GINA CODY
SCHOOL OF ENGINEERING
AND COMPUTER SCIENCE

Resource Allocation for Multiple Workflows in Cloud-Fog Computing Systems

Jean Lucas de Souza Toniolli

A Thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Computer Science at
Concordia University
Montréal, Québec, Canada

December 2020

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Jean Lucas de Souza Toniolli**

Entitled: **Resource Allocation for Multiple Workflows in Cloud-Fog Computing Systems**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Yann-Gaël Guéhéneuc

_____ Examiner
Dr. Yann-Gaël Guéhéneuc

_____ Examiner
Dr. Marta Kersten-Oertel

_____ Thesis Supervisor
Dr. Brigitte Jaumard

Approved by _____
Dr. Hovhannes Harutyunyan, Graduate Program Director

_____ _____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Resource Allocation for Multiple Workflows in Cloud-Fog Computing Systems

Jean Lucas de Souza Toniolli

Constant innovations in the Internet of Things (IoT) in latest years have generated large amounts of data, putting pressure on the infrastructure of cloud computing. Fog computing has recently become a popular computing paradigm that can provide computing resources close to the end users and solve multiple issues with the current cloud-only systems. Fog computing helps to reduce transmission latency and monetary cost for cloud resources, while cloud computing helps to fulfill the increasing demands of large-scale compute-intensive offloading applications. Since its introduction, there has been a great number of studies on fog computing, in which devices that are free-of-charge and closer to the user can provide low-latency services to end devices. However, how to schedule workflow applications in the cloud-fog environment to seek the tradeoff between makespan and price is facing enormous challenges. To address such a problem, we present an adaptation of the Path-Clustering Heuristic to the cloud-fog environment for multiple workflows. Firstly, we define the models for workflow execution time and resource cost in fog computing. Afterwards, we describe the algorithms implemented. We validate our proposal by extensive simulation. Experimental results show that our scheduling adaptation achieves better performance while keeping similar costs compared to others.

Keywords: Fog computing; cloud computing; workflow scheduling; monetary cost; schedule length; heterogeneous systems; directed acyclic graph.

Acknowledgments

I would like to express my deepest gratitude to my parents and sisters, for their love, encouragement and unconditional support while I was walking my own path overseas. Without you, the realization of this dream would not be possible.

To my supervisor, Dr. Brigitte Jaumard, for all of her support, guidance, patience and the opportunities she provided me.

I thank all the professors who provided me not only rational knowledge, but also the manifestation of the character and affectivity of education in the professional training process, for so much that they dedicated themselves to me, not only because they taught me, but because they made me learn. The word master will never do justice to the dedicated professors to whom without naming they will have my eternal thanks.

To my friends, co-workers and brothers of friendship who were part of my education, specially Kia Babashahi and Antoine Hébert, for the incentive and great help they gave me. Certainly, such friendships will continue to be present in my life.

Finally, I would like to dedicate this work to the loving memory of my grandfather, Osmany Toniolli.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the last decade, we have witnessed an explosion in popularity of smart devices. This increasing number of devices connected to the Internet significantly impacts the global volume of generated traffic and ushering in a world of interconnected smart devices, thus driving the rise of the Internet of Things (IoT) [6]. The number of objects connected to the Internet already surpassed the world human population in 2010 [2] and is expected to reach 30 billion by 2020 [21]. Such devices may not only exchange data but may also store and process data, use sensors to collect data from the surrounding environment, and actively intervene on the environment via actuators. The growing and spreading number of IoT devices will unavoidably produce a huge amount of data, which has to be processed, stored and properly accessed by end users and/or client applications. The IoT devices are in the center of the action but lack the computing and storage to perform analytics.

Meanwhile, gaining more and more popularity in recent years, cloud computing [38, 41], widely known as the next generation's computing infrastructure with virtual unlimited resource and service provisioning, provides many features to complement IoT devices (e.g., huge processing power and great storage provision) by expanding their power and extending the capability of computing. It is considered as an extension of computing systems such as parallel, distributed and grid computing. With the help of the Internet, it provides

fast and safe data storage as well as computing power. Hence, cloud computing offers a significant complement to IoT. The inherent limitations of IoT devices can be helped by offloading compute-intensive, resource-consuming tasks up to a set of powerful processors in the cloud, leaving only simple tasks to the capacity bounded smart devices.

However, when IoT meets cloud, many challenges arise. The centralized nature of cloud computing can lead to a considerable topological distance between the cloud resources and the end users devices. Moreover, the costs associated with cloud storage and bandwidth usage are significant and as a result, there is a growing industry trying to address this need. High-speed data processing, analytics and shorter response times are also becoming more necessary than ever. Addressing these issues is difficult given the current centralized, cloud-based model that powers most IoT connected systems. Although the cloud servers have the ability to process the data, they do not correspond to the optimal solution, due to either cost or time inefficiencies.

Fog Computing [10] is a promising solution to alleviate these obstacles. The idea of fog computing is to extend the cloud to be closer to the things that produce and act on IoT data, based on a more distributed framework [5]. The fog computing layer can serve as the perfect middle ground where there is enough computing power to provide analytics, enough computing power to ingest and process the data from the IoT devices, and being close enough to the user, thus compensating both the shortage of cloud computing and its latency [20].

Fog computing was proposed in 2012 by Cisco to overcome limitations of integration between cloud data centres and the IoT. Any local device that has enough computing, storage and networking capabilities to run advanced services can be part of the fog environment such as switches, routers, etc. This fog environment offers much lower latency than cloud computing, located far from the end users, by keeping the data and computations close to end users, thus reducing the communication delay over the Internet and minimizing the

bandwidth burden, deciding wisely what data needs to be sent to the cloud. Nevertheless, the flexibility and scalability of cloud computing can help fog computing to cope with the growing demands for large-scale computation-intensive business applications when the processing capacity of fog computing is insufficient. Therefore, fog computing is not aimed to replace cloud computing, but to add to it in a new computing paradigm, cloud–fog computing [23].

Often, IoT data corresponds to real-time jobs, made up of a set of interdependent tasks, linked with precedence constraints. These jobs are called workflow applications, each can be represented by a Directed Acyclic Graph (DAG) [14] where each DAG has many tasks with communication constraints or time dependencies [39]. All distributed processing nodes (cloud or fog) are managed by a resource broker, which is a resource management component and scheduler for the workflows submitted from fog-side users. In fog computing, as in cloud computing, task scheduling is a systematic approach to allocate available resources for incoming requests from clients to process applications [1]. Therefore, there is an important decision-making process to be managed, which refers to appropriately distributing the IoT data to resources in both the fog and cloud layers, taking into account the computational and communication aspects of each application [23]. Additionally, the problem of scheduling DAG-based workflow applications to obtain optimal results is a non-deterministic NP-Complete problem [37].

1.1 Background

The task scheduling problem is the process of scheduling a set of tasks on distributed nodes from devices with low computational capacity and power, such as IoT devices, in order to satisfy one or more objectives [8]. Thus, fog computing technology moves computations to the edge of the network near the IoT devices. In this thesis, the term “task” is used to indicate a request which is an atomic unit of processing. These tasks can be independent

Requirements/Features	Cloud Computing	Fog Computing
Latency	High	Low
Network Access Type	Mostly WAN	LAN (WLAN)
Server Location	Anywhere within the network	At the edge
Mobility Support	Low	High
Distribution	Centralized	Distributed
User Device	Computers, mobile devices (limited)	Mobile-smart-wearable devices
Management	Service Provider	Local Business
Number of Servers	High	Low

Table 1: Differences Between Cloud and Fog Computing (extracted from Baktir *et al.* [7])

(bag of tasks) or dependent (workflow) on each other [22]. When requests are workflows, the inputs of many tasks depend on the outputs of other tasks, and the tasks cannot start until all of the predecessor tasks are completed. Considering this precedence constraint, we must take into account inter-task data dependencies when scheduling tasks on system resources Two tasks v_i and v_j are dependent if execution of task t_j should be started after completion of task v_i. On the other hand, if they do not have a dependency relationship with each other, tasks are considered independent and can be offloaded separately to resources in the network to be executed simultaneously and in parallel. In this kind of problem, tasks have parameters which include resource requirements, size of the task and its precedents. Each task is gonna have the following two attributes: the ready time and the earliest start time. The ready time refers to when all necessary input data to process a task have arrived at the target processing node, thus making it ready to be executed. The earliest finish time denotes the earlier time that a processing node can execute a task. It could be as soon as the task is ready or on the next gap time of the processor.

