WESTERN SYDNEY
UNIVERSITY

Resource Provisioning and Scheduling Algorithms for Hybrid Workflows in Edge Cloud Computing

by

Raed Alsurdeh

Supervisor: Assoc. Prof. Bahman Javadi
Associate Supervisors: Dr. Rodrigo N. Calheiros and Dr. Kenan M. Matawie

A thesis submitted in total fulfillment for the degree of Doctor of Philosophy

in the

School of Computer, Data and Mathematical Sciences

Western Sydney University

December 2020

Dedication

This Thesis is dedicated to my wife for her absolute love, trust, support and patience

to my father and my mother for their love and encouragement

to my daughters

Acknowledgements

All the praises and thanks be to Allah (SWT) with whose, guidance and blessing have I been able to accomplish this thesis. The completion of this thesis would not have been possible without the support and encouragement of several special people. I would like to take this opportunity to show my gratitude to those who have assisted me in a myriad of ways. First of all, I would to express my endless grateful to my research principle supervisor A/Prof. Bahman Javadi for his remarkable support along my degree journey. Thanks for his extreme academic guidance, brilliant motivation and outstanding support in all critical situations I experienced. I would to thank also my co-supervisor Dr. Rodrigo Calheiros for his remarkable support and brilliant ideas that steered me to the right directions. Also, I would to thank my co-supervisor Dr. Kenan for his exceptional support and guidance in solving mysterious mathematical problems in my research.

Thank you Sana Al-husband, the wife and the friend, for all support you provided, and thank you for the confident you are giving me to pass the hard times. Thank you my parents, you were the great motivation to finish this degree. Thank you my lovely daughters (Lyan, Talya and Julia) for the time you gave me. Thank you my brothers, my sisters, my Parent-in-law and my brothers-in-law for supporting me and my family.

I would like to thank and appreciate all my friends for their outstanding support. Special thank for Mohammad ALKHALAILEH, Maria and Rekha. Thank you Nabil and Guang for the technical support. Thank you to all staff in the library and GRS for all help and support I received. Thank you to Dr Campbell Aitken, who provided professional editing services in accordance with the Institute of Professional Editors' *Guidelines for editing research theses*.

I would like to express my sincere gratitude to Western Sydney University. WSU has assisted me in gaining new research skills and knowledge of IT over the last five years. I am also grateful for Australia for giving the opportunity to have this experience.

Raed Alsurdeh July 2020

Declaration of Authorship

I, Raed Alsurdeh, declare that this thesis titled, 'Resource Provisioning and Scheduling Algorithms for Hybrid Workflows in Edge Cloud Computing' and the work presented in it are my own. I confirm that:

I confirm that the work presented in this thesis is, to the best of my knowledge and belief, original except as acknowledged in the text. I hereby declare that I have not submitted this material, either in full or in part, for a degree at this or any other institution.

Signed: _____

Date: _____

Contents

List of Figures

List of Tables

Abbreviations

BoT	**B**ag **o**f **T**ask
C-GDHW	**C**onstraint-based **G**radient **D**escent search for **H**ybrid **W**orkflows
CC-HWPS	**C**luster-based and **C**ooperative **H**ybrid **W**orkflows **P**rovisioning and **S**cheduling
C-HWPS	**C**luster-based **H**ybrid **W**orkflows **P**rovisioning and **S**cheduling **P**rovisioning and **S**cheduling
CP	**C**ritical **P**ath
DAG	**D**irected **A**cyclic **G**raph
ES	**E**valuation **S**trategies
FDHEFT	**F**uzzy **D**ominance sort-based **H**terogeneous **E**arliest-**F**inish-**T**ime
FIFO	**F**irst-**I**n-**F**irst-**O**ut
GA	**G**enetic **A**lgorithm
GDS	**G**radient **D**escent **S**earch
HEFT	**H**eterogeneous **E**arliest **F**inish-**T**ime
HWRPO	**H**ybrid-**W**orkflow **R**esource **P**rovisioning **O**ptimizer
IaaS	**I**nfrastructure **a**s **a** **S**ervice
IoT	**I**nternet **o**f **T**hings
MIPS	**M**illion **I**nstructions **P**er **S**econd
MOHEFT	**M**ulti-**O**bjective **H** **E**arliest-**F**inish-**T**ime
PaaS	**P**latform **a**s **a** **S**ervice
PCP	**P**artial **C**ritical **P**ath
PSO	**P**article **S**warm **O**ptimization
QoS	**Q**uality **o**f **S**ervice
SaaS	**S**oftware **a**s **a** **S**ervice

SLA	**S**ervice **L**evel **A**greement
VM	**V**irtual **M**achine

Abstract

In recent years, Internet of Things (IoT) technology has been involved in a wide range of application domains to provide real-time monitoring, tracking and analysis services. The worldwide number of IoT-connected devices is projected to increase to 43 billion by 2023, and IoT technologies are expected to engaged in 25% of business sector. Latency-sensitive applications in scope of intelligent video surveillance, smart home, autonomous vehicle, augmented reality, are all emergent research directions in industry and academia. These applications are required connecting large number of sensing devices to attain the desired level of service quality for decision accuracy in a sensitive timely manner. Moreover, continuous data stream imposes processing large amounts of data, which adds a huge overhead on computing and network resources. Thus, latency-sensitive and resource-intensive applications introduce new challenges for current computing models, i.e, batch and stream. In this thesis, we refer to the integrated application model of stream and batch applications as a hybrid workflow model.

The main challenge of the hybrid model is achieving the quality of service (QoS) requirements of the two computation systems. This thesis provides a systemic and detailed modeling for hybrid workflows which describes the internal structure of each application type for purposes of resource estimation, model systems tuning, and cost modeling. For optimizing the execution of hybrid workflows, this thesis proposes algorithms, techniques and frameworks to serve resource provisioning and task scheduling on various computing systems including cloud, edge cloud and cooperative edge cloud. An edge cloud is a hybrid cloud architecture which extends the capability of cloud system closer to end user to deliver low-latency and bandwidth-efficient services. The research work outcomes presented in this thesis demonstrated the novelty of hybrid workflow scheduling problem through multi-direction experimental investigation on the contribution of concepts such as stream featuring, workflow scalability, resource utilization, edge collaboration and QoS-aware optimization. For resource estimation, an evolutionary technique was applied to search the space for an optimal hybrid workflow configuration setup

with stream rate, aggregation window and throughput parameters. The technique showed considerable limitations to handle large scale workflows. Thus, a linear optimization technique with gradient descent search was proposed to solve the high time complexity of the previous technique. For hybrid workflow scheduling, a group-based technique was applied on various computing systems to reduce the execution cost and time. Experimental results show a significant credibility of co-operative edge cloud systems in resolving the issues of hybrid workflow scheduling.

Overall, experimental results provided in this thesis demonstrated strong evidences on the responsibility of proposing different understanding and vision on the applications of integrating stream and batch applications, and how edge computing and other emergent technologies like 5G networks and IoT will contribute on more sophisticated and intelligent solutions in many life disciplines for more safe, secure, healthy, smart and sustainable society.

Chapter 1

Introduction

The Internet of Things (IoT) is a growing technology paradigm; it refers to a large set of objects (machines, devices, etc.) that can connect and share data without requiring human or computer intervention [3]. The IoT brings myriad new forms of business-oriented, user-specific and human-centric applications and its increasing adoption in many application domains generates a new need for rationalized utilization of computing resources to support computations. According to Cisco's market report [4], there will be 12.3 billion mobile-connected devices by 2022, operating a wide variety of IoT applications, ranging from intelligent video surveillance, smart retail to the Internet-of-Vehicles.

Even before the advent of the IoT, massive data generation is beginning to cause bottlenecks in traditional computing systems. For example, in business analytics, huge amounts of data is used to discover and resolve business issues, such as predicting changes in customer behaviours and market conditions, to increase customer satisfaction, and to provide value-added services to customers [5]. Another example is in healthcare, in which numerous organizations are building applications and analytical tools to help patients, physicians and other healthcare stakeholders to measure and maintain quality and find opportunities [6]. The increasingly popularity of IoT usage motivates researchers to introduce reliable and convenient application models to overcome the challenges of processing real-time data generated from IoT devices, as well as the huge amount of data stored during

processing cycles. In 2001, META Group (now Gartner) analyst Doug Laney was the first to define the enormous growth of data as a three-dimension model, or big data model, citing the "3Vs" , i.e., volume, velocity and variety [7]:

"Big data is the representation of information assets described by data size (Volume), data generation speed (Velocity) and Variety to require dedicated technologies and analytical methods for its transformation into value" [8].

High volumes of data require powerful computing systems to ensure meaningful portions are extracted from the raw data, while high data generation speed requires efficient data extraction to process data streams on the fly, because it is time-consuming and costly to store data first and process it afterwards [9]. Velocity refers to the rate of data generation and data transfer, and also measures the required time to process the incoming data streams [10]. The higher the data velocity, the larger the size of the data sets to be processed. For many applications, the velocity of data generation is even more critical than the volume [11]. For example, real-time data can help researchers and businesses make valuable decisions that provide strategic competitive advantages if they handle incoming streams efficiently and within short data acquisition time. An effective system design involves dealing with a high rate of data stream to support business process agility. High rates of data generation are make it more challenging for classical computing models and reduce their efficiency in performing processing, analysis and computation operations. New approaches are needed to satisfy the computation requirements of big data , necessitating re-examination and investigation of current and new analytic models, mathematical prediction models, and algorithms.

Hu et al. [12] defined the phases of big data value chain architecture. The data generation phase focuses on how and what types of data is generated. Data can be obtained from sensors, video, click streams and other digital sources. The data acquisition phase denotes the process of capturing and pre-processing of the data. Collected raw data requires a high transmission mechanism, and sometimes pre-processing to extract meaningful information. This process is known as data cleansing. The data collection depends on the type of available resources, as well

as the objective of data analytics. Data can be collected using various methods. The third phase is data storage, which depends on the processing paradigm.

Big data processing has two main data paradigms, batch and stream processing. Batch processing involves storing the upcoming data prior to processing, while stream processing is related to performing operations on data streams in a real-time or near real-time manner. The value added during stream processing depends on data freshness [13], and stream processing is gaining more attention than batch processing, which remains the common [12]. Batch processing determines huge disk-based storage, whereas stream processing is highly dependent on memory-based hardware architecture.

Data analysis is a domain-specific process; selecting the analytics tools relies on the type of data, which can be structured, semi-structured or unstructured. Data analytics applications are commonly manged and executed with workflow technology. Generally, a workflow is a systematic representation of a process as a set of dependent tasks accordingly to a set of rules [14]. The workflow model aims to automate and minimize the complexity of managing various types of processes, such as human activities, business processes and scientific experiments [15]. A data analytics workflow can be described in a directed acyclic graph (DAG), in which nodes perform data analysis tasks and edges perform the data dependencies between tasks [16]. Data analytics workflows have advantages over traditional workflows that inspire researchers to propose techniques and algorithms to optimize the increasingly high cost of running large-scale versions of these workflows on commodity resources, such as private, public and hybrid clouds. Data analytics workflows can also be referred to hybrid workflows.

This thesis describes the research on hybrid workflow scheduling on variety of computing systems. The next section describes the main terms used in this thesis.

1.1 Background

This section presents an overview of the essential concepts associated with the research problem addressed in the thesis.

1.1.1 Computing Systems

Cloud computing is an internet-based resources delivery system, which provides services on an on-demand and pay-per-use basis. The National Institute of Standards and Technology (NIST) [17] defines this paradigm as *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*. There are three types of cloud computing service: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as Services (SaaS). In IaaS, the cloud provider delivers physical computing resources – such as storage, servers and the network – as services. The resource virtualization on clouds offers a full computing stack to end users allowing them to use cloud resources on a pay-as-you-go basis. PaaS provides a computing platform to build applications, and SaaS grants online-software use subscriptions with online APIs for software integration, upgrades and patches.

The aforementioned features make cloud computing the preferred environment to host big data applications; it can minimize monetary cost through scalability and a pay-as-you-go model, and minimize the makespan (total execution time) based on resource capability and availability. Hybrid workflows imply intensive processing pipelines, in terms of resources, computation, data, latency and bandwidth. These pipelines receive data as streams which are essentially processed in real-time or near real-time, and mostly for data cleansing and preparation. During this stage, stream tasks may receive a large amount of data generated by the IoT objects or devices.

Although cloud computing is one of the most efficient computing systems, with massive processing and storage capabilities, but it is not the ideal computing system for latency-sensitive applications, such as real-time gaming, augmented reality and real-time streaming [18]. Because cloud resources are located close to the core network, the round-trip latency of these applications will be high subject to passing data through multiple gateways. Moreover, transferring large amounts of data with latency-sensitive constraints to such a centralized environment is insignificant in terms of the resource utilization, communication latency and energy consumption of computation servers [19]. In 2021, data produced by IoT devices and network access devices are expected to exceed 847ZB [4]. Rapidly rising demands and growing dependence on high-response information access and efficient data processing mean that edge-oriented computing paradigms have become a necessity for the IoT domain.

Edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services [20]. Basically, an edge is any resource type on the path from cloud to edge, which is able to provide services including computing, storage, and routing. However, a high percentage of processed data at the edge layer is temporary and only a small amount might be meaningful.

Edge computing has a significant role in processing massive data and only uploads processed data to clouds. For instance, in an autonomous vehicle, a huge amount of data, in the form of images and video, must be processed in real time to permit good driving decisions [21]. Sending data to the cloud not guarantee time constrained decisions. In contrast, edge computing is less effective in terms of network latency and unreliability than cloud computing [22]. Another concern of sending data to the cloud is privacy [23]. For example, in smart home applications, data generated from sensors could report sensitive data and this should be consumed within the scope of the private network. Edge computing is well suited for privacy protection due to local processing within controlled and burden communication systems.

Edge computing is capable of overcoming many challenges of IoT applications related to data transfer latency and cost, application responsiveness and data privacy. However, the application of edge computing has some limitations, notably for large-scale applications. Edge resources are limited in computation and storage capacities. For long-term processing, edge resources are not expected to obey the extreme data computation overheads [24]. In addition, edge nodes mostly do not support general purpose computing, because they are pre-programmed and tailored to handle specific types of data.

Edge collaboration is a desirable advantage of edge computing. Edges from different providers and stakeholders can communicate and share data in a geographically distributed manner, regardless of their physical location and network structure [25]. According to Dastjerdi and Buyya [26], in edge cloud computing, we must be concerned about specific resource management and scheduling techniques, including resource distribution, load balancing, migration and consolidation. With efficient and robust cloud edge collaboration, widespread applications in the form of hybrid workflows, in domains like real-time monitoring, autonomous mobile computing, traffic analysis, and crowdsensing, can benefit from the powerful capabilities of cloud computing as well as the approximate computing offered by edge computing.

The next section explains the hybrid workflow concept in terms of structure and computation requirements.

1.1.2 Hybrid Workflows

Generally, a workflow is a systematic representation of a process as a set of dependent tasks according to a set of rules, while workflow structure consists of tasks, data elements, control sequences and task dependencies [14]. A common and efficient representation of such applications is a DAG structured workflow model [16]. DAG representation is widely accepted in many data analytics domains, including

bio-informatics, high-energy physics and astronomy. In this thesis, a hybrid workflow is defined as an integrated application model of stream and batch processing models. The expansion demands of data analytics applications pose challenges for offline and online processing, which refer to stream and batch processing systems, respectively. A wide range of applications are obliged to combine these processing forms [27, 28].

Workflows can be seen from two main perspectives: business and scientific. The workflow management coalition [29] defined a business workflow as "the automation of a business process where inputs (document, information, or tasks) are passed from one business actor to another for action according to a set of procedural rules". Scientific workflows are used for modelling and running scientific experiments. Although business workflows and scientific workflows share the concepts of workflow representation and abstraction, they have many practical differences. Firstly, business workflows have a lower abstraction level as they mostly follow business-specific programming models to meet business requirements, while scientific workflows employ high levels of abstraction to validate scientific hypotheses [30]. Secondly, business workflow tasks can be processed by machines or humans, but in scientific workflows, the scientist's role is to monitor workflow execution and interrupt the execution flow when needed. Finally, scientific workflows are data-centric process flows, while business workflows are controlled by strict procedural rules to identify the flow of business process [31].

A hybrid workflow is the integration of stream and batch data processing models in one data processing pipeline. This thesis proposes algorithms and techniques for hybrid workflow scheduling for different computing systems. Table 1.1 summarizes the main challenges of data analytics workflows processing. The work provided in this thesis handles the hybrid workflow scheduling with two high-level objectives. The first is to illustrate the importance of understanding why and how stream and batch tasks are communicated and collaborate with respect to differences in their specifications, constraints and structure. This thesis provides detailed modelling of each task form. The second objective is affiliated to determine the ability of

the adopted computing system to overcome the challenges of hybrid workflow scheduling.

