dependency and optimally places a task on an available fog device or node. To accomplish this, an IoT system is represented as a weighted directed graph *G*, composed of a number of vertices *V* representing nodes and weighted edges *E*. The weight of each edge is equal to the value determined by equations 1 and 3 for edge and cloud resources, respectively. We then use this information in our optimization component to determine the maximum scores across fog and cloud environments as defined in equations 1 and 3, respectively.

To compute the performance scores, we employ a hybridization of the ant colony and particle swarm optimization methods. The algorithm works by having *k* ants traversing through the graph by following a pheromone trail along the edges of the graph leading to the best available or optimal solution. We use equations 7, 8, 9 and 10, as introduced in [19], for computing the probability of the movement along an edge and the pheromone levels on that edge:

$$p_{xy}^{k} = \frac{\left(\tau_{xy}^{\alpha}\right)\left(\eta_{xy}^{\beta}\right)}{\sum_{z \in allowed_z}(\tau_{xz}^{\alpha})\left(\eta_{xz}^{\beta}\right)} \quad (7)$$

$$\tau_{xy} = (1 - \rho)\tau_{xy} + \sum_{k} \Delta\tau_{xy}^{(k)} \times \gamma_{xy}^{(k)} \quad (8)$$

where ρ is the evaporation rate. For our implementation, we use a constant $\rho = 0.05$. The pheromone change value $\Delta\tau_{xy}^{(k)}$ and the score improvement factor $\gamma_{xy}^{(k)}$ are given by [19]:

$$\Delta\tau_{xy}^{(k)} = \frac{\phi f_{best}}{f_{worst}} \quad (10)$$

$$\gamma_{xy}^{(k)} = \begin{cases} 1, & \text{if } V_{x,score} \leq V_{y,score} \\ 0, & \text{if } V_{x,score} > V_{y,score} \end{cases} \quad (11)$$

where $V_{x,score}$ is the score found using Equations 1 or 3, for the vertex that the ant moved from while $V_{y,score}$ is the score of the destination vertex

The ant colony optimization only executes if the initial fog device score is not equal to or greater than λ. If at any time during the execution of the algorithm a node with a score equal to or greater than λ is found, the task is assigned to that node. Once FogWeaver checks all available nodes within the fog environment, it begins examining all available resources on the cloud environment. Then, the fog- and cloud-based solutions are compared to determine an optimal solution for an available resource that can execute the task being examined.

Experiments and Results

Using our developed optimization model, each available resource that can satisfy the requirements of a given task is examined and a performance score is computed for each resource. These resources can be fog- or cloud-based compute resources. The computed scores are used for comparing the execution plan across both environments by applying an ant colony/particle swarm optimization method with varying weight values (i.e. w_{th}) leading to changes in the cutoff values for both environments.

As the value of w_{th} is lowered, more possible solutions can be achieved since the feasible region becomes larger. We evaluated our optimization model using the Bitbrains and Alibaba's tracing datasets. Figure 1 presents the accuracy for simulating a range of 0 to 1000 fog nodes (i.e. compute resources). To reduce the impact of any possible existing anomalies within the data, we executed 100 trials and computed the average score for 1000 test cases (i.e. fog nodes).

Fig. 1 Scatter Plot of FogWeaver Accuracy with Varying Fog Nodes and Threshold Weights

As can be seen in Figure 1, the accuracy remains above 82% indicating that the optimization model is able to identify an optimal solution (i.e. resource) that can execute a task. That is, the accuracy represents the rate at which our optimization model is able to determine a suitable compute resource across both fog and cloud environments. Since we are considering the cloud environment, the ability to find an optimal solution is increased while decreasing the cutoff threshold without having a significant reduction in terms of the accuracy.

To determine the efficiency of our proposed FogWeaver optimization approach, we executed several test cases with varying the number of fog nodes and the relative importance in terms of the execution time and cost. While executing a task on the cloud may achieve high accuracy in terms of the output results, the execution time and cost may increase significantly. In such cases, the execution time includes the response time taken for a cloud resource receiving a request from a fog device (or node), generating and sending a response back to the originating fog device (or node). Hence, when identifying suitable resources, there may exist multiple conflicting criteria such as reducing execution time and cost while increasing output accuracy.

Figure 1 shows the accuracy as a function of the number of nodes given various values of w_{th}. When w_{th}=1, the algorithm does not enforce a cutoff value and continues searching until it reaches the maximum number of iteration or finds an optimal solution. By varying w_{th}, the threshold score determined by

equation 6 changes. As w_{th} decreases, the score required to allocate a task to a specific resource (or node) also decreases. As a result, a drop in the accuracy and allocation time occurs as presented in Figures 1 and 2.

The minimum accuracy rate as shown in Figure 1 is 82% which is also reflected in Figure 2 having the allocation time dropping as a function of w_{th} when varying the number of fog nodes. To help illustrate this pattern, we present in Figure 2 the logarithmic scale for the y-axis (i.e. the allocation time). Although the relative decrease in the allocation time compared to that of the accuracy when reducing the weight is significant, our FogWeaver optimization model maintains an adequate level of accuracy (i.e. 82% or higher).