There are many forms to evaluate task scheduling approaches to estimate their effectiveness. In this regard, some metrics that can be considered in scheduling approaches in distributed systems such as cloud and fog environment, are as follows

- Execution time: time that a task is running on a processing node and utilizes its resources.
- Makespan: total time take to process a set of tasks for its complete execution.
- Efficiency: proportion of execution time to total makespan.
- Cost: total payment for usage of resources
- Trade-off: a trade off between cost and makespan

1.2 Contributions

Most of the initial scheduling algorithms were designed for scheduling a single workflow only and focused on reducing application time of executing without taking into account the monetary costs of using the cloud [33]. However, a task schedule, which can minimize the completion time of the workflow, but corresponds to a large amount of monetary cost, is not a well optimized solution for fog providers. Thus, in this thesis, we present an adaptation to schedule multiple workflows on the cloud-fog environment that can achieve a good trade off between the workflow execution times and the usage cost of cloud resources by adapting the Path Clustering Algorithm (PCH) [9] to the cloud-fog environment. The main objective of our heuristic is to reduce the communication bottleneck on the schedule by clustering tasks to the same processor. In order to evaluate our proposal we also implemented a data set generator that can output multiple graph topology. We compared our proposal to other methods and the performed simulations demonstrates that our adaptation outperforms the

adapted benchmarks and provides the best overall performance for the users. The results of this thesis have been published [13].

1.3 Outline

The rest of the thesis is organized as follows. The next chapter provides an overview of the related literature. Chapter 3 presents the cloud-fog system model and formulates the task scheduling problem. In Chapter 4, we describe the adaptation of the PCH algorithm and the benchmarks for multiple workflows, while Chapter 5 gives a description of the experimental setup, performance metrics and analyzes the simulation results. The last Chapter summarizes and concludes the thesis.

Chapter 2

Related Work

A lot of research has been done [32, 36, 12, 4] regarding the task or workflow scheduling in distributed environments, especially task scheduling under cloud computing platform. The primary goal of task scheduling is to schedule tasks on processors and minimize the makespan of a schedule, which has been shown to be a NP-complete problem [37].

Sih and Lee [32] introduce Dynamic Level Scheduling Algorithm (DLS). At each step, the algorithm selects the task-processor pair that maximizes the value of the difference between the static level of a task and its earliest start time on a processor, characterized as the dynamic level (DL). The computation cost of a task is the median value of the computation cost of the task on the processors. In this algorithm, upward rank calculation does not consider the communication costs. The scheduling algorithm Longest Dynamic Critical Path (LDCP) was proposed by Daoud *et al.* [12], to support scheduling in heterogeneous computing systems. The LDCP is a list-based scheduling algorithm, which effectively selects the tasks for scheduling in heterogeneous computing systems. This algorithm builds for each processor a DAG, called DAGP, which consists of the initial DAG with the computation costs of the processors. The accurate and better selection of tasks enables the LDCP to provide a high quality result when scheduling heterogeneous systems. However, it is shown that due to its greed nature the algorithm does not scale very well.

One of the most popular list based heuristic to schedule workflow applications is Heterogeneous Earliest Finish Time (HEFT) [36]. It orders the workflow tasks decreasingly based on their DAG upward rank, which can be seen as a reversal depth in the reversal DAG. After computing the priorities of all tasks based on the upward rank value, each task is iteratively assigned to a suitable resource that minimizes its earliest finish time. The Predict Earliest Finish Time (PEFT) [4] algorithm takes advantages of lookahead features by introducing an Optimistic Cost Table (OCT), in which each element of the OCT specifies the maximum value of the shortest path of successors of the current task to the exit task while taking into account the selected processor for the current task. The OCT value is used to decide the rank of each task. PEFT does not allocate the current task on the processor which reduces the earliest finish time, it will allocate the task to the processor that minimizes the Optimistic EFT value which is the earliest finish time value of that tasks plus its OCT value. This way it expects that in next steps it will reach a smaller finish time for the tasks. Yet, these algorithms for heterogeneous systems only consider the minimization of the workflow schedule length and do not discuss the monetary aspect of using computing resources, i.e., the cost of offloading large scale applications to a cloud environment.

The extensive study of the cost and performance-aware large-scale workflow scheduling models has been driven by the emergence of cloud computing [26, 15, 40, 19, 11, 30, 3, 17]. Van den Bossche *et al.* [15] introduce a cost-efficient approach in a hybrid cloud model, which is built from a private cloud and multiple public clouds. The approach determines the most appropriate infrastructure to execute the workflow. Selection also depends on the possibility of meeting the deadline of each workflow as well as the cost savings. Li *et al.* [19] presented the Cost-Conscious Scheduling Heuristic (CCSH), an extension of the HEFT algorithm, which schedules workflows considering both cost and schedule length.

The cloud cost is represented as the cost-conscious factor and is used as a weight to calculate the earliest finish time of each task. Chopra and Singh [11] presented a level and cost-based scheduling algorithm in order to schedule workflow applications on hybrid clouds, in which a deadline is assigned to each task in the workflow application and migrates the tasks to public cloud when their deadline is not met in the private cloud. A new scheduling algorithm based on a fitness adaptive algorithm-job spanning time adaptive genetic algorithm has been developed by Omara and Arafa in [26]. The algorithm enhances the overall performance of the cloud environment while considering monetary costs. However, it has a high complexity cost making it difficulty to apply to large scale workflows.

The budget conscious scheduling algorithm ScaleStar was proposed by Zeng *et al.* in [40]. ScaleStar is based on an objective function called Comparative Advantage (CA) that tries to satisfy the strict budget constraint. Their algorithm achieve good balance between cost savings and schedule length, however, the high complexity of CA prevents the algorithm from being applicable to large scale workflows. Amoon *et al.* [3] presented a three phase algorithm to schedule workflows on a cloud computing system in which they consider a trade off between the monetary and time costs. The three-phase process successfully assigns the cloud server for each task. In [17], a scheduling algorithm is proposed that depends on duplicating and grouping tasks. The scheduling is accomplished by first converting the input DAG into in-tree and then grouping the tasks. Afterwards, it merges the groups of tasks and the groups are allocated for execution in the cloud. However, these works are mainly processed in the cloud, which is different from our framework where each individual task of the workflow can be offloaded to both cloud and fog nodes.

Fog computing is a popular research topic, and many studies have focused on its importance nowadays. It was presented by Bonomi *et al.* [10], where they emphasize that fog computing is a highly virtualized platform which usually, but not only, provides computation, storage, and network services between end devices and cloud computing. This

highlights the importance of the cloud-fog interplay and the role of fog computing in the context of IoT. However, fog computing resource management and task scheduling are still at the initial phase in this paradigm although more and more attention has been paid to it recently [34].

The authors in [18] initially design a region architecture based on fog to provide neighboring computing resources. They explore efficient scheduling algorithms to assign tasks between regions and remote clouds. The trade off between power consumption and transmission delay in different phases of iteration in a cloud-fog environment has been investigated by Deng *et al.* [16]. Based on these two factors, their strategy tries to determine the optimal allocation of workload between fog and cloud layers. Through simulation experiments and analytical solutions, this work demonstrates that the fog can significantly complement the cloud with considerably less communication latency. However, the proposed approach can not be implemented to workflow application as there is no consideration of workflow constraints between workload assignments.

In [25], Nan *et al.* propose an online algorithm, called Unit-slot Optimization, for scheduling applications in a three-tiered system. It is an adaptive decision-making algorithm for distributing the incoming data to the corresponding tiers, which can provide cost-effective processing while ensuring average response time. The proposed approach dynamically adjusts the trade off between response time and average monetary cost, based on the technique of Lyapunov optimization. Still, inter-task dependencies are not considered making it not suitable for scheduling workflow applications. Pham *et al.* [28] proposed a Cost-Makespan aware Workflow (CMaS) scheduling for achieving the balance between performance of application execution and monetary cost for using cloud resources in a cloud-fog environment. Even though this approach is both fog and cloud-aware and is suitable for workflow applications, however, it only considers a single workflow at a time for scheduling.