An example of a situation where hybrid workflow scheduling is critical is anomalous events detection for water distribution systems [32]. High-frequency flow data events are collected and validated in real time. Historical events are categorized and outliers are detected offline. Another example is real-time medical data analysis, where collected data from attached health sensors must be analyzed continuously in real time to enable timely decisions about the patient's care plan. Moreover, medical data need to be aggregated from multiple patients' profiles for smarter disease prediction and treatment [33].

Enabling intelligent transportation systems is another significant application of real-time data analytics. Sensing technologies are located everywhere in big cities and crowded streets to monitor traffic conditions. Road monitoring cameras, GPS and vehicle sensors are common examples of these technologies [34, 35]. These technologies collect real-time data and share it via a variety of communication protocols, like WiFi, 5G and Bluetooth, and various types of data are collected, including vehicle locations and speeds, driving behaviours and road conditions. For smart cities an enormous number of devices and applications must be integrated, which makes it challenging and complicated to accommodate the heterogeneity of these applications in an automated and collaborative paradigm [36]. An application service in this context is finding the quickest route based on current traffic conditions [37]. In smart cities, drivers are able to communicate with 1) each other in real-time through ad-hoc networks, 2) roadside traffic units, and 3) traffic authority management [38]. In more complex scenarios, such as emergency situations requiring an ambulance, drivers also need to communicate with health authorities. This scenario involves complex interconnected services and thus substantial service management system at different layers. Hybrid workflows support this scenario, and accordingly many other smart city services.

TABLE 1.1: Data analytics workflows challenges [2]

Issue	Challenge
Scalability	Well–equipped, efficient management, and resource utilization.
Availability and fault tolerance	Fault Detection, and timely resource availability.
Computing complexity	Big data programming environments are complex to configure.
Elasticity	High scalability according to system state and changes in data processing requirements. Stream processing is the major challenge.
Resource requirements prediction	Understanding resource usage can provide significant details for resource provisioning and load balancing.
Time-sensitive applications	Resource scheduling should be convenient in meeting user deadline expectations and data locality.

1.2 Problem Definition: Hybrid Workflow Scheduling in Cloud Systems

Stream and batch processing have different fundamental processing QoS requirements. Stream processing is latency-sensitive and subjects to constraints like stream rate and throughput [39]. Moreover, for applications like data analytics workflows, stream preprocessing can be a resource-consuming process under the constraint of real-time processing with high stream rate. Furthermore, data stream arrival rate may change over time making it hard to estimate data stream collection and processing intervals. Thus, is not trivial to determine the size of these intervals in advance. If the size is too large, the accuracy of prediction can deteriorate when the nature of the data changes, and if the size is small, the accuracy can worsen when that is rather stationary.

For stream application provisioning, there are two common ways to scale application performance: increasing operator parallelism and adding extra resources [40]. Parallelism can be increased via allocation of new operators on new hardware. The new hardware resources incur more costs, so it is more sensible to employ

more effective operator placement ("called scheduling"), which also impacts on the application service rate. However, producing optimized schedules can be a critical task when distributing parallel computations on heterogeneous processors [41] and prioritizing them to ensure a shorter schedule path execution.

Achieving a high system stability is a critical challenge for real-time processing over fluctuating big data streams. Stability takes priority over efficiency because rescheduling may be required to be dynamically undertaken at runtime. Big data stream is difficult to process in traditional computing systems: data is not available at once, data stream rate is often of high speed and may vary with time, and timely analysis of the data stream is critical. When the data rate is increasing, replicating vertex instances can guarantee meeting workflow execution deadline. When a task execution is parallelized on multiple instances, the input stream must be split onto all instances.

On stream processing, the system throughput depicts the processing rate of at a computation processor for the same data stream input [42]. On the other hand, batch processing has a less time sensitivity processing, but with high corresponding to data aggregation and predictive modelling function over high volume data. Basically, batch processing is not a time-sensitive process; it is more cost-effective than streaming processing due to the amount of resources required to handle large chunks of data. the processing models have different service quality measurements, which determines the efficiency of processing computation under particular QoS requirements. Overall, both processing models are considerable for processing quality or throughput which, determines the efficiency of processing computation under certain user and application QoS requirements.

The processing mechanisms for stream and batch processing systems need to be maintained to obtain a satisfactory level of application execution efficiency. Batch processing is designed for data correctness and completeness, while stream processing can achieve the desired level of efficiency even with a high blocking rate, particularly at peak loads. In a hybrid workflow model, a task represents the execution of either a stream or batch application. The integration between stream

and batch processing models focuses attention on issues such as how to control the execution of continuous and discrete processing? how to manage resources to satisfy the variation in quality of service (QoS) measurements of each processing model? what is the role of stream properties (such as rate, aggregation window length, blocking rate, latency and throughput) on overall application performance? Finally, how does stream processing parameters tuning empowers the performance of resource provisioning and task scheduling?

The research outlined in this thesis focused on resolving three core issues of hybrid workflows:

- Complex and large-scale workflows with high number of integrated applications, (this is different from the traditional model in which the border line between stream and batch processing is clear and both can work as standalone applications),
- Short-term stream intervals and online batch feeding. This feature implies an iterative application processing with batch-based delivery at iteration level,
- Flexibility and parameter tuning. A hybrid model is scalable and adjustable to its parameters.

Workflow scheduling is the process of mapping tasks to computing resources and planning their execution in a way fulfilling dependency flow between tasks and achieving certain QoS parameters and resource preservation goals. It is designed to resolve the scheduling objectives and represent workflow execution performance metrics such as deadline, throughput, energy consumption, security, reliability, runtime, service level agreements (SLAs) and violation rate. The problem of workflow scheduling is composed of two sub-problems:

1. *Resource provisioning.* The provisioning step involves determining resource requirements, selecting and provisioning resources. Resource provisioning

decides which and what amount of resources to be allocated to meet scheduling needs as well as QoS performance constraints. The workflow scheduler should determine computation server (VMs) type, number of VMs to lease and operating interval. The provisioning policy can be static or dynamic. In static policy, heuristics can provide sufficient information about task workloads and the amount of data to be passed on according to workflow topology. However, dynamic policy is the favoured approach for auto-scaled enabled computing systems such as cloud computing. Dynamic provisioning involves estimating preliminary workloads at runtime to avoid issues like over-provisioning, under-provisioning and unbalanced workload distribution.

In hybrid workflows, data streams generated from sources like IoT devices and sensors are passed through processing systems (tasks) to formulate heterogeneous workloads. The amount of data to be processed at stream level is subject to parameters such as transmission medium quality, throughput and distance to computation servers. However, this complexity of stream data acquisition inherently affects the prediction of data received at batch level. Thus, resource provisioning in the context of hybrid workflows is a dynamic and complex task, and subject to understanding the conceptual interaction between heterogeneous workflow tasks.

2. *Task scheduling.* The scheduling process refers to mapping workflow tasks to provisioned resources and determining the sequence of executing these tasks. Task mapping in the context of hybrid workflows extends techniques applied on traditional workflows through consideration of the insensitivity to computation, data and bandwidth, and the sensitivity to latency and throughput. The workflow scheduler should effectively plan the mapping with the variety of resource types offered by computing environments. The scheduler mission is to construct a convenient scheduling plan which accomplishes execution QoS parameters while preserving constraints of stream and batch tasks.

   In hybrid workflow scheduling, application structure and computing environment are significant contributors to the articulation of frameworks for resource provisioning and task scheduling. The framework needs to consider

how application structure complexity and scale are aligned to the required amount of resources. For example, if the workflow is highly stream oriented, long-term provisioning will evolve. Moreover, the adopted computing environment poses resource selection and task allocation problems. For instance, the presence of edge computing extends the traditional systems capabilities to support decentralized location-aware and latency-sensitive applications. In the context of hybrid models, the evolution of emergent computing systems should be reflected on scheduling frameworks to achieve user QoS requirements of low cost and high system responsiveness.

This thesis proposes algorithms and techniques to overcome the challenges of hybrid workflow scheduling on computing systems. The next section illustrates the main thesis contributions.

1.3 Contributions

This thesis describes the study and proposal of solutions for hybrid workflow scheduling two main avenues were followed. The first is an examination of the modelling of hybrid workflows and how the proposed model can effectively meet the requirements of integrating stream and batch tasks. The second is an investigation of efficient algorithms, techniques and computing systems which are well suited to a hybrid model as well as achieving other desired QoS requirements, particularly cost and time. The contributions of this thesis are as follows:

1. **Hybrid workflow modelling.** This thesis outlines comprehensive modelling of hybrid workflows which exposes, at low level, the features of each processing system (i.e. stream and batch). This work involved detailed configuration analysis and formulation of each application system. This elaboration facilitates building cost and performance models to accomplish the requirements of a resource estimation and scheduling framework. The modelling illustrates the following points:

- Provides detailed configuration for each task type. For stream tasks, this includes data rate, data capturing period, processing throughput and queuing system alignment. For batch tasks, this includes data aggregation rate, processing throughput.
- Identifies the interaction between tasks and how this reflects the amount of data generated.
- Specifies how model parameters tuning incorporates resource requirements and QoS constraints achievement.

2. **Resource estimation techniques.** We extended the traditional workflow scheduling framework by adding another layer to allow estimating the amount of resources according to hybrid workflow configuration and overall execution time. The estimation process basically aims to find the amount of resources required to execute workflow tasks by considering the dependency between tasks. The outcome of the estimation phase is a group-based execution plan in which computation resources are minimized.

   The hybrid model determines how stream and batch applications interchange computation and data based on the dependency structure. Stream tasks fundamentally control workflow execution and thus computation and data handling requirements. Based on this assumption, the estimation process targets stream tasks and searches stream configuration space to find the one which satisfies workflow execution constraints with the least amount of resources and time. Moreover, the estimation process corresponds closely to the hybrid model though controlling the scaling of the entire workflow execution while meeting the constraints of each task. To achieve this objective, three estimation techniques are proposed.

   (a) Particle Swarm Optimization (PSO) is utilized to search stream configuration space for the best configuration plan. The plan determines the setup of all stream tasks properties, by which the total amount of resources and time are reduced. Estimation results demonstrated

the viability of the PSO-based technique for small- and medium-scale workflows.

(b) For large-scale workflows, linear search optimization with gradient descent search (GDS) is proposed. The linear technique aims to overcome the complexity of large-scale workflows estimation by considering the functional behaviour of workflow execution. To support the linear estimation mode, we developed an offline execution function estimation based on profiled experimental outcomes.

(c) For more realistic estimation, we improved the GDS-based approach to handle unprofiled workflows. An online estimation function was developed under constraints of linear optimization approach.

3. **Scheduling on various computing systems.** For hybrid workflow scheduling purpose, we proposed a cluster-based optimization technique to provision and schedule hybrid workflows that relies on constructed groups from estimation phase. The scheduling technique considers the composition of hybrid workflows in terms of computation and data dependencies.

   The scheduling framework was applied on three computing systems, namely, cloud, edge cloud, and cooperative edge cloud. For each computing system, we provide an extensive analysis on how hybrid workflow concepts are incorporated with the computation characteristics of each system in terms of achieving QoS requirements as well as reducing over all workflows execution time and cost. The scheduling framework has the following specific contributions:

   - A group-based technique to achieve the optimized scheduling plan with consideration to the hybrid model. A significant correlation exists between workflow scaling through controlling workflow configuration, and scheduling optimization results.
   - Proposing different versions of the group-based scheduling technique which align the properties of each computing system. The scheduling technique design considers the capabilities of resources, the quality of

the networking system and the level of collaboration between computation resources.

1.4 Thesis Organization

Figure 1.1 shows the thesis structure. The chapters are described briefly below:

- Chapter 2 presents a review of the main research works related to the concepts addressed in this thesis. The literature reviewed here covers resource provisioning and task scheduling on different computing systems and for different optimization objectives.

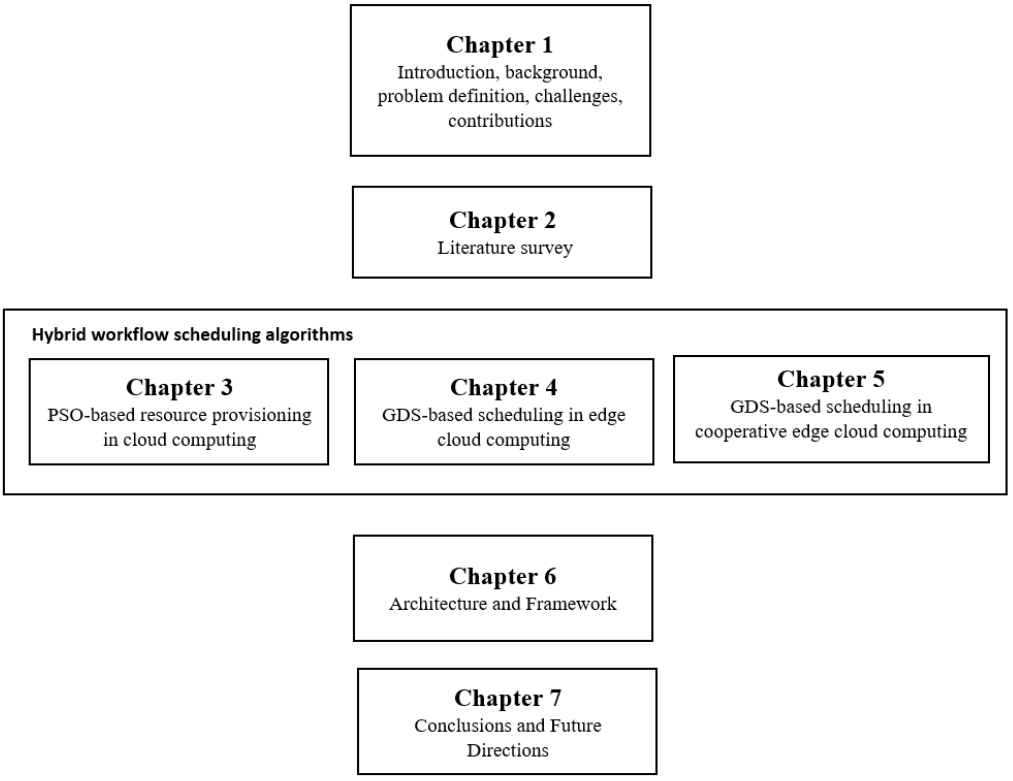

FIGURE 1.1: Thesis organization.

- Chapter 3 (Cloud Resource Provisioning for Hybrid Stream and Batch Workflows) presents a PSO-based resource provisioning scheme for hybrid workflows in cloud systems. This work has been presented as:
  - Alsurdeh, R., Calheiros, R.N., Matawie, K.M. and Javadi, B., 2018, November. Cloud Resource Provisioning for Combined Stream and Batch Workflows. In 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC) (pp. 1-8). IEEE.
- Chapter 4 (Hybrid Workflow Scheduling on Edge Cloud Computing using Gradient Descent Search Approach) presents GDS-based resource provisioning and task scheduling for hybrid workflows in edge cloud systems. This work has been presented as:
  - Alsurdeh, R., Calheiros, R.N., Matawie, K.M. and Javadi, B., 2020, July. Hybrid Workflow Provisioning and Scheduling on Edge Cloud Computing Using a Gradient Descent Search Approach. In 2020 IEEE 19th International Symposium on Parallel and Distributed Computing (ISPDC). IEEE.
  - Alsurdeh, R., Calheiros, R.N., Matawie, K.M. and Javadi, B., Hybrid Workflow Provisioning and Scheduling on Edge Cloud Computing. Future generation computer systems, *in submission*.
- Chapter 5 (Hybrid Workflow Provisioning and Scheduling on Cooperative Edge Cloud Computing) presents GDS-based resource provisioning and task scheduling for hybrid workflows in cooperative edge cloud systems. This work to be presented as:
  - Alsurdeh, R., Calheiros, R.N., Matawie, K.M. and Javadi, B., Hybrid Workflow Provisioning and Scheduling on Cooperative Edge Cloud Computing. In 21th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE/ACM, *in submission*.

- Chapter 6 (A framework for Hybrid Workflow Scheduling in Edge Cloud Systems) presents the design and architecture of an edge cloud computing framework for demonstrating resource provisioning and scheduling algorithms. The proposed architecture is an extension of the 'CloudSim' simulator, and serves to simulate the hosting computing environment for applying framework functionalities.

- Chapter 7 (Conclusions and Future Directions) concludes the thesis, summarizes its findings, and suggests directions for future work.