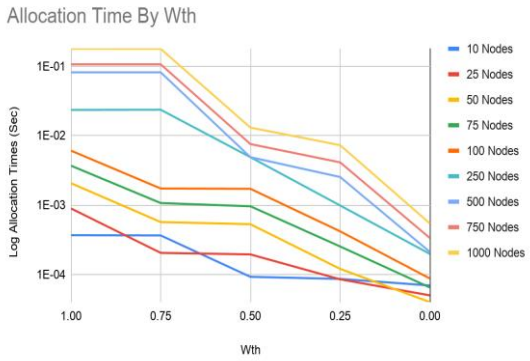

Fig. 2 Allocation Time for Varying Number of Fog Nodes based

Conclusion

As the number of IoT services increases across fog environments, the need for optimizing the resource allocation process while maintaining efficient task execution is indispensable. In this paper, we present FogWeaver, a method that can be incorporated within fog nodes for optimizing the selection of relevant resources that can execute computational tasks. Through FogWeaver, it is possible to reduce the offloading of application parts while enhancing the rate at which tasks execute within distributed fog environments. We evaluated our proposed approach and results show an accuracy rate of 82% in terms of finding a relevant or optimal solution for executing tasks across hybrid cloud and fog environments. For future work, we plan to include multi-criteria decision analysis methods such as TOPSIS for improving the task allocation decision making process.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things", First Edition of the MCC Workshop on Mobile Cloud Computing, pp. 13-16, 2012.

[2] P. Hoang-Nam and M. Tran, "Dynamic Resource Provisioning on Fog Landscapes," Security and Communication Networks, pp. 1-15, 2019.

[3] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Resource Provisioning for IoT Application Services in Smart Cities," 13th Inter'l Conference on Network and Service Management, pp. 1-9, 2017.

[4] N. Kherraf, et al., "Optimized Provisioning of Edge Computing Resources with Heterogeneous Workload in IoT Networks," IEEE Trans. on Network and Service Management, vol. 16(2), pp. 459-474, 2019.

[5] Y. Choi and Y. Lim, "Optimization Approach for Resource Allocation on Cloud Computing for IoT," International Journal of Distributed Sensor Networks, vol. 2016, pp. 1–6, 2016.

[6] M. Barcelo, et al., "IoT-Cloud Service Optimization in Next Generation Smart Environments," in IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 4077-4090, 2016.

[7] L. Zuo, L. Shu, S. Dong, C. Zhu and T. Hara, "A Multi-Objective Optimization Scheduling Method Based on the Ant Colony Algorithm in Cloud Computing," IEEE Access, vol. 3, pp. 2687-2699, 2015.

[8] E. S. Alkayal, N. Jennings and M. Abulkhair, "Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing," IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops), IEEE, pp. 17-24, 2016.

[9] L. Guo, G. Shao and S. Zhao, "Multi-Objective Task Assignment in Cloud Computing by Particle Swarm Optimization," 8th International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1-4, 2012.

[10] A. Verma and S. Kaushal, "A hybrid multi-objective Particle Swarm Optimization for Scientific Workflow Scheduling," Parallel Computing, vol. 62, pp. 1-19, 2017.

[11] J. Durillo, J. García-Nieto, A. Nebro, C. Coello, F. Luna and E. Alba, "Multi-Objective Particle Swarm Optimizers: An Experimental Comparison," in: Ehrgott M., Fonseca C.M., Gandibleux X., Hao JK., Sevaux M. (eds) "Evolutionary Multi-Criterion Optimization,". Springer Lecture Notes in Computer Science, vol. 5467, pp 495-509, 2009.

[12] Q. Liu, et al., "A speculative Approach to Spatial-Temporal Efficiency with Multi-Objective Optimization in a Heterogeneous Cloud Environment," Sec. and Comm. Networks, vol. 9, pp. 4002– 4012, 2016.

[13] R. Van den Bossche, et al., "Cost- Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds," 3rd Inter'l Conf. on Cloud Computing Technology and Science, pp. 320-327, 2011.

[14] M. Hu and B. Veeravalli, "Dynamic Scheduling of Hybrid Real-Time Tasks on Clusters," in IEEE Transactions on Computers, vol. 63, no. 12, pp. 2988-2997, 2014.

[15] F. Ferrandi et al., "Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 29, no. 6, pp. 911-924, 2010

[16] H. Alayed et al., "Enhancement of Ant Colony Optimization for QoS-Aware Web Service Selection," in IEEE Access, vol. 7, pp. 97041-97051, 2019.

[17] S. Shen, V. v. Beek and A. Iosup, "Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters," 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 465-474, 2015.

[18] H. Ding, et al., "Alibaba Cluster Trace Program," [Online]. Available: https://github.com/alibaba/clusterdata. [Accessed: 12-August-2020].

[19] R. Arora, "Optimization: Algorithms and Applications," CRC Press, ISBN: 9780429162527, 2015.

[20] J. Wang, Z. Lu, and E. Al-Masri, "Edgify: Resource Allocation Optimization for Edge Clouds Using Stable Matching," In Companion Proceedings of the Web Conference 2020 (WWW '20), pp. 128–130.