To the best of our understanding, in the context of cloud–fog, multiple workflows scheduling have not been studied extensively so far. The fog and cloud-aware approach in this paper is suitable for scheduling multiple workflows to achieve a good trade off between time and the cost for the use of cloud and resources utilizing possible schedule gaps. Table 2 summarizes task scheduling references.

Reference	Environment	Application Type	Min. makespan	Min. cost	Trade-off
Topcuoglu *et al.* [35]	Heterogeneous	Workflow	Yes	No	No
Arabnejad and Barbosa [4]	Heterogeneous	Workflow	Yes	No	No
Van den Bossche *et al.* [15]	Cloud	Bag of Tasks	Yes	Yes	No
Li *et al.* [19]	Cloud	Workflow	Yes	Yes	Yes
Chopra and Singh [11]	Cloud	Workflow	Yes	Yes	No
Hoang and Dang [18]	Fog and Cloud	Bag of Tasks	Yes	Yes	No
Den *et al.* [16]	Fog and Cloud	Bag of Tasks	Yes	Yes	Yes
Nan *et al.* [25]	Fog and Cloud	Bag of Tasks	Yes	Yes	Yes
Pham *et al.* [29]	Fog and Cloud	Workflow	Yes	Yes	Yes

Table 2: Existing task scheduling approaches

Chapter 3

System Model and Problem Formulation

Workflow applications scheduling in cloud-fog environment is formulated as the problem of assigning heterogeneous computing resources to the tasks of workflow application in order to minimize the makespan and cost of workflow applications scheduling. We assume here that each workflow is represented by a DAG. In DAGs, nodes act as tasks to be executed and edges act as precedence constraints between tasks or the communication edges between tasks [24]. Independent or unrelated tasks can be assigned to different processors to be executed simultaneously. Note, however, that if dependent tasks are assigned to the same processor, this saves the communication time between them.

We consider a three-layered cloud-fog computing system model as shown in Figure 1. Such an environment is a distributed computing platform based on collaboration between cloud and fog computing for executing large-scale offloaded workflow applications. The system model consists of three parts, i.e., the IoT layer, Fog layer and Cloud layer. The applications are uploaded from the user devices to a thick client, also known as a broker, in the fog functioning as a centralized management node which results in a cost effective service with a shorter response time. The broker acts as a task scheduler and resource management component. The functions of the broker are to receive all applications from users via Wi-Fi, manage the resources available on the cloud and fog nodes and create

accordingly the most appropriate program for a concurrent arrival of multiple applications, and decide which component of each application should be completed on which resource.

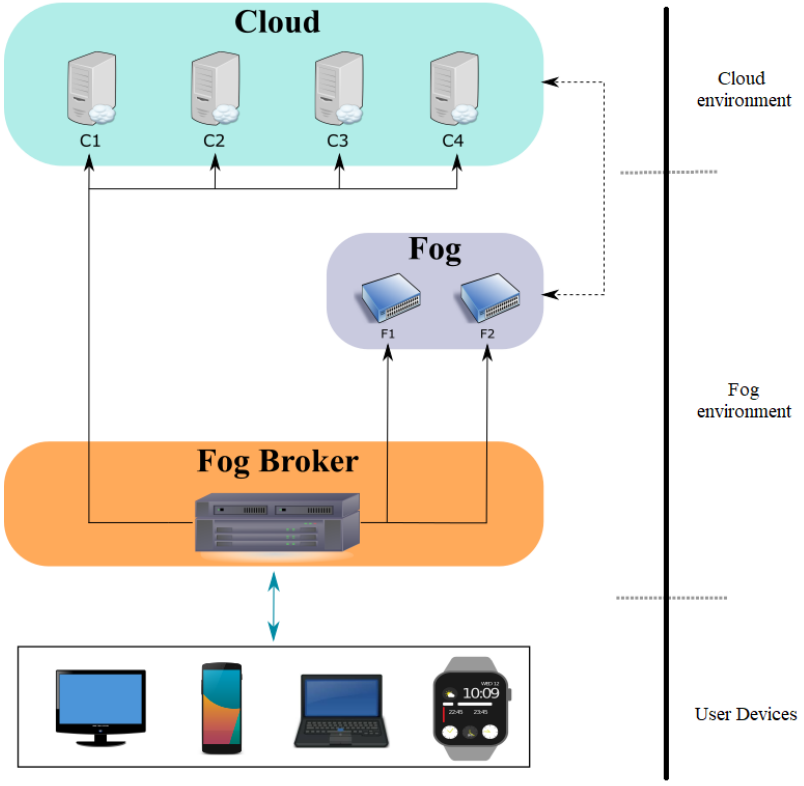

Figure 1: System model

Let X be a set of devices in the IoT layer, connected to the fog broker. We assume that each fog node is deployed at a fog network near the user devices and the computing power of a fog node is much stronger than any user device. The user devices are connected to an access point (AP) via a shared radio communication channel, e.g., a wireless network. The AP is connected to the broker via a high speed link to which the user devices submit their requests and the transmission delay between AP and broker can be neglected. We assume that fog nodes and cloud nodes have enough CPU resources so that each task can be executed independently by a dedicated CPU. Additionally, we enforce the realistic duplex single-port communication model, where at any point in time, each processor can, in

parallel, execute a task, send one data, and receive another data.

The input of the multiple application task scheduling involves a list of DAG-based applications, i.e., tasks graphs and, a heterogeneous processor graph with fog and cloud nodes. The output is a schedule representing the assignment of a processor to each task. A schedule consists of each task in each application being assigned with start and finish time to a node in the processor graph. In the next subsections, we formulate the application's model.

3.1 Task Graph

A workflow is represented by a DAG ant is denoted as $G = (V, E)$, where V is the set of data processing components, i.e., the tasks, and E is the set of directed edges between the tasks standing for the data dependencies. Each node in $V = \{v_1, v_2, ..., v_n\}$ represents a task and each edge $e_{ij} = (v_i, v_j) \in E$ represents the precedence relationship between tasks v_i and v_j, which means task v_j cannot start until the precedent task v_i completes. Each task $v_i \in V$ can be defined as a 4-tuple $v_i = (v_i^{\text{ID}}, w_i, \text{STO}_i, \text{MEM}_i)$ where v_i^{ID} denotes its identification in the application by numerical values, w_i represents a positive workload denoting its computation resource amount (e.g., CPU cycles) when it is executed, STO_i is the task storage requirement, and MEM_i is the amout of memory required by the task. Besides, it also contains a set of all preceding sub-tasks $\text{PRED}(v_i)$ and a set of all successive sub-tasks $\text{SUCC}(v_i)$. Each edge $e_{ij} \in E$ has a non-negative weight C_{ij} which denotes the amount of data migration between two tasks with precedence relationships. We assume that a task v_i without any predecessor, $\text{PRED}(v_i) = 0$, is an entry task v_{ENTRY}, and a task that does not have any successors, $\text{SUCC}(v_i) = 0$, is an end task v_{EXIT}.

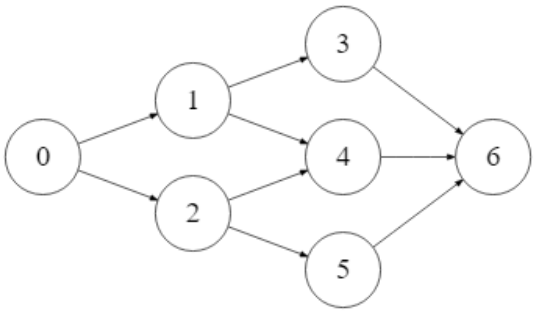

Figure 2: A task graph example of an workflow application

3.2 Processor Graph

The computing resources can be represented with a processor graph, $PG = (N, D)$, which denotes the topology of the cloud-fog network. It consists of a set of fully connected heterogeneous processors that can communicate with each other through LAN and WAN [16]. Each node can be either a cloud server or a fog node, and each edge $d_{ij} \in D$ denotes a link between processor P_i and P_j. Any resource in the processor . Each processor P_i has a processing rate p_i and a bandwidth bw_i on the link connecting it to other processors. N_{fog} and N_{cloud} denote the set of fog nodes and cloud servers respectively. In the next section, we describe our proposed approaches.

3.3 Time Model

The workflow execution time contains two parts: the computation cost and the communication time. The average execution cost of a task v_i and average communication time between two tasks v_i and v_j, respectively, are defined as

$$\overline{w}(v_i) = \frac{w_i}{(\sum\limits_{P_n \in N} p_n)/n} \tag{1}$$

$$\overline{\mathrm{C}}(e_{ij}) = \frac{c_{ij}}{(\sum\limits_{P_n \in N} bw_n)/n} \tag{2}$$

where n is the number of processors in the cloud-fog environment.