Chapter 2

Literature review

The IoT is reforming the future of connectivity and reachability. Enormous number of objects to be online connected leading to extensive data generation that threatens to overwhelm storage systems and cause a significant surge in application reaction time. McKinsey [43] estimated that 1 trillion IoT devices will be interconnected by 2025. According to this estimation, by 2025 the IoT will have an economic impact of USD 11 trillion per year, represents 11% of the global economic output. IoT application models formulate the integration between stream and batch processing to achieve data analytics objectives. In this thesis, this integration is defined as a hybrid workflow model.

In previous chapter, section 1.1.2 discussed the main concepts of hybrid workflows and how they bring additional challenges over traditional workflows are highlighted. These can be summarized as complexity and high scalability due to the high number of integrated applications, a short-term application delivery and iterative model, and high sensitivity to structure and configuration parameters. In addition, section 1.2 highlights the main challenges of hybrid workflow scheduling for both resource provisioning and task mapping. This chapter presents an extensive review of research on resource provisioning and workflow scheduling for various computing systems and optimization objectives.

2.1 Resource Provisioning and Workflow Scheduling

Workflow scheduling is one of the high demanding research directions in academia and industry, and there is an extensive research body on resource provisioning and workflow scheduling. Many algorithms and techniques have been proposed for scheduling optimization in computing environments, including the grid [44], cloud [45, 46], multi-cloud [47], and edge [48]. The main objective of workflow scheduling is to generate scheduling plan(s) that maximize efficiency under certain optimization objective(s) such as makespan, execution time, monetary cost, reliability and resource utilization. [15]. Directed acyclic graph (DAG) is commonly used to represent a scheduling plan; vertices denote workflow tasks, and the arcs shows the dependencies among them [16]. Workflow and DAG are often used interchangeably in the literature.

Hybrid workflow scheduling implies an understanding of the differences in processing behaviour of stream and batch applications in order to propose workflow schedulers which support maintainable integration between these applications with consideration of user QoS constraints. In this chapter, we provide a broad analysis of the research body on workflow scheduling, including resource provisioning and task allocation in different computing systems.

2.1.1 Resource Provisioning in Cloud Computing

In cloud computing, resource provisioning is an adaptive process of provisioning and deprovisioning resources according to workload changes, and accordingly meet the requirements of service demand at a certain point in time [49]. Resource provisioning is critical to control resource utilization and cost by avoiding resource over-provisioning. Provisioning is estimated by reactive or proactive mechanisms. Reactive mechanisms continuously track service workloads and fire scaling triggers

in response to resource demands. However, the time needed to update and apply a new provisioning plan can be destructive, particularly for latency-sensitive applications which involve real-time or near real-time processing.

Proactive mechanisms are efficient in incorporating timely-constrained provisioning by observing workloads and predicting resource demands through applying statistical or mathematical models, such as queuing theory, reinforcement learning and control theory [50]. The magnitude of these workloads and their arrival patterns are frequently fluctuating and unpredictable according to user interactions. Therefore, to deal with this variability and simultaneously evade performance degradation and service level violations, dynamic and timely resources provisioning process needs to be considerable.

The control model is applied for parameter tuning automation of software components that are modeled based on queuing networks. Since the iterative optimization of the queuing models supports the accomplishment of admirable adaptability to changing environments, it also amplifies the runtime complexity for solving optimization models at each step. [51]. Vozmediano et al. [52] employed machine learning techniques on an SLA-based predictive auto-scaling mechanism that aims to estimate the processing load and provision the required number of servers in a cloud system to meet an application deadline and reduce energy consumption and infrastructure costs. The results show improved forecasting accuracy compared to other classical models. The work assumes auto-scaling on an hourly basis and does not consider how parameters like data aggregation time and processing throughput can affect the estimated amount of resources. Mao et al. [53] proposed two algorithms to resolve the auto-scaling issue: scheduling-first, and scaling first. The first algorithm applies total budget distribution to each workflow task, generates the fastest execution plan, and finally acquires cloud resources. The second algorithm estimates the required resources with regard to data size and the resource capabilities, and finally schedules the workflow tasks.

Malawski et al. [54] resolved resources auto-scaling in a cloud system by problem prioritizing under the budget and deadline constraints. The main objective

is to enhance workflow system throughput by adopting static and dynamic resource provisioning and workflow scheduling. The Dynamic Provisioning Dynamic Scheduling algorithm applies adaptive scheduling and resource provisioning to reduce resources cost. A pre-allocation is established and an initial number of VMs is calculated based on the budget and deadline. Periodically, VMs with low utilization are shut down and new VMs are leased based on budget. The scheduling phase maps tasks based on their priority on random VMs. One limitation of the algorithm is the unrestricted resource provisioning according to budget limit; this may result in low VM utilization due to the high idle time waiting for ready tasks.

Zhang et al. [55] applied the same fixed-term estimation strategy for cloud workload prediction based on stacked auto encoders. They used a canonical decomposition format to reduce the training time by compressing the input parameters. Nikraveshet al. [56] proposed a predictive auto-scaling system to scale the cloud resources automatically for different type of workloads and fixed observation window size. The work did not illustrate how workload features are injected on the prediction model. Klinkenberg et al. [57] worked on time series data prediction based on an adaptive sliding window. The technique is not suitable for time-sensitive application due to the complexity of real-time windows size identification. Deypir et al. [58] used an adjustment technique for stream arrival rate with variable window size. Experimental evaluations showed the proposed technique efficiency in adapting window size to improve system performance. However, the technique involves user intervention to set window size reduction threshold.

Warneke and Kao [59] considered the dependencies between application tasks, and determined the specifications of cloud resources needed to provide efficient resource allocation and scheduling framework. However, application modelling does not study how changes in workflow structure or input data are reflected in generated schedules. Shao et la. [60] provided an energy-aware scheduling approach for big data applications in cloud systems. The adopted workload model only considers the number of incoming jobs in a batch mode. Li et al. [61] presented a predictive scheduling framework for stream applications. Based on the application graph, the proposed work aims to reduce the average processing time

of incoming tuples. Their conclusions were that it is not trivial to determine the characteristics of data stream applications for the purposes of resource provisioning and allocation. Cheng et al.[62] considered the dependencies between jobs and applied different scheduling policies. Their proposed scheduler adjusts resource sharing and job parallelism schema by analyzing data collected from application performance profiling for parameters such as end-to-end latency and throughput.

The discussed research body covers many aspects of resource provisioning and scheduling for stream workflows. However, it contains the following gaps. Firstly, stream workflow modelling is not provided in a way the expresses data behaviour with respect to variation in stream configuration parameters of arrival rate and throughput. Secondly, for hybrid workflows, like data analytics, resource provisioning techniques do not comprise data transition between stream and batch application, and how parameters tuning can lead to convenient provisioning and scheduling plans. Lastly, work to date has focused on clustering-based scheduling; grouping stream and batch tasks is not covered. To overcome these limitations, the thesis proposed techniques and algorithms for hybrid workflows, including, resource estimation and provisioning, and task allocation on a cloud system.

2.1.2 Workflow Scheduling in Cloud Computing

Cloud computing is a service-based computing model, which offers computation, storage, networking and application services through a massive and interconnected resources topology for geographically distributed and high-quality data centres [63]. In the cloud computing environment, the pay-per-use cost model applies dynamic and cost-based resource provisioning to accommodate the diversity of application models and QoS requirements. The scheduling technique plays an essential role in cloud computing, and it is utilized to coordinate application execution to attain sustainable resources utilization with realization of constraints like execution deadline, budget, reliability, energy, security and throughput. Thus, well-formed workflow scheduling or task allocation can greatly improve overall workflow application execution performance [64, 65].

The rest of this section discusses recent research on resource provisioning and task scheduling in cloud computing.

Best-effort scheduling

Deadline-constrained scheduling algorithms consider monetary cost as the most important factor for building schedules. High-performance resources are always costly. Consequently, the workflow scheduler should be aware of the trade-off between execution time and monetary cost when allocating resources. The simplest solution for deadline constraint workflows is minimizing the critical path by selecting low-cost resources for non-critical tasks.

The heterogeneous earliest finish-time (HEFT) [64] algorithm tries to minimizes the overall workflow makespan through minimizing the earliest finish time for critical tasks. HEFT considers both the execution and communication time between resources. The technique is one of the best heuristics scheduling algorithms available [66, 67]. HEFT performs task allocation in two steps: task prioritizing and instance selection. Task prioritizing ranks tasks in a list according to the cumulative execution time on each VM instance and average communication time between VMs of dependent tasks. The unallocated task with highest rank is selected and mapped to its best instance, which guarantees the lowest finish time. The predict earliest-finish-time algorithm [66] proposes a lock-ahead strategy without adding overhead-time on the overall scheduling process. The algorithm recomputes the average execution time for unallocated tasks to priority task executions. An optimistic cost table is used for calculating the time complexity.

Abrishami et al. [68] adapted the partial critical path (PCP) algorithm for deadline-constrain. The algorithm works by generating PCPs, and for each PCP, included tasks are allocated to the same VM instance based on heuristics. Grouping critical tasks can enhance resource utilization and reduce the total execution time while meeting the deadline. The technique can reduce computation time and cost, but the communication time and cost are not considered. In cloud computing, data transfer times can maximize the overall workflows execution time and cost. IC-PCP overcomes the issue by grouping and scheduling dependent tasks on

the same VM to reduce the amount of transferred data between VMs. However, IC-PCP does not provide an accurate estimate of execution and transmission time [69].

Zeng et al. [70] proposed a backtracking algorithm, ScaleStar, designed to generate an optimized schedule on complex infrastructure by recomputing the makespan and cost after each scheduling step. Sahni et al. [19] applied the grouping strategy by constructing pipelines of interdependent tasks aiming to reduce the workflow execution cost while meeting the deadline. However, no priority policy is applied to select the next allocated pipeline [71]. Lin et al. [72] proposed a technique to minimize the overall workflow execution delay considering the budget constraint. The scheduling problem assumes a one-to-one VM allocation to each workflow task. Fard et al. [73] proposed a bi-criteria algorithm that studies the impact of user constraints on schedule optimization. They studied four objectives: makespan, economic cost, energy consumption, and reliability. Bessai et al. [74] proposed a Pareto approach based on three resource selection policies: time-based, cost-based, and cost-time-based. Their approach differs from other works in assuming boundary levels for time and cost constraints instead of conflicting time and cost objectives. Verma et al. [75] suggested a Bi-Criteria Priority, Particle Swarm Optimization (PSO) to optimize the execution time under budget and deadline constraints. Their simulation results indicated a significant decrease in execution cost in comparison to the standard PSO algorithm. Chen et al. [76] proposed a novel multi-objective ACS-based (MOACS) a co-evolutionary multiple populations to optimize optimizes both execution time and execution cost.

Quality of service-aware workflow scheduling

Quality of service-aware workflow scheduling targets optimizing some objectives with constraints on other objectives. This is more related to real-world applications where the main aim is minimizing or maximizing an objective function while satisfying the user's QoS requirements [77]. A successful workflow execution is highly related to the degree to which QoS objectives are met.

Hen et al. [78] proposed an ant colony optimization approach to address makespan, monetary cost and reliability. The adopted algorithm allows workflow users to identify QoS boundaries with minimum QoS thresholds in domain-specific applications. The idea is to get the optimal user satisfaction with workflow execution outcomes. The Pareto approach gives more flexibility to estimate QoS requirements by generating a set of schedules. Durillo and Prodan extended the HEFT algorithm [64] by proposing the Multi-objective-HEFT (MOHEFT) algorithm [79]. MOHEFT is a generic multi-objective scheduling algorithm, which works by generating several scheduling solutions. The quality of a solution is assessed using metric of crowding distance. However, complete coverage traversing is adopted in MOHEFT to generate new solutions for assigning tasks to instances, which consumes a large amount of time. Thus, the technique is not efficient for large-scale workflows [76]. Zhou et al. [80] designed a fuzzy dominance sort-based heterogeneous earliest-finish-time (FDHEFT) algorithm to solve the workflow scheduling problem in the cloud. Compared to MOHEFT, FDHEFT achieves a lower time overhead by pruning the candidate tradeoff solutions by finding better solutions by using fuzzy dominance sort.

Since the workflow scheduling is an NP-hard problem, traditional methods such as dynamic programming or greedy algorithm are inapplicable to large scale workflow scheduling [66]. Evolutionary computation algorithms can provide better performance on complex optimization problems. Several evolutionary and meta-heuristics algorithms have been proposed for single and multi-objective scheduling. Meta-heuristics algorithms have the advantage over local search-based heuristics in generating scheduling plans in multi-dimensional searching on the problem space. Meta-heuristic algorithms, known as "non-deterministic" algorithms, have the objective of finding an optimal schedule by searching the resource space.

The genetic algorithm (GA) for task scheduling problem has been studied intensively in the literature. Wu et al. [81] proposed a novel GA scheduling algorithm to automate GA parameters selection by adopting a dynamic technique to evolve both solution structure and value. Wu et al. [82] proposed a PSO-based algorithm and, A revised discrete PSO. The algorithm considers both computation cost and

communication cost. Rodriguez et al. [83] studied VM features (booting time, performance, etc.) and resource provisioning to determine the optimal scheduling strategy in the public cloud. They proposed a novel PSO-based algorithm with consideration of the aforementioned parameters which had not been considered previously.

For cost optimization, Pandey et al. [84] proposed a PSO-based heuristic scheduling algorithm. The main objective is to reduce the total data transfer and execution cost for data-intensive workflow applications in public clouds. The idea was to effectively distribute all costs on workflow tasks, where the cost on each resource is calculated independently. The trade-off between cost and time optimizing is a considerable challenge for scheduling algorithms; reducing the task execution time is mostly involved a larger investment and higher cost, while a low investment often leads to poor time efficiency. As a result, take execution time and cost should be jointly considered by the scheduling schema. Bilgaiyan et al. [85] worked on the same idea, but using cat swarm optimization (CSO) [84] instead of PSO-based optimization to minimize the number of iterations [86].

Kumar et al. [87] proposed a time and cost optimization for hybrid clouds algorithm to minimize the execution time and cost of multiple workflow scheduling on hybrid resources. The algorithm helps to decide on target execution resources. Malawski et al. [88] proposed a deadline-constrained mathematical model to optimizes workflow scheduling in a multi-cloud environment. To reduce communication time and cost, the method proposes global shared storage for intermediate files. The scheduling and data placement problem is formulated as a mixed integer optimization problem. The authors adopted a group-based scheduling technique, in which tasks are grouped based on their computational cost and received/generated data. They assumed no resource sharing among groups, but this can degrade resource utilization and incur higher costs for complex and large-scale workflows.

Big data workflow scheduling

Big data workflows are designed to handle huge amount of data and is emerged as a datacentric workflow approach to analyze data with large scale, high complexity,

and high rate of acquisition [89]. Commonly, datacentric workflows are modelled as a DAG of data processing tasks with a set of data dependencies between the tasks. Three main features should be considered for big data workflows. The first is the variety of data sources and data formats. Data from different resources may differ in data transfer rate and speed, and data processing performance. This difference intensely influences the workflow design and execution [90]. Consequently, the datacentric workflow representation should be adapted in a way stating the diverse types of data. The second feature is decentralized workflow scheduling. Task execution is moved to the distributed computation node closest to the data location resource. This is definitely reasonable when the input data sets are very big. The decision to perform the remote execution should be undertaken in the workflow deployment stage for better workflow execution optimization. The third feature is data placement. Some scientific workflow execution involves intermediate data storage [91]. In big data applications, the intermediate data size can be rapidly increased during the execution phase, thus, there is a necessity to adopt a distributed computing environment to attain an efficient data execution [92].

Mirshekarian et al. demonstrated the statistical correlation between the datacentric workflow scheduling problem and flow-shop scheduling, which is a special case of the JobShop scheduling problem [93]. The JobShop scheduling objective is to reduce the total execution time by varying the allocation of independent tasks on machines. Albrecht et al. [94] proposed a makeflow framework for executing data-centric workflows. The framework emphasizes the accuracy in representing the correlation between the workflow task description and the scheduling plan. The model supposes a solid relationship between obtaining sufficient information about datasets and jobs, and effective workflow execution. In addition, the model takes into consideration parameters including data transfer latency, communication bandwidth and processor capacity. Deng et al. [2] applied a task duplication technique on distributed data locations. The scheduling process can rely on different allocation plans to minimize the data transfer cost and time. Yuan et al. [95] built data-centric workflow based on an intermediate data dependency Graph.

Data provenance determines the intermediate data hosts, and therefore, the scheduler can construct the optimized scheduling to reduce the data transfer and storage costs. Chervenak et al. [96] suggested a resource policy which can describe the current resource and data models based on heuristic information gathered from previous data transfer, storage and staging processes. Ghafarian and Javadi [97] proposed a scheduling algorithm for data-intensive workflows on distributed resources under the deadline constraint. The work suggested workflow partitioning to minimize the data dependencies, and thus minimize data transfer cost and time.

In addition to the datacentric scheduling issue in big data applications, which is mostly related to the high volume of data to be processed or migrated between computation machines, real-time scheduling is an issue due to high acquisition rate as well as the hard deadline constraint to process incoming streams at runtime. Real-time systems are becoming more and more popular; they measure the quality of computation correctness by its logical correctness and time results [98]. Violating the time constraint might harm application correctness. The quality of real-time scheduling algorithms directly affects application performance in terms of throughput, response time and application correctness [99].

A highly related concept to real-time processing is stream processing. Both processing models share the latency-sensitive feature, but with less restriction for the stream-based model. Stream workflow is the modelling of complex and continuous data processing through a set of connected processing tasks (operators) while formulating the concept of pipeline processing. Continuous processing differentiates stream-based workflows from traditional workflow systems in that the processing pipeline remains active to process an infinite stream. Stream workflow scheduling is a complex problem and hard to maintain with traditional batch-aware scheduling algorithms. This is due to the wide distribution of data sources, real-time constraints, the necessity of high-dynamicity resource management techniques to overcome the challenges of data rate variation as well as the heterogeneity of processing operators, and data locality, because a high volume of intermediate data might be generated as stream processing in progresses.

Most stream scheduling algorithms are designed and integrated with stream processing frameworks. A sufficient strategy to reduce the total data movement is a cluster-based approach, which implies assigning interrelated tasks in one cluster, and then applies resource allocation at cluster-level [100]. Venkataraman et al. [101] built a micro-batch stream processing system on top of Apache Spark. The system considers both throughput (successful percentage of processed data) and latency. To overcome the limitations of centralized scheduling, the system adopts the group/cluster-based scheduling approach in which batches are combined to enlarge computation granularity, and thus reduce the computation cost. One issue is data dependencies between tasks. To resolve this, the authors proposed a pre-scheduling technique based on local schedulers and queuing systems.

Spark's scheduler [102] schedules jobs in first-in-first-out (FIFO) mode while considering dependencies between application jobs. The scheduler experiences high latency for running long-term jobs. Storm's default scheduling algorithm applies an isolation technique to schedule typologies on static resources. An application owner can set computation resources, which makes the scheduler inefficient in terms of load balancing and resource availability. Peng et al.[103] built R-Storm as an extension to Storm, the proposed scheduling is a resource-aware scheduling approach with breadth first traversal to group jobs based on data dependency to decease transfer time and cost. Eskandari et al. [104] designed P-Scheduler, which implements an adaptive scheduling schema which involves weighting communication edges between graph nodes. The framework was applied in homogeneous computation environment and demonstrated a transfer time and cost reduction of 50% compared to the Storm FIFO scheduler. Meng-meng et al. [105] implemented a profiling technique to record workloads and data traffic between computation nodes. They proposed a dynamic scheduler designed to reduce the data communication over topology edges.

2.2 Workflow Scheduling in Edge Cloud Computing

Cloud computing offers pay-as-you-go and powerful resources which can reduce stream processing time and improve applications throughout. However, latency is a bottleneck for running stream application on cloud systems. Issues like streaming from devices over long distances with cloud resources and the unpredictable quality of transmission networks need to be considered in stream workflow scheduling. Computing models like edge computing and edge cloud computing are proposed to resolve such issues [106–110].

Edge computing is a computing model which brings the computation closer to data sources [111]. Edge computing is intended to overcome the challenges (limited bandwidth and network latency) of migrating large amounts of data for real-time data processing. Recently, many models have been proposed to conquer the challenges of stream workflows by adopting resource provisioning and task scheduling techniques for use in an edge cloud computing environment, to improve edge resource utilization, reduce latency by processing data-intensive tasks in nearby edges, maximize communication stability for high-performance stream processing, and provide an efficient task placement strategy to achieve reliable resource provisioning.

In edge computing, common challenges of workflow scheduling are scalability, self-adaptability and reliability [112]. Many scheduling techniques have been proposed for latency-sensitive workflows on edge computing. Yin et al. [113] proposed a streaming processing scheduling on cooperative cloud edge computing network to reduce the end-to-end latency of applications. Skarlat et al. [112] proposed three QoS-based workflow scheduling techniques for fog cloud resources, namely static, online and hybrid scheduling. The work adopted the concept of fog colonies, and assumed cooperation at colony level. The concept of fog computing is similar to edge computing in term of moving the computation closer to data sources [25].

However, the main difference between the two computing systems is where computing power is located. Fog computing relies on network resources and adopts computation transmission between data endpoints and network devices. Nevertheless, edge computing locates processing power in edge devices such as embedded automation controllers [114].

Sun et al. [115] proposed a two-level Genetic algorithm for resource scheduling on multi-edge clusters to reduce latency and improve system stability, Rahbari and Nickray [116] proposed a symbiotic organisms search based on the knapsack algorithm to reduce the delay and energy consumption in fog networks, and Deng et al. [117] introduced an approximation solution for minimizing the communication delay for workload allocation in the edge cloud environment. Pham and Huh [118] proposed a heuristic-based algorithm for workflow scheduling on cloud fog computing for optimizing the balance between execution time and monetary cost; however, the collaboration between fog nodes is not clearly stated, and only a traditional workflow model is handled.

Madej et al. [119] proposed a fairness-based scheduler for edge cloud computing. The paper compares four scheduling techniques, namely, FIFO, a client fair, priority fair, and a hybrid that accounts for the fairness of both clients and job priorities. The experimental results demonstrate that the hybrid technique is the best and that the fair scheduler is feasible to implement in edge cloud systems. Naha et al. [120] proposed deadline-constrained resource allocation and provisioning in the fog cloud environment. The algorithm addresses the issue of user QoS variation and resource limitations on fog computing. Resources are allocated based on a scoring system and considering various parameters. Zhou et al. [121] proposed an online gradient descent technique to estimate the rate of data streams, which reduces the cost of a cross-edge IoT data streaming system by adopting dynamic resource provisioning fr an edge environment. Ren et al. [48] concluded that shifting to real-time and context-aware IoT applications provisioning on the edge cloud requires tailored, transparent computing architecture for IoT applications, which can eliminate the advantages of edge computing. However, these authors made

some effort to propose estimation, resource provisioning, and task scheduling techniques in the context of hybrid workflows, such as a considering the integration between two different computing models.

Alsaffar et al. [122] proposed a decision tree approach for learning the provisioning model to construct a resource allocation plan in fog cloud system in consideration for computation request completion time and service complexity. Shao et al. [123] proposed a novel data replica placement technique for processing data-centric workflows in edge cloud system. The technique aims to reduce the communication costs of migrating large datasets while preserving the execution deadline. This work has many similarities with the research performed for this thesis. However, the latency-issue of data transmission is not clearly investigated and the data placement strategy does not apply a learning schema from heuristics to optimize future data placement through estimations. In addition, the edge cloud cooperation is not well-defined to overcome data transmission delay.

Long et al. [124] proposed a cooperative edge computing framework for delay-sensitive multimedia processing application. A greedy algorithm proposed to improve human detection accuracy under a deadline constraint. The framework only considers the work with lightweight stream workloads. Zhang et al. [125] proposed a game-theoretic framework for cooperative task allocation for delay-sensitive social sensing applications on edge cloud systems. The work assumed a selective edge resources approach based on rational actors who are unwilling to collaborate with others unless incentives are provided.

For complex workflows, such as hybrid workflows, a dynamic distributed and decentralized computing model, such as an edge cloud, is convenient for reliable and scalable execution [126]. With this model, the need arises for developing schedulers that can intelligently partition workflows and allocate the partition. According to Dastjerdi and Buyya [26], in edge cloud computing, we must be concerned about several resource management and scheduling techniques, including resource distribution, load balancing, migration, and consolidation. In addition, understanding data stream behaviour for stream applications is a worthwhile approach for VM

selection on optimal resource placements [127]. Beraldi et al. [128] proposed a general-purpose cooperative edge schema to reduce execution delay, the blocking percentage (improve service throughput) on edge data centres. An edge data centre migrates a service to a nearby centre when its service buffer is full. Edge cooperation has demonstrated a significant improvement in resource utilization for distributed computing systems [129]. Zafari et al. [130] concluded that heterogeneity, and lack of storage capacity and computation capability of edge resources, are the main drivers of edge cooperation to enhance their utilization and reduce outsourcing to cloud data centres.

2.3 Summary

This chapter provided an extensive background research on various directions to cover the main aspects related to this thesis, which covers the two main research areas, which workflow scheduling and edge cloud computing environment. To illustrate the interaction between these research areas, we synthesized the research work undertaken on resource provisioning and workflow scheduling, including application models, and the impact of adopting a certain computing system on scheduling performance. To summarize, the literature shows a variation on implementing edge cloud and cooperative edge cloud to resolve issues of latency-sensitive and resource-intensive applications. The following conclusions can be made. First, edge cooperation is limited to edge data centre level, and cooperation mechanisms are not well defined or structured. Second, hybrid workflow scheduling has not been investigated, and the integration between stream and batch processing is not clearly associated with the scheduling process. Finally, the impact of stream and batch processing parameter tuning has not been studied. For example, how change in stream arrival rate or processing throughput could affect the scheduling behaviour and optimization decisions has yet to be explained thoroughly.

Next chapter describes the work on dynamic resource provisioning for hybrid workflow execution on cloud systems. The provisioning framework is designed to optimize the cost of running hybrid workflows on cloud resources by considering the hybrid structure and performing efficient workflow execution parameters' tuning.

Chapter 3

Cloud Resource Provisioning for Hybrid Stream and Batch Workflows

The literature analysis provided in the previous chapter highlighted certain issues in context of resource provisioning for running hybrid workflows. These issues are summarized in two points: the limited work for recognizing hybrid workflow structure when proposing and developing provisioning frameworks, and the lack of studying the contribution of stream features' tuning on optimizing the cost of running hybrid workflows on cloud systems. This chapter proposes a resource estimation and provisioning framework for hybrid workflows in cloud systems, which aims to find an optimal workflow configuration plan that tries to find an optimal join optimization for workflow monetary cost and execution time. For resource estimation, a meta-heuristics optimization technique, Particle Swarm Optimization (PSO) was adopted to find the optimized workflow configuration plan which requires the minimum number of computation units while achieving the constraints of deadline and throughput. For resource provisioning in cloud systems, a group-based technique was utilized to find the resource provisioning plan that optimize the combination of workflow execution monetary cost and time. Results showed

the framework capability of controlling the execution of hybrid workflows by efficiently tuning several parameters, including stream arrival rate and processing throughput. For large scale workflows, the execution time and cost can be reduced by 45% and 30% on average, respectively.

3.1 Introduction

The increasing adoption of IoT the technology in many application domains generates a new need for rationalized utilization of computing resources supporting such computations. IoT applications can be represented as workflows in which stream and batch applications are integrated to accomplish data analytics objectives which can be referred to as a hybrid workflow model. As stated in section 1.1.2, a hybrid workflow represents an integration between stream and batch processing models. The hybrid model satisfies the following properties: 1) complex structure and number of interconnected applications. We assumed unconstrained data dependency model between applications. In contrast, a traditional data analytics models enforces a coarse-grained data dependency between stream and batch processing. 2) Fast delivery with iterative production at hybrid level. A hybrid workflow scheduling is defined as the management of QoS optimization for short-term and iterative application executions with hard-constrained of waiting time and throughput. Short-term service delivery workflows allow dynamic systems to adapt their behaviour. For example, short-term forecasting in transportation systems allows producing alternative routes to avoid traffic congestion before gridlock [131]. 3) Flexibility and parameter tuning. A hybrid model is scalable and adjustable to its parameters. Stream processing performance quality is sensitive to stream rate and processing latency. For example, in health monitoring systems, achieving data provenance required an immediate response to medical alerts by retrieving related data and undertaken needed actions [132].

Most IoT-based applications are comprised of a large and complex structure of interrelated tasks which are demanding with respect to computation, storage and

bandwidth [133]. The hybrid approach benefits from the capabilities of batch and real-time processing in big data application domains [134]. Hybrid workflow modelling is applied in many application domains, such as traffic monitoring [135], crowdsensing and social data mining [136], and weather data analysis [137]. All mentioned applications satisfy the conditions of complexity, fast delivery and scalability.

Research on hybrid models of stream and batch application is not new, and several architectures have been proposed in industry and academia. For example, Apache Beam [138] provides a unified model for both stream and batch processing to build a processing pipeline (DAG). Apache Beam works on top of computation engines, like Apache Flink [139], in which the optimizer enumerates different physical plans to achieve a cost-efficient execution on physical resources. The cost optimization incorporates network, disk and I/O usage costs. The execution cardinality performance was improved by proposing advanced estimations based user-defined functions, Flink's optimizer can use hints provided by the programmer. However, according to IBM researchers' experiments [140], the framework is unable to handle large variation in stream processing window size, causing processing overlap among several computation periods.

This chapter describes research into the execution of hybrid workflows in cloud systems. Cloud resource provisioning has received considerable research attention in recent years, and many models have been proposed to overcome the challenge of reducing resource usage cost while balancing other quality measurements [141–143]. To best of our knowledge, none of the workflow resource provisioning and scheduling techniques have advanced a detailed proposition for how to incorporate the structure of hybrid workflows (as a integration of stream and batch applications) as well as meet the challenges of iterative system production and sensitivity to stream configuration properties.

This chapter provides an adaptive resource estimation and provisioning framework for hybrid workflows in cloud systems, and makes the following contributions.

- A resource provisioning algorithm for multi-objective optimization of monetary cost and execution time for hybrid workflows that considers the characteristics of stream and batch applications, and the dependencies between tasks.

- An adjustment-based resource estimation algorithm for stream applications using PSO. The algorithm adjusts the stream arrival rate and aggregation windows size to optimize the cost of resources while satisfying the task execution throughput.

- A group-based resource provisioning strategy on cloud resources. The strategy implies grouping non-dependent tasks and performs cumulative resource provisioning in a periodic manner which can minimize the monetary cost of long-term workflow executions.

The rest of this chapter is structured as follows. Section 3.2, discusses related works on workflow resource provisioning and scheduling on cloud systems. A detailed description of hybrid workflow modeling is provided in section 3.3, the estimation and provisioning framework is provided in section 3.4, and section 3.5 shows the experimental findings and provides insights about the main results. A chapter summary is provided in section 3.6.

3.2 Related Work

Resource provisioning is an adaptive process of determining the resources needed to accomplish the execution of an application based on workload changes, structure complexity and QoS requirements. In cloud systems, it is essential to implement a reasonable provisioning plan to avoid the cost overhead of over-provisioning or QoS violation of under-provisioning. Experiments by Calzarossa et al. [144] demonstrated the correlation between auto-scaling policies, performance in improving VM utilization and the workload characteristics and patterns. Resource provisioning of hybrid workflows implies the recognition of batch and stream workloads,

which are identifiable according to their processing features and requirements [145]. Batch workloads fundamentally represent long-term computation-intensive execution cycles with minimal user intervention, while interactive-workloads, i.e, stream workloads, in contrast, refer to live and short-term request/response computation cycles with high parallelization. Moreover, stream workloads are infrequently deterministic, alternatively they are characterized by patterns that reflect user behaviours and interactions.

Sections 2.1.1 and 2.1.2 provide a broad discussion of resource provisioning and workflow scheduling in cloud computing. In the context of hybrid workflows, predictive provisioning mechanisms can handle the dynamic nature of stream workloads and accordingly the variation on batch processing resource demands with respect to the computation dependency incorporated by the hybrid model. Several predictive provisioning techniques are proposed in the literature. Some research considered the dependency between workflow tasks and proposed techniques to reflect the correspondence between application structure and amount of resources [59, 61, 104]. However, the provided application models do not study how changes in workflow structure or input data reflect on generated schedules. Cheng et al. [62] considered the dependencies between jobs and applied various scheduling policies. Their proposed scheduler adjusts resource sharing and job parallelism schema by analyzing collected data from application performance profiling for parameters such as end-to-end latency and throughput.

Control theory with a sliding window technique is widely used for stream application auto-scalling [56–58]. The adopted prediction strategies work by automating configuration parameters at runtime and with fixed time intervals. These authors did not illustrate how workload features are injected into the prediction model, and also there are not visible for time-sensitive application due to the complexity of real-time windows size identification. Recently, Vozmediano et al. [52] proposed an SLA-based predictive auto-scaling in a cloud system to meet application deadlines and reduce energy consumption and infrastructure costs. The work assumed hourly auto-scaling and did not consider how parameters like data aggregation time and processing throughput can affect the estimated amount of resources.