A task can only be executed by a processor when all of its preceding tasks are completed. DTT(v_i) is defined as the time when all its immediate predecessor tasks have finished, i.e., the finish time of the last preceding task of v_i, which means, it is the time when all input data of v_i is ready to be transferred to the target processor. Thus, it is called data transfer time of task v_i and defined by

$$\mathrm{DTT}(v_i) = \max_{v_j \in \mathrm{PRED}(v_i), P_m \in N} \left[t_f(v_j, P_m) \right] \tag{3}$$

where $t_f(v_j, P_m)$ is defined as the finish time of task v_j on processor P_m. As we can see, the entry task will have a value of 0 for DTT.

The communication cost of transferring data from preceding tasks of v_i processed at processor P_n to processor P_m to execute v_i assuming task v_i would be processed at processor P_m is defined as

$$\mathrm{C}(e_i^{mn}) = \begin{cases} (\sum\limits_{\substack{v_j \in \mathrm{EXEC}(P_m) \\ \cup \\ \mathrm{PRED}(v_i)}} \mathrm{C}_{ji}) \times \left(\dfrac{1}{bw_m} + \dfrac{1}{bw_n} \right) & \text{if } n \neq m \\ 0 & \text{if } n = m \end{cases} \tag{4}$$

where the set of executed tasks at node P_m is $\mathrm{EXEC}(P_m)$.

We define the ready time of task v_i on processor P_m as $t_{dr}(v_i, P_m)$ and all necessary input data of v_i have arrived at the target processing node P_m is defined by

$$t_{dr}(v_i, P_n) = \mathrm{DTT}(v_i) + \max_{P_m \in N} \mathrm{C}(e_i^{mn}). \tag{5}$$

In order to schedule a task on a processor, we need to be able to accommodate tasks on Idle Time Slots (ITS) as shown in Figure 3.

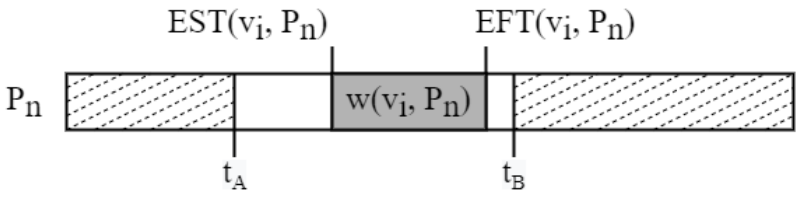

Figure 3: An example of an idle time slot

For that purpose, we define the EST and EFT attributes. EST(v_i, P_n) and EFT(v_i, P_n) denote the earliest execution start time and the earliest execution finish time of task v_i on processor P_n respectively. By searching the earliest ITS on a processor P_n which is able to execute task v_i, we can define the EST attribute. Also, in order to compute the EFT of a task v_i, all immediate predecessors tasks of v_i must have been scheduled. For an idle time slot $[t_A, t_B]$ on P_n to host task v_i, there must be no task being executed within this time slot on P_n. A free task v_i can be scheduled on P_n within $[t_A, t_B]$ if

$$max\{t_A, t_{dr}(v_i, P_n)\} + w(v_i, P_n) \leq t_B \tag{6}$$

For the found interval $[t_A, t_B]$, the EST(v_i, P_m) and EFT(v_i, P_m) are computed as follows

$$\text{EST}(v_i, P_n) = \max\{\text{TIME}(P_n), t_{dr}(v_i, P_n)\} \tag{7}$$

$$\text{EFT}(v_i, P_n) = \text{EST}(v_i, P_n) + w(v_i, P_n), \tag{8}$$

where $w(v_i, P_n)$ is the execution time for task v_i on processor P_n and TIME(P_n) is the earliest ITS on processor P_n capable of allocating task v_i. After a task v_i is scheduled on a processor P_n, the earliest finish time of v_i will be the EFT(v_i, P_n) value. After all tasks in a graph are scheduled, the schedule length or makespan will be the actual finish time of the

exit task, defined as

$$\text{makespan} = \text{AFT}(v_{\text{EXIT}}). \tag{9}$$

The objective function of this task scheduling problem is to determine the assignment of tasks of a given application to processors such that the metric of the algorithm is either maximized or minimized.

3.4 Cost model

The monetary cost for executing a task v_i on processor P_n is denoted by $\text{COST}(v_i, P_n)$. The only cost involved for fog processors is when they need to receive data from cloud processors. In contrast, the monetary cost for cloud processors includes processing cost, storage cost, memory cost and communication cost for outgoing data from other cloud processors to the target processor P_n to execute task v_i. Each cost is calculated as follows. Cost of processing a task on a cloud node is expressed as

$$C_{\text{PROC}}^{(v_i,P_n)} = \text{COST}_1 * w(v_i, P_n), \tag{10}$$

where COST_1 is the processing cost per time unit of workflow execution on processor P_n. Suppose that the amount of money per time unit for transferring outgoing data from processor P_n is COST_2, then the communication cost is calculated as follows:

$$C_{\text{COM}}^{(v_i,P_n)} = \text{COST}_2 \times \sum_{v_j \in \text{EXEC}(P_n)}^{v_j \in \text{PRED}(v_i)} \text{C}_{ji}. \tag{11}$$

Let COST_3 be the storage cost per data unit and let st_i be the storage size of task v_i on processor P_n. Then the storage cost of task v_i on processor P_n is calculated as

$$C_{st}^{(v_i,P_n)} = \text{COST}_3 \times st_i. \tag{12}$$

Further, we compute the cost of using the memory of processor P_n for task V_i as follows:

$$C_{\text{MEM}}^{(v_i,P_n)} = \text{COST}_4 \times s_{\text{MEM}} \tag{13}$$

where COST_4 is the memory cost per data unit and s_{MEM} is the size of the memory used. Therefore, the total cost for executing task v_i on a specific node P_n is defined by:

$$\text{COST}(v_i, P_n) = \begin{cases} C_{\text{PROC}}^{(v_i,P_n)} + \sum\limits_{P_m \in N_{\text{CLOUD}}} C_{\text{COM}}^{(v_i,P_n)} + C_{st}^{(v_i,P_n)} + C_{\text{MEM}}^{(v_i,P_n)} & \text{if } P_n \in N_{\text{CLOUD}} \\ \sum\limits_{P_m \in N_{\text{CLOUD}}} C_{\text{COM}}^{(v_i,P_n)} & \text{if } P_n \in N_{\text{FOG}}. \end{cases} \tag{14}$$

Chapter 4

Multiple Workflows Scheduling Algorithms

The starting point in the scheduling of multiple workflows is to decide if the DAGs will be scheduled sequentially, if they will be combined in a single DAG and scheduled after that or if they will be scheduled independently in turns. To schedule more than one DAG we can adopt three strategies, in general:

- Schedule the DAGs independently, one after another.
- Schedule the DAGs independently in turns, interleaving parts of each DAG being scheduled.
- Merge the DAGs into a single one, and schedule this resulting DAG.

To schedule multiple workflows we assume that, at a given time, we have tasks of T workflows to be scheduled. The strategy used for the algorithms is a list-based interleaved approach. This approach was used given better overall results compared to the other methods. In the interleaved approach the algorithm schedules pieces of each DAG in turns,

interleaving their tasks in the schedule of the available resources. The algorithms prioritizes and selects the tasks to be scheduled, and also selects the resources where these tasks will run. With this approach all workflows have similar makespans, it is scalable, maintains the good schedules and fairness with variations in the number of results. In this work, we used an adapted version of the Path Clustering Heuristic [9], which is explained in Section 4.2.4. An example of scheduling is shown in Figure 4.