Group-based provisioning is a common strategy, particularly for batch and micro-batch workloads. Abrishami et al. [68] adapted the PCP to group critical tasks to enhance resource utilization and meet application deadlines. Venkataraman et al. [101] applied group-based provisioning for stream applications on top of Apache Spark. The system considers both throughput (successful percentage of processed data) and latency. The work applied fixed clustering approach and does not incorporate the dataflow structure. In this chapter, the group-based technique is extended by considering a dynamic execution critical path (CP) in relation to stream configuration parameters and dependency between batch and stream tasks.

3.3 Hybrid Workflow Model

A hybrid workflow $w = G(T, E)$ is a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$ formulated as a DAG, and has a dependency schema $E = \{e_{12}, e_{13}, \ldots, e_{nm}\}$ [16]. Table 3.1 presents the mathematical notations used in application modeling. Two tasks t_i and t_j are connected if and only if a direct edge e_{ij} exists in E. A task t_i can represent one of two application types, stream ${t_i}^S$ and batch ${t_i}^B$. The terms "task" and "application" are interchangeably used in this chapter. Specifically, a task t_i is the abstraction of an application, which may include additional layers of complexity and dependency. For example, a stream task ${t_i}^S$ represents a set of processing operators in a certain stream execution pipeline [146]; meanwhile, a batch task ${t_i}^B$ forms a complete flow of training for a classification model [147]. Note that the internal structure and behaviour of tasks is out of the scope of this thesis. Figure 3.1 shows an example of a hybrid workflow structure.

3.3.1 Stream Task

A stream task ${t_i}^S$ has the following features:

$$ {t_i}^S = \{\lambda_i, \mu_i, \omega_i, \tau_i, \alpha_i, d_i, c_i\} \tag{3.1} $$

Queuing system is a common technique for seamless and transparent stream application auto-scaling [148]. Here, the execution of a stream task ${t_i}^S$ is modeled as an $M/G/c$ queuing system [149]. The main objective of the adopted queuing modeling is to estimate the number of servers or parallelization level needed to run the application under constraints of system utilization, waiting time, and system throughput.

The adopted queuing system has the following aspects: a data stream arrives to an infinite waiting queue that is served by c identical servers, the server service time is a random variable with general distribution and mean $1/\mu$, the interarrival times between element arrivals to the queue is random variable with mean $1/\lambda$, and all service and interarrival times are assumed independent. Arriving stream

TABLE 3.1: Mathematical notations

Notation	**Description**
t_i	Application task, either stream or batch
λ_i	Stream arrival rate (msg/s), which refers to the number of received messages (msg) per second. The message size is an application dependent. At workflow level, we assume a single stream data source
ω_i	Stream processing aggregation window size(s) The window size refers to the time until a stream task publishes its output for next batch task or storage
μ_i	Application (task) service rate (msg/s), which expresses the capability of a server to process incoming messages. A server is an abstraction of a computation unit, and it used for the purpose of resource estimation
τ_i	Application (task) minimum throughput, which is the percentage of processed received messages in a given time length
α_i	Application (task) data production factor, which determines the amount of data to be passed to the next batch task(s)
d_i	Amount of data generated by a task
ϑ_i	Batch deadline(s)
c_i	Number of computation servers.
ρ	System utilization, assumed to be < 1
W_q	Waiting time in queue
σ_s^2	The variance of a server service time
S^2	The coefficient square of a server service time variance

elements are served in first-come first-served (FCFS) manner, and only one stream element may receive a service from a server at a given time. Such system is often referred to as $M/G/c$, following Kendall's notation [150] (M indicates a Poisson distribution for the interarrival rate, G indicates a general distribution for service times, and c is the number of servers).

Modeling a stream application as a $M/G/c$ queue helps in predicting the performance measurements of an application, such as resource utilization and data throughput. Dor et al. [151] investigated the correspondence between a relatively simple queuing model of an FPGA-accelerated BLAST implementation and empirical measurements taken from executions of the actual application. The study shows that simple queuing networks can accurately model the performance of a heterogeneous stream application. Li et al. [61] argued that queuing systems have better performance under certain assumptions of a data stream application.

It is assumed that at a given moment of time, all servers c will be busy. Based on that, Hokstad [152] proved that we can treat $M/G/c$ with service time ($S = 1/\mu$) as a $M/G/1$ with service time ($S = 1/\mu c$). The approximation by Kleinrock [149] is used as:

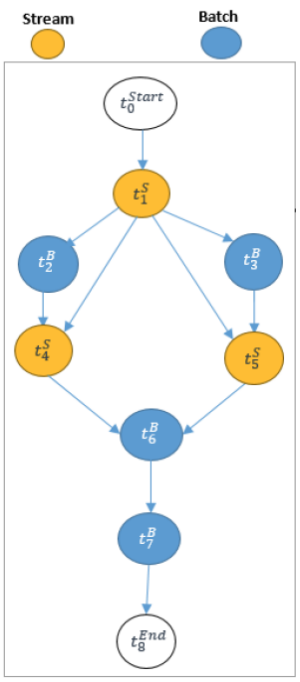

FIGURE 3.1: Hybrid workflow example

$$\rho = \frac{\lambda}{\mu c} \tag{3.2}$$

$$W_q = \frac{\rho(1+S^2)}{2(1-\rho)\mu} \tag{3.3}$$

$$ES^2 = \sigma_s^2 \mu^2 \tag{3.4}$$

Based on the queuing formulation, the number of servers c_i required to run task t_i under minimum system utilization ρ, is computed as:

$$c_i = \frac{\lambda_i}{\mu_i \rho} \qquad \rho < 1 \tag{3.5}$$

3.3.2 Batch Task

A batch task ${t_i}^B$ has the following features:

$${t_i}^B = \{\vartheta_i, \tau_i, \alpha_i, \mu_i, d_i, L_i, c_i\} \tag{3.6}$$

A batch task receives data collectively from stream and batch tasks. To estimate the number of servers c_i or the parallelization factor for a batch task ${t_i}^B$, firstly, the total amount of received data L_i is computed as follows:

$$L_i = \frac{\tau_i \sum_{j=1}^{n} d_j}{\mu_i} \tag{3.7}$$

Where d_j is the amount of data generated from task t_j, and then the following formula is applied:

$$c_i = \frac{L_i}{\vartheta_i \alpha_i} \tag{3.8}$$

3.4 Resource estimation and provisioning framework

Resource estimation is a multi-objective problem of minimizing the total execution time E and number of computation units (cores) R. The desired outcome of the resource estimator is a group-based workflow execution plan that can simplify the provisioning and scheduling in a computing system, particularly for complex hybrid workflows. It is assumed resource provisioning occurs on a group basis, which means allocating resources for all tasks included in a period of time as a bag of task (BoT) resource provisioning problem. Rodriguez and Buyya [153] demonstrated that fine-grained resource provisioning for BoT applications can reduce the execution time while achieving budget constraints. Thus, in the estimation process, the specific concern is to reduce the end-to-end execution time (critical path) E with the minimum number of computation cores among all execution periods R. The optimization function for the resource estimation was modelled as:

$$\min(E.R) \tag{3.9}$$

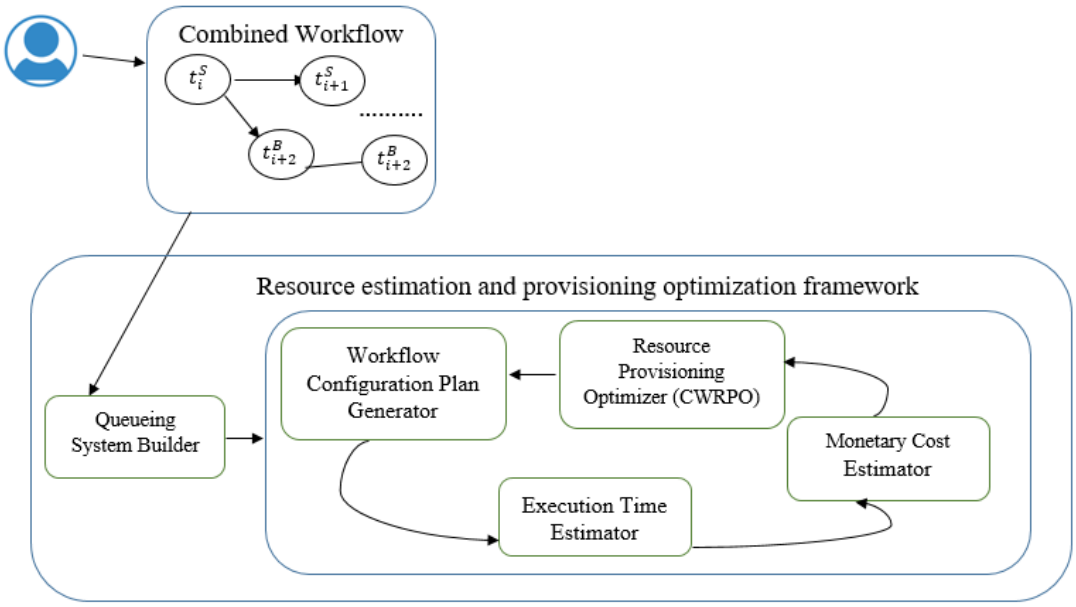

FIGURE 3.2: Resource estimation and provisioning framework

This section discusses the proposed resource estimation and provisioning framework for hybrid workflows. Figure 3.2 shows the main framework components and dependencies between these components, which are explained below.

3.4.1 Queuing System Builder

The first step after receiving a user workflow is constructing and validating queuing systems for stream tasks. This step validates the initial configuration of stream tasks to align the constraints of the adopted queuing system in terms of maximum waiting time, minimum system utilization and minimum workflow execution throughput.

3.4.2 Workflow Configuration Plan Generator

This is the core component of the framework in which workflow configuration plans are generated. A configuration plan is a set of values for stream applications that refers to properties of window size and arrival rate. Based on the dependencies between workflow tasks, these values have direct influence on measuring execution time and monetary cost of resources for both stream and batch applications. For the purpose of generating configuration plans, PSO technique was adopted [154].

Particle Swarm Optimization (PSO) is a stochastic global optimization method introduced by Eberhart and Kennedy [154] and based on simulation of social behaviour. As in GAs and evaluation strategies (ES), PSO exploits a population of potential solutions to prop the search space. PSO relies on exchange of information between individuals, called *particles*, of the population, called the *swarm*. Each particle adjusts its trajectory towards its own previous best position (*local best*), and towards the best previous position attained by the entire population (*global best*). This behaviour improves the converge time to get a global minimum with a reasonably good solution. A particle movement (new position) is coordinated by its velocity, which has both magnitude and direction. The particle velocity is

influenced by the particle's best position and the global best position, and also controlled by parameters including inertia weight and acceleration coefficients.

The objective of adopting PSO is to generate adjusted workflow configuration plans by randomizing data stream arrival rate λ and aggregation window size ω in order to minimize the optimization objective function and meet the constraints of throughput τ and deadline ϑ for each stream and batch application, respectively. This kind of PSO modeling is called constrained-PSO optimization [155].

Fitting a constrained-PSO method to the proposed optimization problem will identify both the fitness function and the particle structure. The fitness function, also called the objective function, measures the performance of a particle for the purpose of comparison with the local and global optimum. To serve the objective PSO engine for adjusting the values of arrival rate λ and window size ω, a particle is structured to hold required values for all stream tasks, and the particle dimension equals the number of stream tasks. Thus, the particle structure is workflow-dependent, with it is complexity derived from the number of stream tasks as well as the workflow structure complexity. For example, Table 3.2 shows how a 5-dimensional particle is structured for a workflow with five stream applications. Columns of arrival rate λ and window size ω values express the particle position in a random iteration. Moreover, a particle position is the representation of a workflow configuration plan which will feed the cost computation process to generate values that will be applied for the objective function.

Algorithm 1 provides the steps to find the optimal configuration plan for a given

TABLE 3.2: An example of particle position (a configuration plan)

Task Index i	**Arrival Rate** (λ') **(msg/s)**	**Window Size** (ω') **(s)**
2	4792	275
3	4250	344
7	3989	250
8	3685	1100
12	3700	1250

Algorithm 1 Finding an optimal workflow configuration plan

```
1: Load PSO Configuration
2: Initialize Particles P
3: P.pos ← NULL
4: P.gbest ← inf
5: for i ← 1, n do
6:     Randomize P_i.pos
7:     P_i.velcity ← 0
8:     P_i.cost ← callculteCost(P_i.pos)
9:     P_i.lbest ← P_i.cost
10: for j ← 1, m do
11:     for i ← 1, n do
12:         Update P_i.velcity
13:         P_i.pos ← P_i.pos + P_i.velcity
14:         P_i.cost ← callculteCost(P_i.pos)
15:         if P_i.cost ≤ P_i.lcost then
16:             P_i.lbest ← P_i.cost
17:             if P_i.lbest ≤ P.gbest then
18:                 P.gbest ← P_i.lbest
19:                 P.pos ← P_i.pos
20: if P.gbest == inf then
21:     return NULL
22: else
23:     return P.pos
```

hybrid workflow. The PSO-based technique has two main steps: it generates solutions (configuration plans), and evaluates the performance of these solutions. A particle position refers to a workflow configuration plan. The algorithm performs m x n iterations to find the global best configuration which has minimum cost, where m is number of dimensions and n is number of particles. At each iteration, the particle P_i's position is updated based on its current position and velocity. Next, in line 14, the new position cost is evaluated by calling the cost evaluator in Algorithm 2. The code on lines 14-19 examines the new cost against local and global minimums, and updates them accordingly. Finally, the workflow configuration with minimum cost is returned.

Algorithm 2 shows the computation procedure to calculate the optimization value of a configuration plan, which is represented as a PSO particle position. To evaluate the performance for a given solution, calculate end-to-end workflow execution time E and total number of cores R are calculated as follows.

Algorithm 2 Calculating the optimization value of a configuration plan

```
1:  procedure CALLCULTECOST(P , T)
2:      R ← 0
3:      E ← 0
4:      for i ← 1, n do
5:          if T_i.type == STREAM then
6:              T_i.R ← findStreamC(P_i.λ)
7:              T_i.E ← P_i.ω
8:          else
9:              T_i.R, T_i.E ← ProcessBatchTask(T_i)
10:
11:         R ← R + T_i.R
12:     E ← findWorflowCP(T)
13:     if E ≠ NULL then
14:         return E ∗ R
15:     else
16:         return inf
```

In line 6, number of servers c_i for a stream task is calculated based on the proposed queuing system. The number represents the level of parallelism, because each level is equivalent to a single-core machine. The number of servers c_i is derived from Equation 3.5.

$$c = \frac{\lambda'}{\mu\rho} \tag{3.10}$$

Furthermore, it is assumed that execution time for a stream task is equivalent to the length of aggregation window ω.

For a batch task, line 9, number of resources R and execution time E are computed based on the deterministic model in Algorithm 3, which shows execution time and cost calculations upon incoming computation load. According to the hybrid model, batch tasks can receive data from stream and batch tasks. The algorithm starts by calculating the total loaded data, lines 4-6, then estimates the total time needed to process the data according to task execution configuration. The last step is to incrementally adding resources until the deadline constraint ϑ, lines 8-10, is satisfied.

Algorithm 3 Process batch task T_i to find number of servers and execution time

```
1:  procedure ProcessBatchTask(T_i)
2:      R ← 0
3:      E ← 0
4:      totalSLoad ← ComputePredecessorStreamLoad()
5:      totalBLoad ← ComputePredecessorBatchLoad()
6:      totalLoad ← totalSLoad + totalBLoad
7:      totalBTime ← ComputeTotalProcessingTime()
8:      while totalBTime ≥ ϑ do
9:          C ← C + 1
10:         R ← updateET()
11:     return C, R
```

Back to Algorithm 2, After finding the number of servers and execution time for each task, workflow returns to Algorithm 2, which finds the longest workflow execution time or critical path CP.

3.4.3 Execution Time Estimator

The resource estimator is the framework component responsible for estimating the maximum execution of a hybrid workflow based on a given workflow configuration and by aggregating batch tasks in groups. Each group incorporates a set of independent tasks which can be executed concurrently. The main feature of grouping tasks is constructing computation periods that allow accurate resource estimation without violating the execution deadline constraint. Stream tasks have different behaviour because they need continuous processing. Thus, they cannot be allocated within batch groups. Instead, they are placed in a separate group for resource estimation. Correspondingly, the stream arrival rate contributes to group formulation, thereby measuring the size of data passed to the next tasks.

The objective of the group-based resource estimation is to find the required number of cores and maximum execution time for each task group based on the variation of stream processing parameters of aggregation window size and arrival rate. This formulates the behaviour of hybrid workflow with regard to tuning stream processing proprieties in the resource estimation and scheduling process.

Algorithm 4 Workflow Tasks Grouping

```
1: procedure CreateWorkflowGroups(T)
2:     G = {}
3:     for each t ∈ T do
4:         CallculateExecutionTime(t)
5:         g = FindClosestGroupWithRelaxation(t, G)
6:         if g == NULL then
7:             AddNewGroup(t,G)
8:         else
9:             UpdateGroup(t,g)
10:    R, E = EstimateCoresAndTime(G)
11:    return R, E
```

Algorithm 4 presents the main steps of the group-based workflow estimation process. The algorithm aims to assign workflow tasks and estimate resource and execution time for each group. Firstly, for each task, the algorithm calls the function The *CallculateExecutionTime* (line 4) to calculate the execution time, EST (Earliest Start Time) and LFT (Latest Finish Time). Based on the dependency structure for the workflow. Considering these values, the *FindClosestGroupWithRelaxation* function (line 5) will try to fit the task into an existing group. Moreover, it is assumed that adding a relaxation percentage (less than 5%) to the leading task in a group will alleviate the grouping strategy for complex workflows, If the algorithm fails in fitting that task, a new group will be created. Next, the algorithm calls the *EstimateCoresAndTime* to calculate the number of cores and execution length for each group.

For stream tasks, the number of tasks is estimated using the adopted queuing system (Equation 3.5), and for batch tasks, the number of cores is calculated using a deterministic model (Equation 3.8). Finally, the algorithm returns the cumulative values for the number of cores as (R) and the execution time as (E) for all constructed groups (G). The next section describes the resource provisioning process for workflow groups in a cloud system.

3.4.4 Hybrid-Workflow Resource Provisioning Optimizer

The optimizer aims to control the execution of hybrid workflows in order to optimize the combination of monetary cost C and execution time T. It is assumed the two objectives are equally significant to the overall optimization decision. A configuration plan is evaluated based on the following optimization function:

$$min(C.T) \tag{3.11}$$

Subject to:

$$\text{c1: } E_{t_i{}^B} < \vartheta_{t_i{}^B} \text{ , } \forall t_i{}^B \in T$$

$$\text{c2: } \tau_{t_i} > \tau_{t_D} \text{ , } \forall t_i{}^S \in T$$

Where:

E_{t_i} = Batch Task execution time

ϑ_{t_i} = Batch Task deadline

τ_{t_i} = Task throughput

τ_{t_D} = Application-defined throughput

$c1$ = All batch tasks achieve the deadline constraint

$c1$ = All tasks accomplish the minimum level of processing throughput

To calculate the workflow execution cost C and time T, a group-based provisioning technique was adopted to allocate required resources R_i for each group G_i on a cloud system. The group-based technique works by finding the provisioning plan with minimum cost. Algorithm 5 presents high-level steps for provisioning workflow groups on a cloud system. The algorithm receives inputs of task groups G as a result of the estimation process, and cloud VM configurations V. In each iteration, lines 4-11, the algorithm searches for the group with lowest execution time and provisioning cost. The function *provisionGroupWithVMConfig* finds the number of cloud VMs under the configuration v which satisfies resource demand of group g. Moreover, in lines 9-11, the cumulative values of time T and cost C are updated, and provisioned group g is excluded from groups list G. The process ends after allocating all groups.

3.5 Performance evaluation

This section presents the experimental setup and results of the performance evaluation for the proposed framework.

3.5.1 Experimental setup

Shukla et al. [1] provide valuable benchmarking results for standard IoT application dataflows for both stream and batch applications. In addition, they discussed the standard structure of these applications by identifying the dependencies between stream and batch tasks. Figure 3.3 shows four IoT-based dataflow application tasks. Extracting data from streams and performing predictions are stream tasks, whereas training machine learning models and showing statistical results are more related to batch processing. Shukla et al. [1] evaluated each application type based on the micro-benchmarking for sub-tasks, and by running each sub-task in a single-core machine. We used Shukla et al.'s benchmarking results to define initial models parameters as well as build workflows for experiments. Three workflows were constructed: small, medium horizontal scale, and large vertical-scale. Workflow scalability refers to the structure complexity in terms of workflow depth (number of service layers) and number of tasks at each layer. Workflow execution time should be more sensitive to vertical scalability because more service layers are added to the workflow service chain. On the other hand, horizontal scalability involves expanding the workflow by adding more input data stream inputs and running more services in highly decomposed and flatten manner. This makes a greater contribution to the total monetary cost refers to the additional computation overhead caused by service layer expansion (adding more tasks). Table 3.3 presents the main characteristics of these workflows.

CloudSim [157] was extended to support the execution of workflows. A single data centre and three VM types were modelled to simulate a cloud resource provider. The VM types have the same configurations of Amazon EC2. Table 3.4 shows the

Algorithm 5 Group-based Resource provisioning on cloud computing environment

```
procedure HWRPO(G, V)
    T = C = T_temp = C_temp = T_min = C_min = ∞
    while G ≠ ∅ do
        for each g ∈ G do
            for each v ∈ V do
                T_temp, C_temp = provisionGroupWithVMConfig(g, v)
                if T_temp * C_temp < T_min * C_min then
                    T_min, C_min, = T_temp, C_temp
            T = T + T_min
            C = C + C_min
            G = G − g
```

details about VM configurations. As dynamic provisioning is adopted, the provisioner can benefit from the variety of VM configuration to generate an optimized resource provisioning setup. Thus, the VM type has the significant contribution on provisioner plan, instead of number of instances. T2 instances are used for their burstable performance as they can provide a baseline level of high sustainable CPU performance as long as a workload needs it. Moreover, t2 instances are suitable for general-purpose workloads, which are adopted on this experiment [156].

The performance of the proposed Hybrid Workflow Resource Provisioning Optimizer (HWRPO) was compared with other techniques, namely, full mode and random selection. The full mode technique implies running the workflow with a

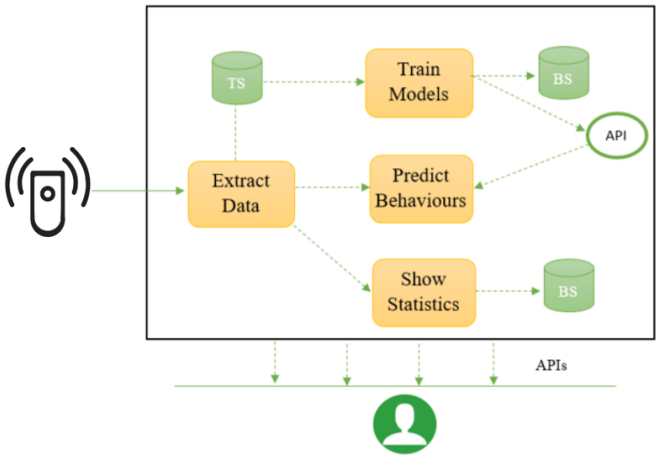

FIGURE 3.3: High level stream and batch applications dependencies for data analytics workflows [1]

TABLE 3.3: Workflows Characteristics

Workflow	**#Stream Tasks**	**#Batch Tasks**	**Scale Mode**
Small	3	5	Equal
Medium	15	15	Horizontal
Large	25	42	Vertical

TABLE 3.4: Types of VMs used in performance evaluation

Name	**CPU capacity (MIPS)**	**Price per hour**
t2.large	2	$0.0928
t2.xlarge	4	$0.1856
t2.2xlarge	8	$0.3712

semi-optimization process at queuing system utilization level. The random selection technique is a provisioning technique without optimization efforts at any framework execution level, either on queuing system utilization or task clustering.

3.5.2 Results and discussions

This section discusses the results of applying the resource estimation and provisioning framework on three different hybrid workflows. We evaluated the association between the variation in model parameters (window size, arrival rate, and throughput) workflow scalability (vertical and horizontal), and optimization parameters (execution time and monetary cost). For each model parameter, simulation was carried out 30 times, and average values were used for comparing the performance of full mode, HWRPO, and random selection techniques. model parameter values were varied between 25% and 175%. Relative percentages were proposed due to the variation in parameter values among workflow tasks.

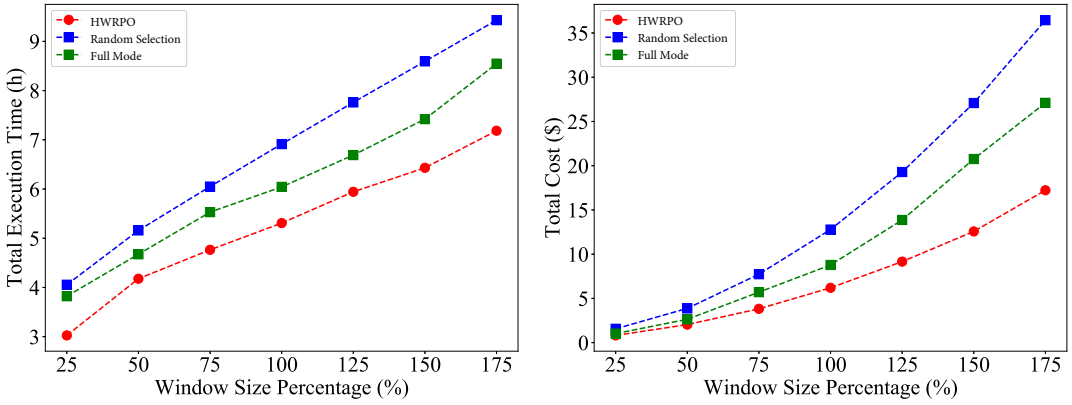

FIGURE 3.4: Window size variation impact on the small workflow

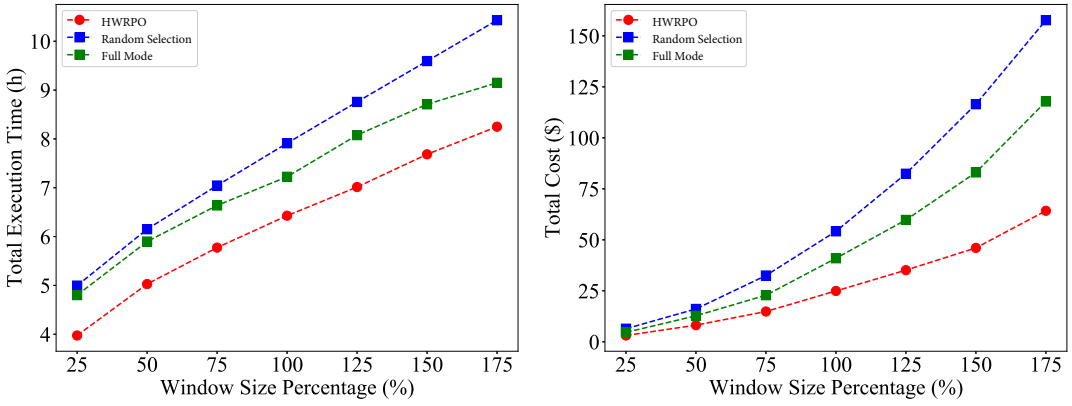

FIGURE 3.5: Window size variation impact on the medium workflow

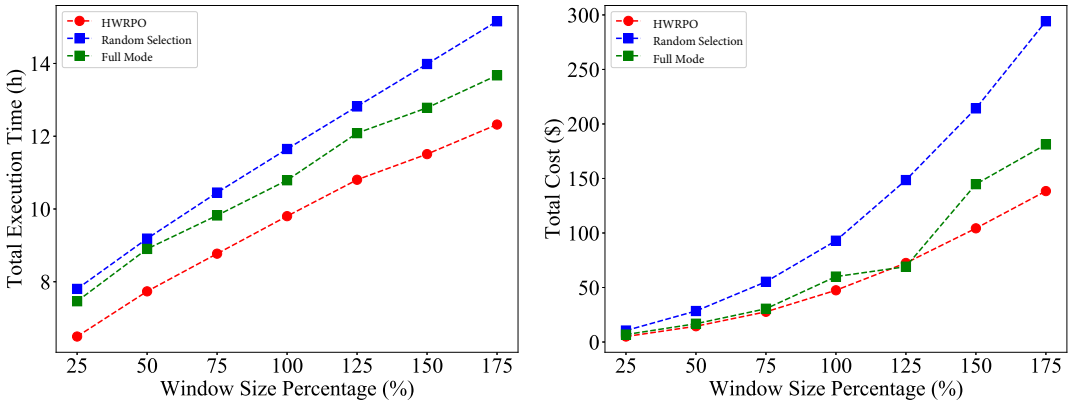

FIGURE 3.6: Window size variation impact on the large workflow

3.5.2.1 Window size

Window size is the time needed for processing and aggregating the incoming data stream. Figures 3.4, 3.5 and 3.6 show the results of varying window size on execution time and monetary cost. HWRPO demonstrates high stability in controlling the workflow execution time with increasing window size. Figures 3.5 and 3.6 show that HWRPO produces a slight difference in execution time optimization between horizontal scalability and vertical scalability.

With horizontal scalability, HWRPO was able to reduce cost linearly with increase in window size, with about an 80% reduction compared to the full-Mode technique. This is due to ability to HWRPO's cluster/group a higher number of tasks/applications and perform periodic resource provisioning more effectively than vertical scalability, which results in a 30% maximum cost reduction. This allows IoT applications to effectively produce more data either by adding more IoT devices (horizontal scaling) or dividing the incoming data load on applications (vertical scaling).

3.5.2.2 Arrival Rate

Results from Figures 3.7, 3.8 and 3.9 show that HWRPO exerts steady control over the incoming arrival rate under the constraints of the queuing system and throughput. For all simulated workflow structures, HWRPO reduced execution time compared with other techniques. However, Figure 3.7 shows less efficiency in execution reduction with 14% on average, compared to 25% in Figure 3.8. This can be explained by HWRPO's high efficiency in cost reduction, with averages of 60% and 40% for vertical and horizontal scalability respectively. In fact the impact of arrival rate on monetary cost is strongly related to the throughput constraint. As the constraint is relaxed, the algorithm will try to drop as many of the incoming stream messages as possible. This scenario is convenient for IoT application models, where performance is not aligned with generated data volume (e.g. in modeling of rare phenomena such as earthquakes and floods).