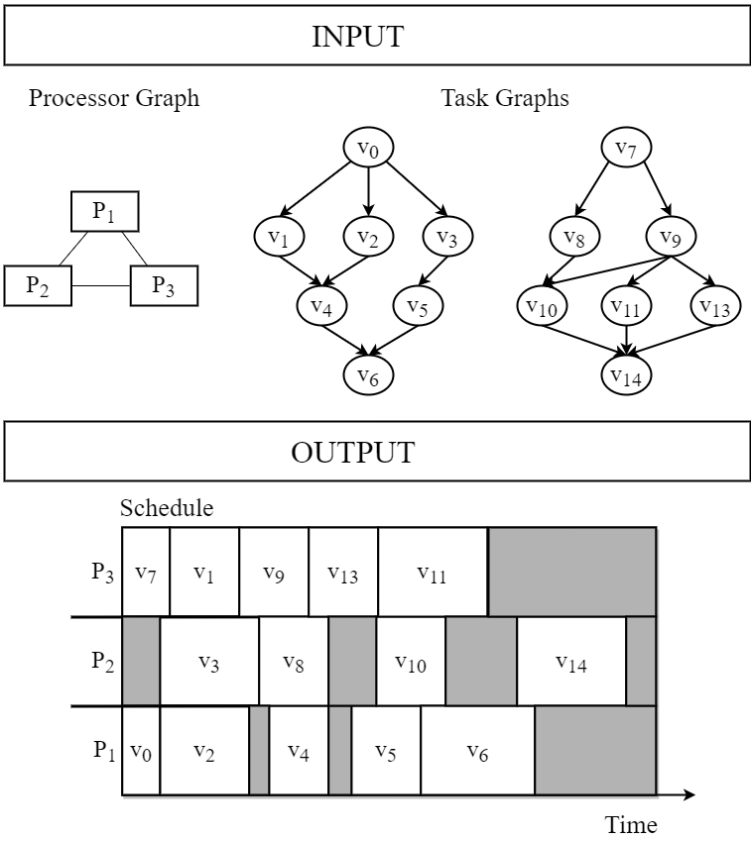

Figure 4: Example of multiple applications task scheduling

Given a list TG with T applications summing t tasks in total where every application is a task graph $G = (V, E)$ and a processor graph with network topology $PG = (N, D)$, the algorithms will choose the most appropriate schedule to execute the tasks. Their major objective is to either minimize the schedule length, the cost for the use of cloud resources or to achieve a good trade off between both. The list-based approach, which is a priority

based technique consisting in two phases is used for the algorithms. The first phase is done the same way for all algorithms where each node of the task graphs receives a priority level and is ordered into a list. The second phase is different for each algorithm, thus it will be explained separately for each algorithm.

4.1 Task prioritizing phase

Before scheduling, the tasks of each application are labeled with scheduling priorities and are ordered afterwards in a list based on upward ranking [36]. The scheduling priorities are calculated for each application separately. Basically, this priority is the length of the critical path from task v_i to the exit task. The scheduling priority is defined by the following recursively function

$$\text{PRIO}(v_i) = \begin{cases} \overline{w}(v_i) + \max\limits_{v_j \in \text{SUCC}(v_i)} \left[\overline{c}(e_{ij}) + \text{PRIO}(v_j)\right] & \text{if } v_i \neq v_{\text{EXIT}} \\ \overline{w}(v_i) & \text{if } v_i = v_{\text{EXIT}} \end{cases} \tag{15}$$

Lastly, the t tasks are sorted by a non-increasing order of their priority level. The topological order of each application and its precedence constraints are preserved by this.

4.2 Processor selection phase

After the tasks are sorted in a non-increasing order the algorithm will iteratively assign processor nodes in the cloud-fog network to execute them. All the algorithms in this work use the same way to get the scheduling list which is described in Algorithm 1. In the next sections, we will briefly describe how each algorithm selects the processor for each task.

Algorithm 1 Get Scheduling List

Input: List TG of Task Graphs
Output: Scheduling list L
1: **procedure** GETSCHEDULINGLIST
2: **for all** $G_i \in TG$ **do**
3: Compute the priority level PRIO(v_i) of each task $v_i \in G_i.V$ by traversing the graph upward, starting from v_{exit}
4: $L \leftarrow$ Sort all tasks into list L by non-increasing order of priority levels
5: **return** L

4.2.1 Cost-Makespan aware Scheduling

In CMaS, the scheduling list L is iterated and each task will be scheduled following the maximization of the utility function. The utility function $U(v_i, P_n)$ computes the tradeoff between the cost and EFT and is defined as follows

$$U(v_i, P_n) = \frac{\min\limits_{P_k \in N} \left[\text{COST}(v_i, P_k)\right]}{\text{COST}(v_i, P_n)} \times \frac{\min\limits_{P_k \in N} \left[\text{EFT}(v_i, P_k)\right]}{\text{EFT}(v_i, P_n)} \tag{16}$$

Task v_i is scheduled to processor P_n if it provides the maximal value of the tradeoff $U(v_i, P_n)$. This method is shown in Algorithm 2.

Algorithm 2 Cost-Makespan aware Scheduling

Input: List TG of Task Graphs and Processor Graph
Output: Task Schedule
1: **procedure** MULTI-CMAS
2: $L \leftarrow$ getSchedulingList(TG)
3: **for all** $v_i \in L$ **do**
4: **for all** $P_n \in N$ **do**
5: Compute $EST(v_i, P_n)$, $EFT(v_i, P_n)$ and $cost(v_i, P_n)$
6: Compute the utility function $U(v_i, P_n)$
7: Schedule task v_i to processor P_n that maximizes the U of task v_i

4.2.2 Heterogeneous Earliest Finish Time

The HEFT algorithm is considered as a well-known list scheduling algorithm that can produce very low schedule length in comparison to other scheduling algorithms. The processor selection phase is done by the minimization of the completion time of each task. The algorithm iterates over the scheduling list and schedules each task v_i to a processor P_n if it minimizes the EFT value $EFT(v_i, P_n)$. This method is shown in Algorithm 3

Algorithm 3 Heterogeneous Earliest Finish Time

Input: List TG of Task Graphs and Processor Graph
Output: Task Schedule
1: **procedure** MULTI-HEFT
2: $L \leftarrow$ getSchedulingList(TG)
3: **for all** $v_i \in L$ **do**
4: **for all** $P_n \in N$ **do**
5: Compute $EST(v_i, P_n)$ and $EFT(v_i, P_n)$
6: Schedule task v_i to processor P_n that minimizes the EFT of task v_i

4.2.3 Cost-Conscious Scheduling Heuristic

In this method, the effect of the monetary cost for the use of cloud services is represented by the input cost-conscious factor which is used as a weight to calculate the earliest finish time (EFT) of each task. The algorithm assigns a processor P_n to a task v_i if it minimizes the Cost-EFT (CEFT) value which is defined as follows

$$\text{CEFT}(v_i, P_n) = \text{EST}(v_i, P_n) + w(v_i, P_n) + \delta_{in} \times w(v_i, P_n), \tag{17}$$

$$\delta_{in} \in [0, \frac{\text{COST}(v_i, P_n)}{\max\limits_{P_m \in N} \left[\text{COST}(v_i, P_m)\right]}], \tag{18}$$

where δ_{in} is the input cost-conscious factor. CCSH is shown in Algorithm 4

Algorithm 4 Cost-Conscious Scheduling Heuristic

Input: List TG of Task Graphs and Processor Graph
Output: Task Schedule
1: **procedure** MULTI-CCSH
2: $L \leftarrow$ getSchedulingList(TG)
3: **for all** $v_i \in L$ **do**
4: **for all** $P_n \in N$ **do**
5: Compute EST(v_i, P_n), COST(v_i, P_n) and CEFT(v_i, P_n)
6: Schedule task v_i to processor P_n that minimizes the CEFT of task v_i

4.2.4 Adapted Path-Clustering Heuristic

The Path Clustering Heuristic is a DAG scheduling heuristic that utilizes the clustering technique to create task clusters and the list planning technique to select processors to accommodate the tasks. The PCH groups paths of the DAG, creating clusters of tasks that reduces the communication between them. Before each processor selection phase, PCH performs a clustering phase which is described as follows.

After the task prioritizing phase, the tasks are added to a queue cls_{list}. The algorithm uses the priority of each task to select the first task to be added to the first cluster. The first task v_i selected to compose a cluster cls_k is the unscheduled node with the highest priority. it is added to cls_k, then the algorithm starts a depth-first search (DFS) on the DAG starting with v_i which is the unscheduled task with the highest priority, selecting $v_s \in succ(v_i)$ with the highest $pri(v_s)$ and adding it to cls_k. The DFS stops when it reaches a task v_s which has a non scheduled predecessor. With this, clusters always have only tasks with all predecessors already scheduled or to be scheduled along with them. Figure 5 shows an example of clusters created by PCH and the resulting schedule.