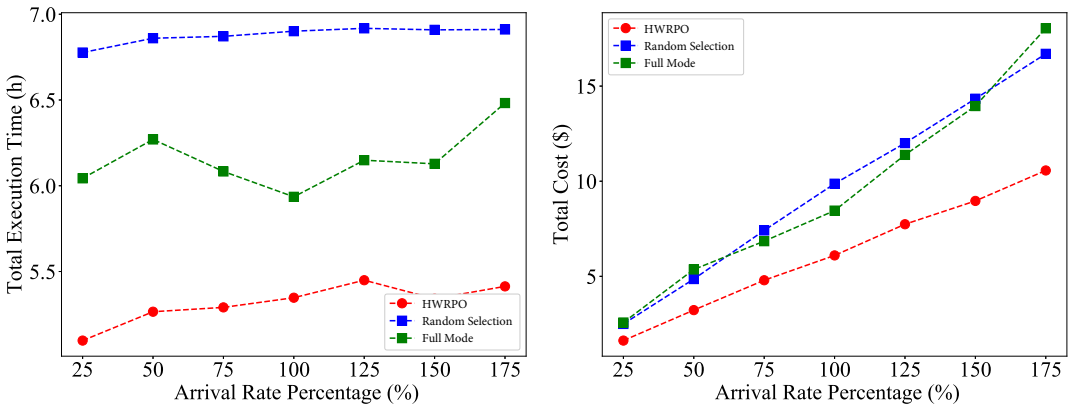

FIGURE 3.7: Arrival rate variation impact on the Small workflow

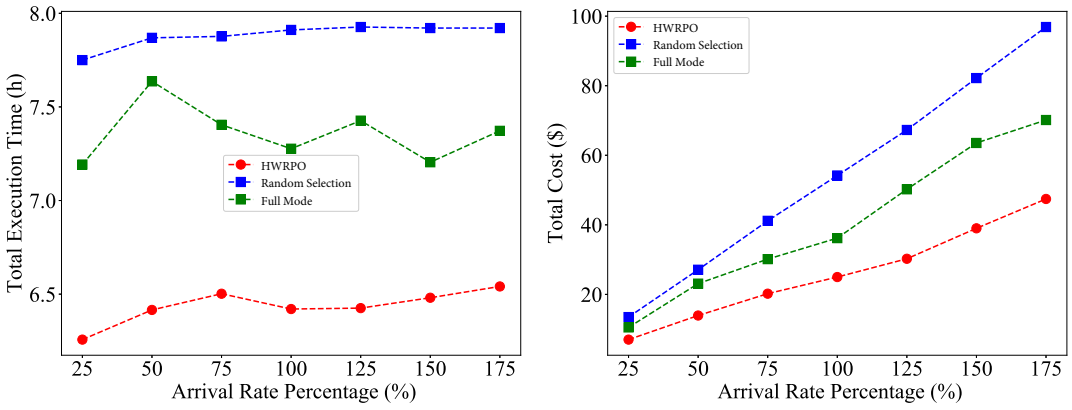

FIGURE 3.8: Arrival rate variation impact on the Medium workflow

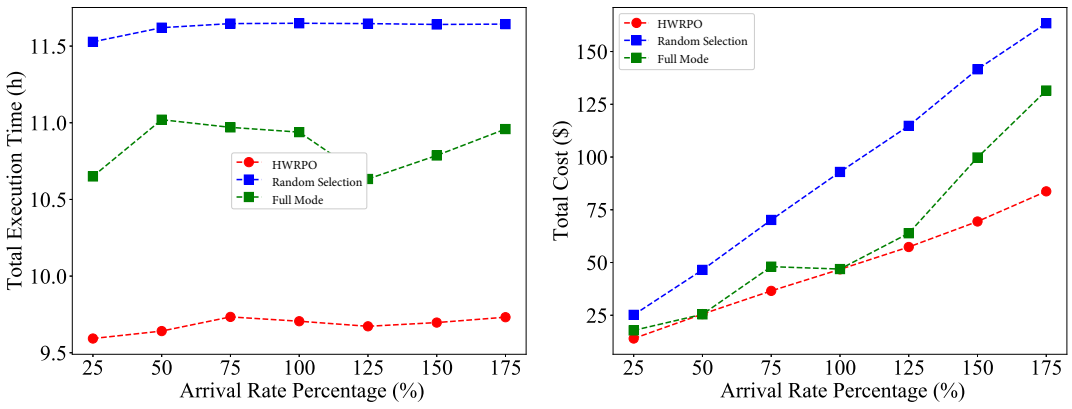

FIGURE 3.9: Arrival rate variation impact on the Large workflow

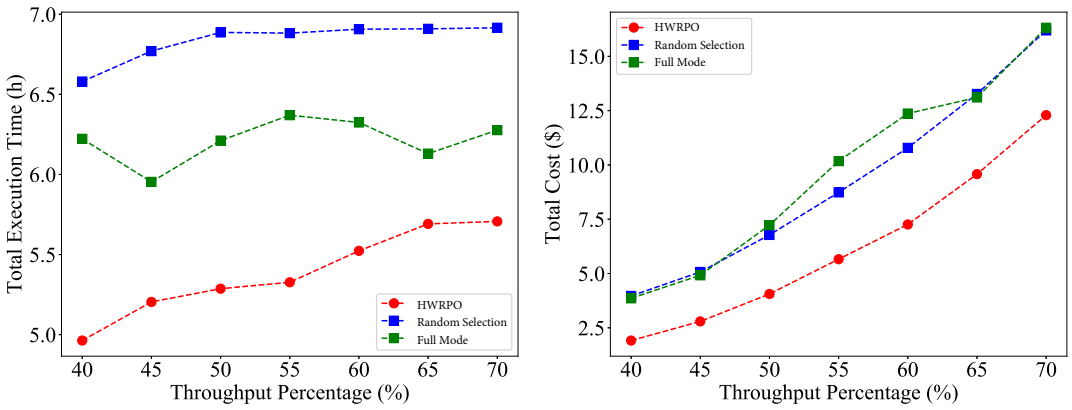

FIGURE 3.10: Throughput variation impact on the Small workflow

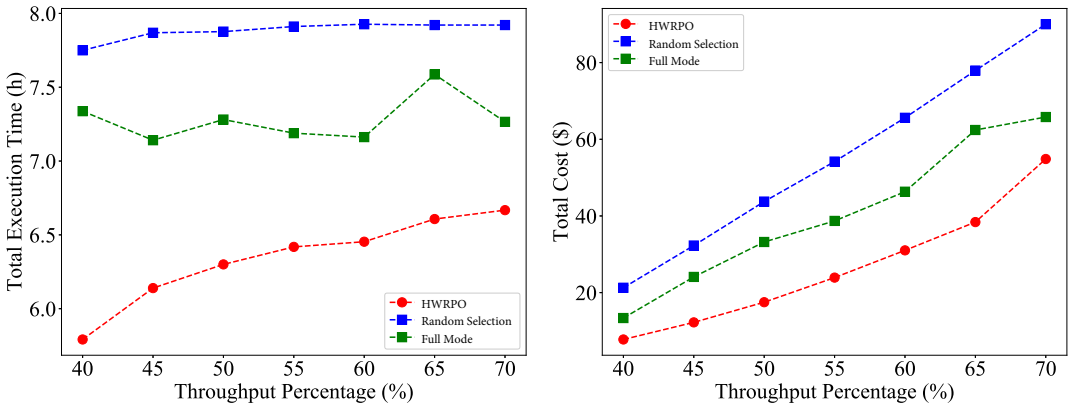

FIGURE 3.11: Throughput variation impact on the Medium workflow

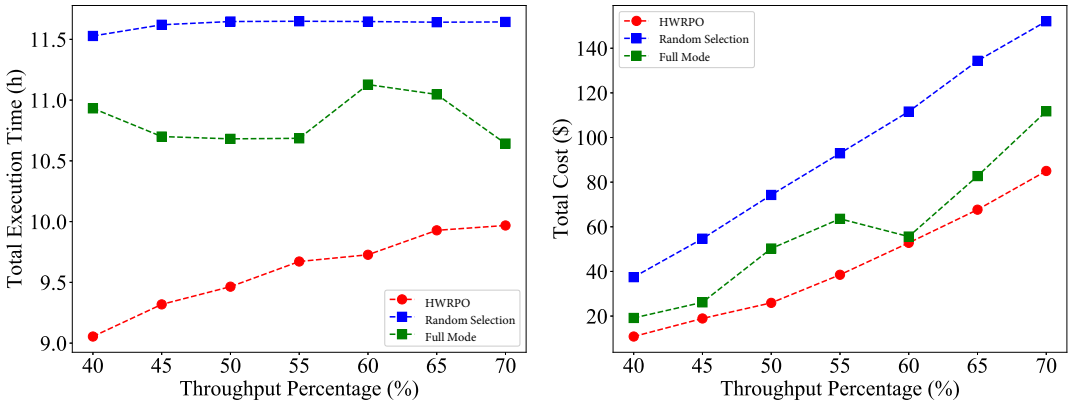

FIGURE 3.12: Throughput variation impact on the Large workflow

3.5.2.3 Throughput

The throughput constraint was varied in the range 40-70%. Figures 3.10, 3.11 and 3.12 show that monetary cost increases exponentially while throughput is moving

to its peak value. In complex workflows, shown in Figures 3.11 and 3.12, the increase in throughput value drives HWRPO to add more resources to allow the processing of more data. For example, in the vertical scaling example (Figure 3.12), cost increases by 400% when moving from 40% to 70% throughput. Setting up the throughput constraint is application and performance dependent; advanced heuristics algorithms and tuning techniques can produce significant cost reduction with complex and long-term hybrid workflow execution.

3.6 Summary

This chapter presented a resource estimation and provisioning framework, HWRPO, for hybrid workflow, which aims to generate an optimized workflow configuration plan in which the workflow execution time and monetary cost can be reduced. We developed a simulation environment to evaluate the influence of resource provisioning model parameters, namely, arrival rate, window size and throughput, on the optimization objective. In addition, the framework was applied to multiple workflow structures with various scalability factors.

The HWRPO technique was compared with baseline (random selection) and full mode techniques. Overall, HWRPO demonstrates promising execution time and cost reduction with most parameter variations for the three workflow cases. Window size has a powerful influence on both execution time and cost. The results showed that HWRPO was able to control execution time and cost through the efficient adoption of task clustering and periodic resource provisioning techniques. In addition, the results showed a correlation between arrival rate and throughput. IoT-based workflows can reduce cost efficiently with minimized throughput constraints. Furthermore, the analysis highlighted the sensitivity of the optimization objective to the throughput constraint, and the necessity of building efficient tuning techniques to guarantee a reasonable margin of workflow execution optimization.

For large-scale workflows, adopting a PSO technique to search the space with massive configuration plans can increase time complexity and affect overall framework performance. In addition, the work described in this chapter did not consider the transmission network quality for migrating stream data to the cloud. The next chapter presents more advanced optimization methods which can support resource provisioning and scheduling for complex hybrid workflows in an edge cloud system.

Chapter 4

Hybrid Workflow Scheduling on Edge Cloud Computing

In chapter 3, a resource provisioning framework for hybrid workflows on cloud systems was proposed. Two limitations were observed. The former is related to search time complexity for large-scale workflows, while the latter is related to handling data migration latency to the cloud computing layer. This chapter proposes two-stage hybrid workflow scheduling framework for edge cloud computing. In the first stage, a resource estimation algorithm based on a linear optimization approach is proposed, and in the second stage a cluster-based provisioning and scheduling technique for hybrid workflows on heterogeneous edge cloud resources is proposed. This chapter also provides a multi-objective optimization model for execution time and monetary cost under constraints of deadline and throughput. The results demonstrate the framework's excellent performance in controlling the execution of hybrid workflows by efficient tuning for stream processing parameters, such as arrival rate and processing throughput. Under working constraints, the proposed scheduler provides significant improvements for large hybrid workflows in terms of execution time and monetary cost, an average of 8% and 35% respectively.

4.1 Introduction

Cloud and edge computing are likely to be the core techniques of future computing facilities and are already adopted in most data processing scenarios [158]. While cloud computing offers a robust and scalable computation model, a continuous data stream from a large number of IoT devices creates bottlenecks with respect to latency and cost constraints. Cloud computing is not the ideal computing system for latency-sensitive applications, such as real-time gaming, augmented reality and real-time streaming [18]. Because cloud resources are located closer to the core network, the round-trip latency of these application will be high due to passing data through multiple gateways. Moreover, transferring large amounts of data with latency-sensitive constraints to such a centralized environment is insignificant in terms of resource utilization, communication latency and the energy consumption of computation servers [19]. On the other hand, processing streams in nearby resources, at the edge layer, promises to avoid cloud limitations [20]. Edge computing refers to technologies that permit the computation at the network edges, and act as an intermediate layer to transmit streams to cloud services on behalf of IoT devices [20]. However, a high percentage of this data is temporary and only a small amount might contain meaningful data. Edge computing has a significant role in processing massive data and only uploads processed data to clouds. For instance, an autonomous vehicle requires a huge amount of data, in forms of images and video, to be processed in real-time to produce good driving decisions. However, the availability and low efficiency of edge servers are decisive factors in quality achievement [121]. Hybrid workflow specifications require the development of efficient resource provisioning and scheduling techniques which coordinate the execution of hybrid workflows on edge cloud systems.

In this chapter, we proposed a hybrid workflow scheduling framework for edge cloud computing which considers the integration requirements of hybrid workflows while optimizing the execution time and monetary cost. The framework involves algorithms for resource estimation, provisioning and task scheduling. The terms

task and application are used interchangeably in this work. This chapter makes the following contributions.

- A hybrid workflow resource estimation algorithm based on the gradient descent search (GDS) technique. The algorithm reduces the workflow execution time and the number of computation units based on the characteristics of stream and batch applications and dependencies between them.
- A cluster-based resource provisioning and scheduling algorithm for a hybrid workflow in an edge cloud computing environment. The algorithm optimizes overall workflow execution time and cost with respect to data communication between workflow tasks.
- A comprehensive performance analysis of estimation and scheduling algorithms, including scheduling adaptability, edge capability and optimization time.

The rest of the chapter is structured as follows. Section 4.2 discusses related work on workflow resource estimation, provisioning, and task scheduling. A detailed description of resource and application modelling is provided in Section 4.3. Section 4.4 provides a detailed description of the hybrid workflow estimation and scheduling. Section 4.5 presents the experimental findings and provides insights into the main results. Conclusions and suggestions for future work are provided in Section 4.6.

4.2 Related Work

Workflow scheduling became one of the major problems in cloud computing due to the rapid growth of on-demand requests and the heterogeneous nature of cloud resources [159]. The literature contains extensive research on provisioning and scheduling in the context of grid and cloud computing, involving various optimization techniques and QoS constraints [45, 46, 160]. Most existing techniques

have contributed to the optimization of workflow execution to achieve objectives of time, cost, energy, scalability, and reliability in a single or multi-cloud computing environment. However, few researchers have proposed models which are efficient and relevant to hybrid workflows.

Hybrid workflows are integrated computation models of stream and batch data processing pipelines with varying QoS requirements [161]. Stream processing is latency-sensitive and subject to constraints like stream rate and throughput [39]. On the other hand, batch processing involves less time-sensitive processing, but with high corresponding to intensive computation such as data aggregation and model prediction. There is a large body of research body on the scope of computation-centric workflow scheduling [141–143], which is perfectly meet the requirements of batch processing in terms of computation complexity and data size.

Many techniques have been proposed to conquer the challenges of stream workflow scheduling [110]. As noted in earlier chapters, Zhou et al. [121] proposed an online gradient descent (OGD) technique to estimate the rate of data streams for applying dynamic resource provisioning in an edge environment. Ren et al. [48] concluded that shifting to real-time and context-aware IoT application provisioning on the edge cloud imposes a transparent computing architecture for IoT applications, which can take advantage of edge computing. Long et al. [124] proposed an edge computing framework for delay-sensitive multimedia processing application. These authors proposed a greedy algorithm to improve human detection accuracy under deadline constraint, but their framework only considers lightweight stream workloads. Zhang et al. [125] proposed a game-theoretic framework for cooperative task allocation for delay-sensitive social sensing applications in edge cloud systems. However, as outlined above, most stream workflow scheduling is latency-sensitive, and issues like handling large intermediate/temporary data and integration with data-intensive systems, mostly at batch level, are not considered.

There has been relatively little published work on problems associated with hybrid workflows, such as how the dependency and integration between stream and

batch affects resource provisioning and task scheduling techniques. Moreover, few researchers have explored the opportunities of combining edge and cloud computing for the purposes of hybrid workflow scheduling, or the challenges of this combination with respect to load balancing, edge capacity limitations and data communication between edge and cloud resources.

This chapter outlines an end-to-end hybrid workflow scheduling framework for an edge cloud computing environment. The framework considers the challenges of combining two computation paradigms, and provides solutions for resource estimation and provisioning and task allocation in an edge cloud computing environment.

4.3 Edge Cloud Computing System Model

In this chapter, we adopted an edge cloud computing model to meet the computation requirements of hybrid workflows. The edge cloud model is convenient for hybrid workflows, because the stream tasks with latency sensitivity can benefit from the availability of edge resources, whereas batch tasks with heavy workloads can be processed at powerful computation nodes in the public cloud. Figure 4.1 shows the adopted edge cloud system model.

This chapter concentrates on a three-layer resource model composed of IoT, edge and multi-cloud, as can be seen in Figure 4.1. The three layers are explained as follows.

- *Layer 1 (IoT).* The layer represents the user interaction layer, in which IoT devices (sensors, smartphones, relays, etc.) are responsible for collecting information and performing operations that involve the use of limited resources. IoT devices can send workloads to closer edge nodes or cloud VMs.
- *Layer 2 (edge layer):* This layer represents all devices on the path between the IoT layer and cloud layer. An edge node can be a non-stationary computation device, such as mobile devices or a Raspberry Pi, or a stationary

device, such as a personal desktop, company server or a cloudlet. In this thesis, it is assumed that edge devices are limited in computation and storage capabilities, but able to handle some pre-processing tasks on the stream processing pipeline. Overall, the edge layer offers computation close to data sources (IoT devices and sensors) to reduce data transfer time, and performs pre-processing in a timely manner constraint [20].

- *Layer 3 (multi-cloud services):* The third layer includes cloud VMs that have high computational and storage capabilities, and are able to handle heavy weight computations such as predictive analysis, machine learning, business intelligence, big data analytics, and complex visualization.