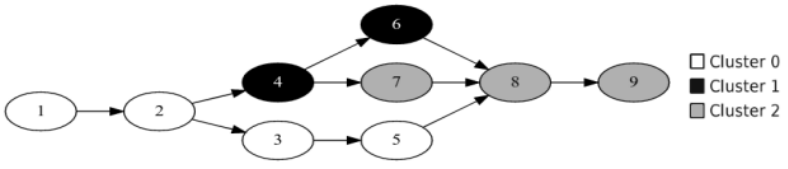

Figure 5: Example of clustering workflows using the adapted PCH

For each cluster created, the algorithm selects a resource to schedule it. The processor selection step is performed after each clustering step. The PCH looks for the processor which minimizes the EFT of the last task from the cluster. The criterion for selecting the processor for a cluster is therefore to minimize the makespan of the clusters, which is described as

$$U_{cls_k} = \min_{P_n \in N} \max_{v_i \in cls_k} \text{EFT}(v_i, P_n) \tag{19}$$

The algorithm repeats the clustering and processor selection steps until all nodes in cls are scheduled. In the original version the nodes selected in the DFS are based on the priority and EST of that task, but given intensive testing we decided to only use the priority of the tasks given the better results. This method is presented in Algorithm 5.

Algorithm 5 Adapted Path-Clustering Heuristic

Input: List TG of Task Graphs and Processor Graph
Output: Task Schedule
1: **procedure** MULTI-PCH
2: $L \leftarrow$ getSchedulingList(TG)
3: **while** there are unscheduled tasks in L **do**
4: $cluster \leftarrow$ cluster to be scheduled
5: **for all** $v_i \in cluster$ **do**
6: **for all** $P_n \in N$ **do**
7: Compute EST(v_i, P_m) and EFT(v_i, P_m)
8: BEST_PROCESSOR $\leftarrow$ Compute the processor which minimizes the maximum EFT of tasks in the cluster
9: **for all** $v_i \in$ CLUSTER **do**
10: Schedule the task v_i to BEST_PROCESSOR

Chapter 5

Evaluation

In this section, we present experiments via numerical simulations to evaluate the adapted PCH and validate the trade-off between cost and makespan length for the scheduling of multiple workflow applications simultaneously in a cloud-fog computing environment. First we provide details of our simulation studies, then we investigate the performance of our proposed approach by evaluating cost, schedule length and trade-off between them under different settings with the other algorithms described in Chapter 4.

5.1 Experimental Settings

All of the scheduling methods were implemented by us using Java with JDK 1.8 based on the running environment of Intel Xeon E3-1271 v3, quad-core 3.6 GHz CPU, 32GB RAM and Fedora 29 OS.

Next, some details about experimental settings are described as shown in Table 3. We specify the performance parameters regarding cloud servers and fog servers where various servers have different processing capabilities. In our simulation, we use MIPS (Million instructions per second) to represent the processing capacity of processors. In the simulation, we consider a scenario with 14 fog nodes and 24 cloud servers in the fog environment.

For tasks in workflow, the computation workload of each task ranges from 100 to 15,000 MI (Million instructions) and the I/O data of a task has a size from 100 to 1,000 MB. The experiment contains several simultaneous workflows and each workflow has from 10 to 80 tasks. The pricing for fog and cloud resources is show in Table 4, which is the same using in [29].

Table 3: Characteristics of the cloud-fog environment

	Number of servers	Processing rate (MIPS)	Bandwidth (Mbps)
Cloud servers	12	250	{10; 100; 512; 1,024}
	12	1,500	{10; 100; 512; 1,024}
Fog servers	7	100	1,024
	7	500	1,024

5.2 Generation of Task Graphs

A random task graph generator was implemented to generate constructed workflows to represent application's DAGs with various configurations which depend on input parameters given below. We first execute the random task graph generator to create the application DAGs, then we follow by executing the scheduling algorithm to produce the task schedules, and, finally, we evaluate and validate our algorithm based on the generated task schedules. The generator algorithm is based upon a layered approach first proposed by Tobita and Kasahara [35]. They designed it specifically for validating scheduling heuristics and is based in the concept they call layers, i.e., the dependencies between tasks are one way meaning that tasks cannot impose further dependencies on their antecedents. Other authors as [35, 31, 27] have proposed algorithms for the synthetic generation of task graphs as well. Based on all of the above, we developed a random graph generator with characteristics as

explained next. The parameters that define the DAG generation are as follows

- n: number of computation nodes in the DAG (i.e., workflow tasks);
- Shape parameter of the graph α: we assume that the height (depth) of the DAG is randomly generate from a uniform distribution with a mean value equal to $\frac{\sqrt{n}}{\alpha}$. Whereas, number of tasks of each layer (width of each layer) is generated randomly from a uniform distribution whose mean value equal to $\sqrt{n} \times \alpha$. A dense graph (a shorter graph with high parallelism) can be generated with $\alpha >> 1.0$, whereas a smaller value induces a thinner DAG with low task parallelism $\alpha << 1.0$
- Communication-to-computation ratio CCR: ratio of the sum of the edge weights to the sum of the node weights in a DAG. It represents the relation between the average communication cost between tasks in the graph over the average computation cost of all tasks. An application can be considered as computation-intesive if the value of CCRs is low, whereas a high value of CCR indicates that the application is communication-intensive

To evaluate further the performance of our approach, for each experiment, we generate a data set of task graphs where values of the parameters are given below.

- Number of Workflows = $\{10, 20, 30, 40, 50\}$
- Number of Tasks = $\{10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80\}$
- α = $\{0.5, 1.0, 2.0, 1.5, 2.0\}$
- CCR = $\{0.1, 0.5, 1, 2, 5\}$

5.3 Performance Metrics

The aim of scheduling algorithms in a cloud-fog environment is to maximize the tradeoff between the makespan and the cost of task execution. The makespan represents the total time elapsed between the start time of the first executed task to the completion time of the last executed task. We considered the start time of entry tasks as zero and makespan can be calculated as the finish time of the last exit task. The performance metric used to evaluate the adapted PCH in comparison to the other algorithms is described below.

Table 4: Resource pricing

Parameter	Value
Processing cost per time unit	[0.1, 0.5]
Communication cost per data unit	[0.3, 0.34, 0.5, 0.7]
Unit cost of memory used	[0.01, 0.1]
Unit cost of storage used	[0.05, 0.2]

5.3.1 Cost Makespan Tradeoff

The comparison criteria called Cost-makespan Tradeoff (CMT) [28] was proposed to evaluate which algorithm can achieve better tradeoff between the makespan and the cost of task execution at each configuration of the simulations. The method is defined as follows:

$$\text{CMT}(a_i) = \frac{\min\limits_{a_k \in AL} \left[\text{COST}(a_k)\right]}{\text{COST}(a_i)} \times \frac{\min\limits_{a_k \in AL} \left[\text{makespan}(a_k)\right]}{\text{makespan}(a_i)}. \tag{20}$$

5.4 Numerical Results and Analysis

Here, we compare the four scheduling strategies. To test the influence of different number of tasks on experimental results, each workflow is randomly generated by DAG generator,

where the number of tasks in a DAG varies from 10 to 80. The combinations of the DAG generator can produce 20 different random graphs for each workflow size. For each DAG, 10 different random graphs were generated with the same structure but with different edge and node weights. For example, when the number of tasks of a workflow is 50, the ordinate value is the average makespan of 200 different experiments with 10, 20, 30, and 50 workflows, respectively.

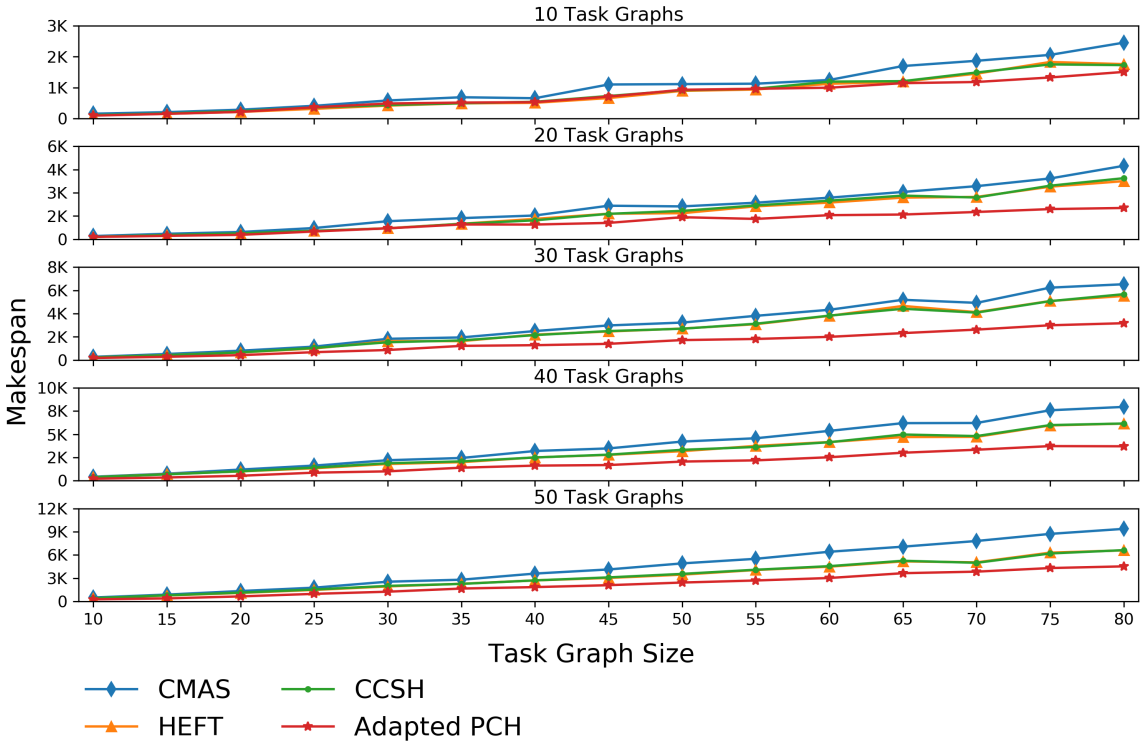

Figure 6: Comparison of the execution time

The simulation results reveal that our adaptation of PCH outperforms the other algorithms both in schedule length and cost. Figure 6 shows that in terms of schedule length, CMaS gets the worst case, the adapted PCH algorithm obtains the best result overall while CCSH and HEFT are in the middle. The adapted PCH is overall 28.28% better than HEFT,

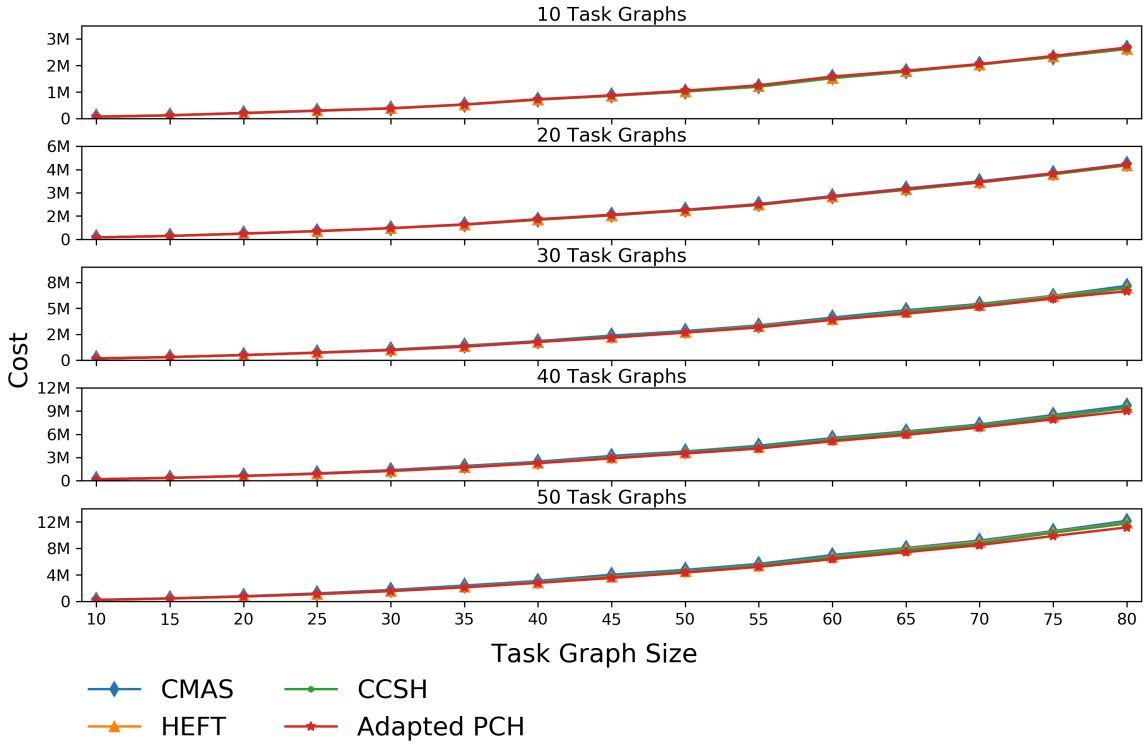

Figure 7: Comparison of the execution cost

29.56% better than CCSH and 43.60% better than CMaS. However, regarding the monetary cost for cloud resources, the adapted PCH keeps only a minimal margin below the other algorithms but still costs less overall as we can see in Figure 7. This means that the adapted PCH has a slight economic advantage while still having a much larger advantage over effectiveness.

In order to demonstrate more clearly that the adapted PCH can achieve a better tradeoff between the makespan and the cost of task execution than other methods, we will use the CMT comparison criteria described in Section 5.3. The maximum CMT value of an algorithm is 1, which is reachable if the schedule length and the monetary cost of that algorithm are the best (i.e., smallest) compared with the others.

Otherwise, the level of CMT an algorithm can achieve will demonstrate how good it is

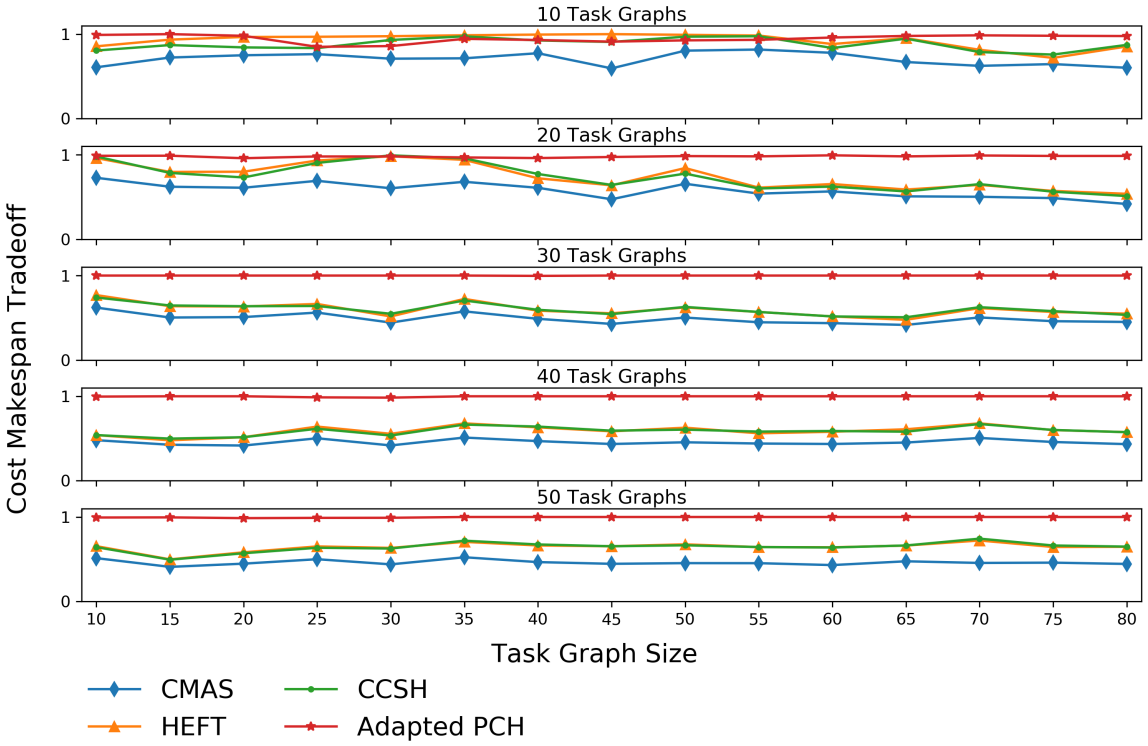

Figure 8: Cost-makespan tradeoff comparison

in terms of tradeoff level on monetary cost and schedule length. Figure 8 shows the comparison between the algorithms on the CMT metric. We can see that the adapted PCH is stable and achieve the highest CMT value compared with the others in most of the cases. The average CMT value of the adapted PCH on all task graph sizes and number of task graphs is better than HEFT by 28.86%, CCSH by 29.80%, and CMaS by 44.86%. As the number of tasks increases, we can observe that the performance of the adapted PCH increases in comparison to the other algorithms. The reason behind these results is as follows. The main objective of the adapted PCH is to minimize communication time by scheduling a cluster to the same processor. Even though it only takes into account the makespan of application execution to schedule the tasks, it ends up reducing communication costs by scheduling the cluster to the same processor and in so it achieves a better tradeoff than the

other algorithms.

Chapter 6

Conclusion and Future work

6.1 Conclusion

Workflow scheduling in fog computing is an important yet open issue. An increasing number of smart devices are emerging and joining the Internet, thus giving rise to a huge amount of data. The combined fog-cloud architecture is a promising model that if well exploited can provide efficient data processing of various applications or services, especially those which are compute-intensive. Those data have to be processed in either Fog or Cloud tier as fast as possible with a reasonable cost. In this thesis, we studied the problem of scheduling multiple application in the cloud-fog environment. We adapted the PCH algorithm for the cloud-fog environment aiming to maintain the tradeoff between cost and schedule length. We also implemented three other algorithms in this context of multiple applications scheduling. Simulation results demonstrated that the proposed adaptation can achieve the best overall performance in cost and time compared with the other strategies in the cloud-fog environment.

6.2 Future Work

In the future, we will further consider additional constraints such as fog provider's budget and deadline constraint of a workflow execution. We will also explore other popular multi-workflow scheduling optimization algorithms in the context of cloud and fog computing. We also aim to take into account other constraints such as RAM and energy consumption because these are factors that impact how the resources would be allocated or where tasks would be offloaded. Last but not least, we are interested in we evaluate the performance of the algorithms with respect to real-world applications.

Bibliography

[1] S. Agarwal, S. Yadav, and A. Yadav. An efficient architecture and algorithm for resource provisioning in fog computing. *International Journal of Information Engineering and Electronic Business*, 8(1):48, 2016.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015.

[3] M. Amoon, N. El-Bahnasawy, and M. ElKazaz. An efficient cost-based algorithm for scheduling workflow tasks in cloud computing systems. *Neural Computing and Applications*, 31(5):1–11, 2018.

[4] H. Arabnejad and J. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, 2014.

[5] H. Atlam, R. Walters, and G. Wills. Fog computing and the internet of things: a review. *Big Data and Cognitive Computing*, 2(2):10, 2018.

[6] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[7] A. C. Baktir, A. Ozgovde, and C. Ersoy. How can edge computing benefit from software-defined networking: A survey, use cases, and future directions. *IEEE Communications Surveys & Tutorials*, 19(4):2359–2391, 2017.

[8] A. Bhattacharya and P. De. A survey of adaptation techniques in computation offloading. *Journal of Network and Computer Applications*, 78:97–115, 2017.

[9] L. Bittencourt and E. Madeira. Fulfilling task dependence gaps for workflow scheduling on grids. In *2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System*, pages 468–475, Shanghai, China, 2007. IEEE.

[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, Helsinki, Finland, 2012. ACM.

[11] N. Chopra and S. Singh. Deadline and cost based workflow scheduling in hybrid cloud. In *2013 international conference on advances in computing, communications and informatics (ICACCI)*, pages 840–846, Mysore, India, 2013. IEEE.

[12] M. Daoud and N. Kharma. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *Journal of Parallel and distributed computing*, 68(4):399–409, 2008.

[13] J. de Souza Toniolli and B. Jaumard. Resource allocation for multiple workflows in cloud-fog computing systems. In *IEEE/ACM International Conference on Utility and Cloud Computing Companion*, pages 77–84, 2019.

[14] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[15] R. V. den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. In *IEEE International Conference on Cloud Computing technology and Ccience*, pages 320–327, Athens, Greece, 2011. IEEE.

[16] R. Deng, R. Lu, C. Lai, T. Luan, and H. Liang. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet of Things Journal*, 3(6):1171–1181, 2016.

[17] X. Geng, Y. Mao, M. Xiong, and Y. Liu. An improved task scheduling algorithm for scientific workflow in cloud computing environment. *Cluster Computing*, 21:1–10, 2018.

[18] D. Hoang and T. Dang. Fbrc: Optimization of task scheduling in fog-based region and cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 1109–1114, Sydney, NSW, Australia, 2017. IEEE.

[19] J. Li, S. Su, X. Cheng, Q. Huang, and Z. Zhang. Cost-conscious scheduling for large graph processing in the cloud. In *IEEE international conference on high performance computing and communications*, pages 808–813, Banff, AB, Canada, 2011. IEEE.

[20] Y. Lin and H. Shen. Cloudfog: leveraging fog to extend cloud gaming for thin-client mmog with high quality of service. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):431–445, 2016.

[21] S. Lucero et al. IoT platforms: enabling the internet of things. 2016.

[22] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[23] X. Masip-Bruin, E. Marín-Tordera, A. Alonso, and J. Garcia. Fog-to-cloud computing (f2c): The key technology enabler for dependable e-health services deployment. In *2016 Mediterranean ad hoc networking workshop (Med-Hoc-Net)*, pages 1–5, Vilanova i la Geltru, Spain, 2016. IEEE.

[24] S. Mustafa, B. Nazir, A. Hayat, S. M. A, et al. Resource management in cloud computing: Taxonomy, prospects, and challenges. *Computers & Electrical Engineering*, 47:186–203, 2015.

[25] Y. Nan, W. Li, W. Bao, F. Delicato, P. Pires, and A. Zomaya. Cost-effective processing for delay-sensitive applications in cloud of things systems. In *2016 IEEE 15th international symposium on network computing and applications (NCA)*, pages 162–169, Cambridge, MA, USA, 2016. IEEE.

[26] F. A. Omara and M. M. Arafa. Genetic algorithms for task scheduling problem. In *Foundations of Computational Intelligence Volume 3*, pages 479–507. Springer, Orlando, FL, USA, 2009.

[27] G.-L. Park. Performance evaluation of a list scheduling algorithm in distributed memory multiprocessor systems. *Future Generation Computer Systems*, 20(2):249 – 256, 2004. Modeling and simulation in supercomputing and telecommunications.

[28] X.-Q. Pham and E.-N. Huh. Towards task scheduling in a cloud-fog computing system. In *2016 18th Asia-Pacific network operations and management symposium (APNOMS)*, pages 1–4, Kanazawa, Japan, 2016. IEEE.

[29] X.-Q. Pham, N. Man, N. Tri, N. Thai, and E.-N. Huh. A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. *International Journal of Distributed Sensor Networks*, 13:1 – 16, 2017.

[30] P. Phuoc Hung and E.-N. Huh. An adaptive procedure for task scheduling optimization in mobile cloud computing. *Mathematical Problems in Engineering*, 2015:13, 2015.

[31] S. Shivle, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco. Mapping of subtasks with multiple versions in a heterogeneous ad hoc grid environment. In *Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, pages 380–387, July 2004.

[32] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE transactions on Parallel and Distributed systems*, 4(2):175–187, 1993.

[33] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 39(4-5):177–188, 2013.

[34] M. Taneja and A. Davy. Resource aware placement of iot application modules in fog-cloud computing paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1222–1228, Lisbon, Portugal, 2017. IEEE.

[35] T. Tobita and H. Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002.

[36] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[37] J. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[38] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[39] R. Xu, Y. Wang, H. Luo, F. Wang, Y. Xie, X. Liu, and Y. Yang. A sufficient and necessary temporal violation handling point selection strategy in cloud workflow. *Future Generation Computer Systems*, 86:464–479, 2018.

[40] L. Zeng, B. Veeravalli, and X. Li. Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In *IEEE International Conference on Advanced Information Networking and Applications*, pages 534–541, Fukuoka, Japan, 2012. IEEE.

[41] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.