At the edge and cloud layers, the work assumes various machine configurations $R_i = \{VM_{i,1}, VM_{i,2}, ..., VM_{i,n}\}$, where n is the number of VM configurations

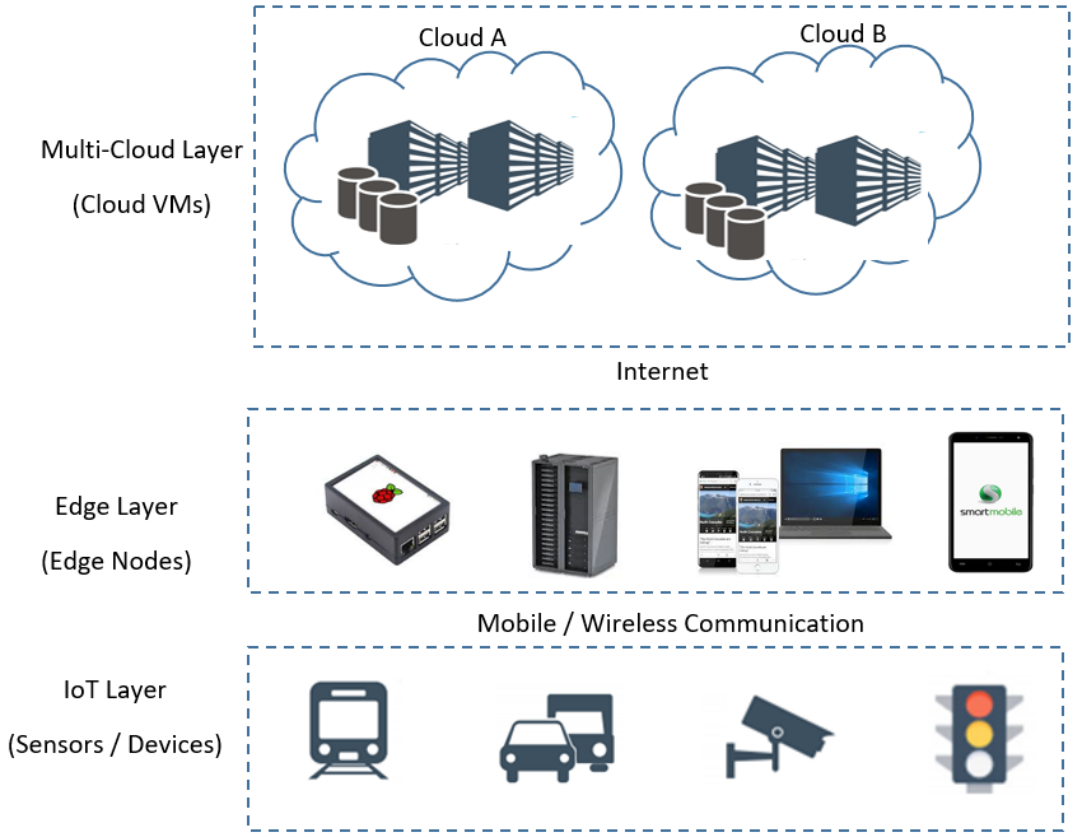

FIGURE 4.1: Edge cloud computing system model

in a data center, and a VM configuration refers to the VM characteristics of computation power, memory and processing cost.

4.4 Hybrid Workflow Scheduling in Edge Cloud Resources

Workflow scheduling is utilized to plan resource allocation for tasks based on the required QoS parameters, and it is responsible for selecting optimal virtual machines for workflow execution using a preferred scheduling algorithm. In this chapter, a hybrid workflow scheduling framework for edge cloud computing was proposed. The framework is designed to meet the requirements of hybrid workflow computation as a combination of stream and batch processing models. Moreover, offline scheduling is assumed because information about incoming jobs is known in advance [162]. Without loss of generality, Figure 4.2 shows the main framework steps for hybrid workflow resource estimation and scheduling.

The input workflow is an example of a hybrid workflow that shows the dependency flow between stream and batch tasks. Start and end tasks are dummy tasks and do not contribute to the hybrid workflow execution. The framework specifies a multi-level optimization technique for resource estimation and workflow scheduling. For resource estimation, a Constraint-based Gradient Descent search for Hybrid Workflows (C-GDHW) is proposed, which aims to minimize workflow execution time with the optimal number of processing cores by finding the optimal task grouping formulation. Next, for provisioning and scheduling, a cluster-based hybrid workflow provisioning and scheduling (C-HWPS) algorithm was adopted. It is based on an improved clustering technique and aims to optimize the execution of hybrid workflows in edge cloud computing in terms of execution time and monetary cost. The framework is explained in the rest of this section.

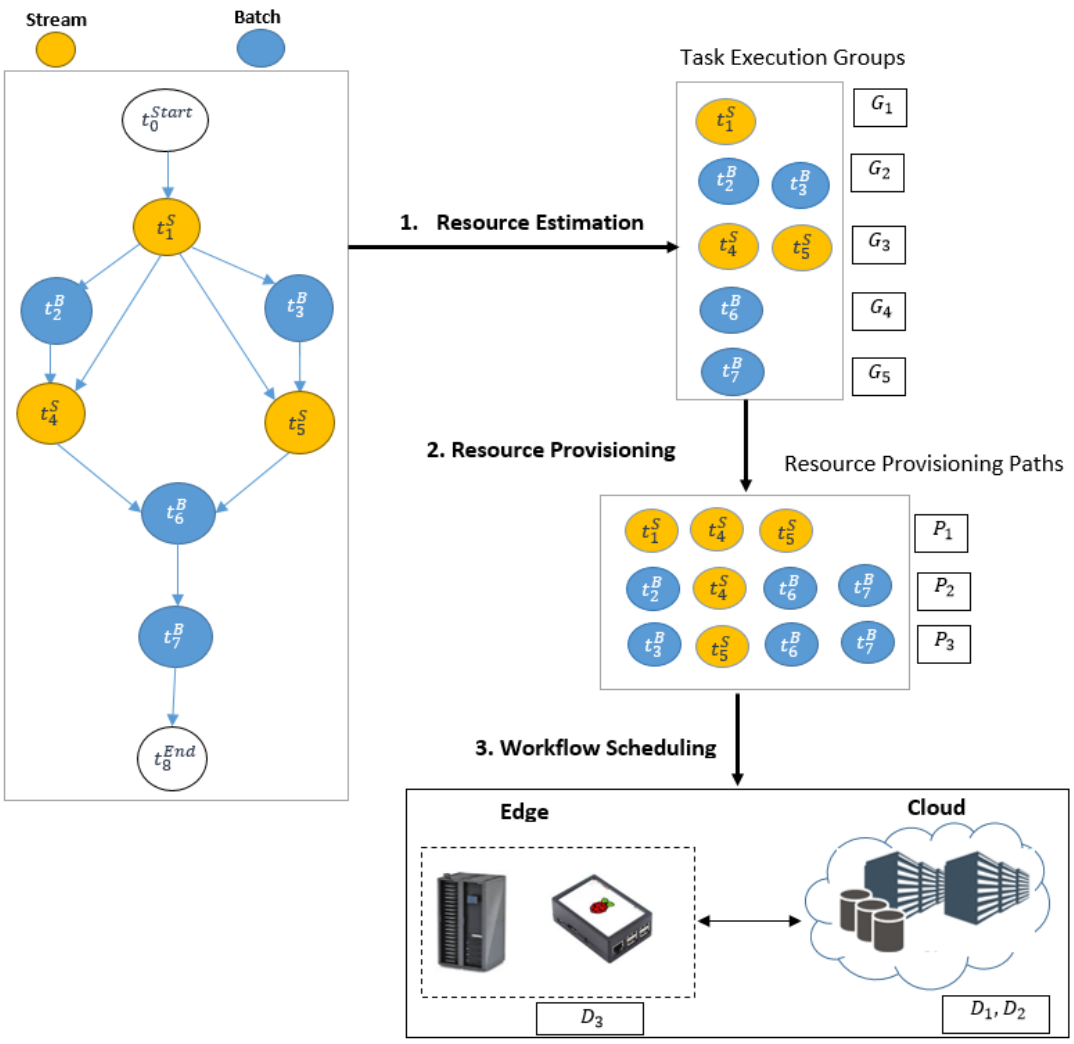

FIGURE 4.2: A high-level abstraction of hybrid workflow estimation and scheduling

4.4.1 Resource Estimation with Gradient Descent Search Approach

Resource estimation is a multi-objective problem of minimizing the total execution time E and number of computation units (cores) R. The desired outcome of the resource estimator is a group-based workflow execution plan, which can simplify provisioning and scheduling in an edge cloud computing environment, particularly for complex hybrid workflows. Thus, in the estimation process, the specific concern is to reduce the end-to-end execution time (critical path) E with the minimum number of computation cores among all execution periods R. The optimization

function for the resource estimation was modelled as:

$$\min(E.R) \tag{4.1}$$

4.4.1.1 Resource Estimation Problem Formulation

Gradient descent (GD) is an iterative optimization method to find a local minimum for linear and non-linear functions [163]. It is computationally easy and requires no memory, since the only input to its decision comes from the previous step. GDS was adopted in this work to find the optimal combination of problem input parameters (i.e., arrival rate λ , aggregation window size ω, and minimum workflow throughput τ) that minimizes the problem cost function under constraints of maximum batch task deadline ϑ , maximum stream processing waiting time in queue, and minimum workflow throughput τ. Therefore, the estimation process is a constrained GDS search problem.

Gradient descent is founded on the observation that if the multi-variable function $\mathbf{F(x)}$ is distinct and differentiable in a neighborhood of a point x_i, then $\mathbf{F(x)}$ decreases fastest if one goes from x_i in the direction of the negative gradient of $\mathbf{F}$, $\Delta F(x_{i-1})$ at x_i. The function position is calculated as follows:

$$x_i = x_{i-1} - \gamma_i \Delta F(x_{i-1}), n > 1 \tag{4.2}$$

where x_{i-1} is the current position, x_i is the next position, γ_i is the weight factor (or the step size), and $\Delta F(x_{i-1})$ is the direction of the steepest ascent. For minor and sufficient value of γ_i, $F(x_{i-1}) \geq F(x_i)$. The search problem starts with an initial guess $x(0)$ for a local minimum of $\mathbf{F}$, and considers the sequence $x_1, x_2, ...$ such that:

$$F(x_0) \geq F(x_1) \geq F(x_2)... \tag{4.3}$$

However, the value of step size γ can be changed with each iteration.

4.4.1.2 Resource Estimation Algorithm

The estimation algorithm includes a set of steps to find the optimized estimation cost (number of cores) with minimum execution time (Equation 4.1). Resource estimation implies searching for the best possible workflow configuration (in terms of stream arrival rate and aggregation window size) that ensures an optimized combination of execution time and the number of cores. The steps of exploiting the GDS algorithm for resource estimation are explained next. Figure 4.3 shows a visual presentation of the estimation process using a GDS technique.

The estimation process starts with an initial guess X_0 for the workflow execution configuration. A workflow configuration determines the execution behaviour of the hybrid workflow, which reflects how the change in stream task configuration (arrival rate, windows size and throughput) affects workflow execution time and cost based on the dependency between these tasks. Hybrid workflow modelling allows seamless cooperation between the two computing models. The estimation algorithm will iteratively produce a new stream configuration by calculating the function gradient value. In addition, the estimation process includes steps to validate and compute the optimality for each new configuration. A detailed description of the estimation algorithm C-GDHW is as follows.

Step 1: Initialize Problem The algorithm starts with an initial configuration input $x(0)$. The input is a $3 \times d$ matrix, where 3 refers to the number of problem parameters (λ, ω, τ) and d refers to the number of workflow stream tasks. The algorithm will iteratively update the solution (at each step) until it converges and no-cost improvement is progressed. At each iteration, the algorithm will compute the gradient and find the next position (configuration plan) x_i, see Equation 4.2.

Step 2: Validate Workflow Configuration

A workflow configuration is a set of values for the three workflow execution parameters, namely, stream arrival rate λ, aggregation window size ω, and workflow throughput τ. For each stream task ${t_i}^S$, find the number of servers which satisfies

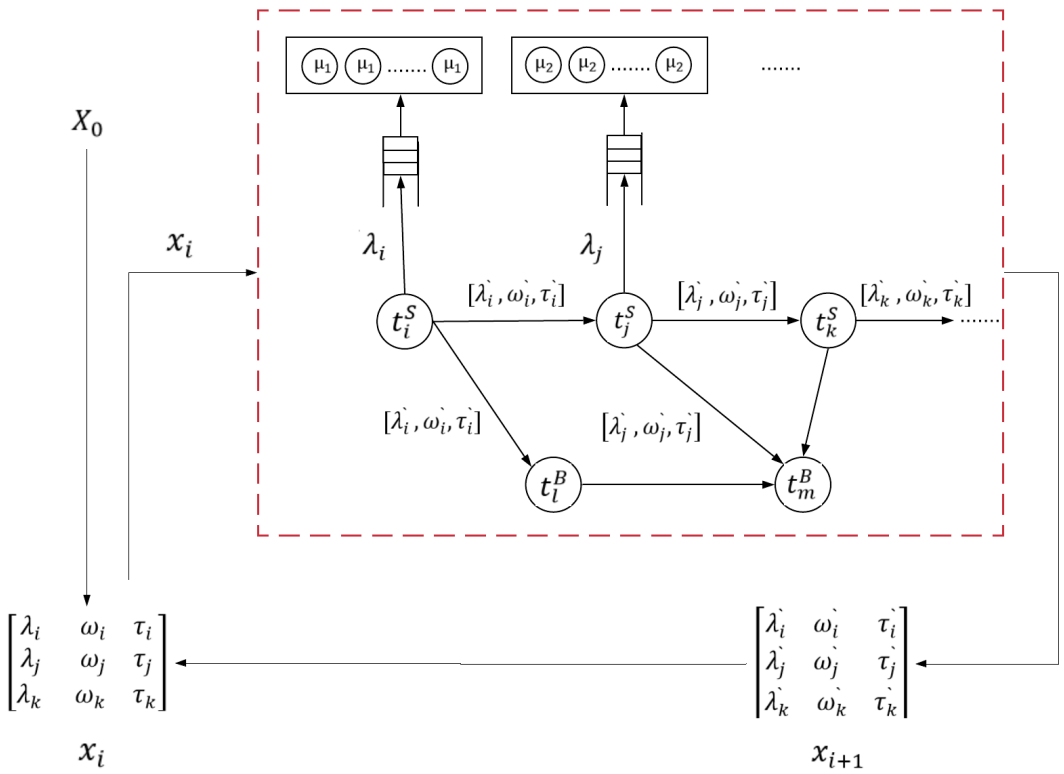

FIGURE 4.3: Resource estimation formulation with GDS technique

the following constraints:

$$\omega_i \geq \omega_i' \geq \frac{\rho.\mu_i.s_i}{\tau_i} \tag{4.4}$$

$$\lambda_{i-1} \geq \lambda_i' \geq \rho.\mu_i.s_i \tag{4.5}$$

where ω_i' and λ_i' are minimum values of window size and arrival time, respectively. However, these values can be any within the defined range. For each batch task ${t_i}^B$, find the number of servers which satisfies the following constraint:

$$\vartheta_i \geq E_i \geq \frac{ET_i}{s_i} \tag{4.6}$$

where ET_i and E_i are processing time, and task execution time, respectively.

Step 3: Formulate Task Groups and Calculate Estimation Values

section 3.4.3 provided a complete description of how to evaluate the cost of a workflow configuration in terms of execution time E and total number of required computation cores R based on task-grouping technique. Figure 4.2 shows an example of task grouping. Based on the input workflow, the estimator grouped

the tasks into five groups $\{G_1, G_2, G_3, G_4, G_5\}$. For instance, batch tasks ${t_2}^B$ and ${t_3}^B$ are in the same group, thus, the scheduler will handle them as a group for provisioning and scheduling purposes.

4.4.2 Hybrid Workflow Provisioning and Scheduling on Edge Cloud Computing Environment

In the estimation stage, the optimizer tries to group workflow tasks to minimize the overall workflow execution time with the least number of cores. However, it is worth noting that a core represents a VM with a single core use mostly for application benchmarking [1]. The next step is to allocate resources and schedule workflow tasks in the edge cloud computing environment. The process aims to optimize the workflow with respect to execution time T and monetary cost C. At this stage, the provisioner will consider both data transfer and computation while computing T and C. The hybrid workflow optimization in the edge cloud system needs to achieve the following objective function:

$$\min(T.C) \tag{4.7}$$

A C-HWPS algorithm was adopted to provision demanded resources and schedule the tasks in edge cloud computing environment. The clustering technique works by constructing execution paths from the starting task to the last task while considering task dependencies along any execution route. Figure 4.2 provides an example of the execution of three paths based on the input workflow. Three execution paths are constructed $\{P_1, P_2, P_3\}$. The first path includes all stream tasks, because continues stream processing is assumed. The other two paths show the execution sequence from the first batch to the last batch task. The provisioning and scheduling process is an iterative process over execution paths, such that, in each iteration, the scheduler decides on the next scheduling target path with minimum execution time T (processing and data transfer) and cost C (processing

Algorithm 6 Cluster-based Hybrid Workflow Provisioning and Scheduling

```
1:  procedure C-HWPS(G, D)
2:      P = A = {}
3:      T = C = T_temp = C_temp = T_min = C_min = ∞
4:      p_target = null
5:      S_target = null
6:      P = findExecutionPaths(G)
7:      while P ≠ ∅ do
8:          U = getUnScheduledPaths(P)
9:          for each p ∈ U do
10:             T_temp, C_temp, S = computePathInEC([p − A], D)
11:             if T_temp.C_temp < T_min.C_min then
12:                 T_min, C_min, = T_temp, C_temp
13:                 p_target, S_target = p, S
14:         schedulePathInEC([p − A], S_target)
15:         A = A + [p − A]
16:         P = P - p_target
17:         T = T + T_temp
18:         C = C + C_temp
19:         T_temp = C_temp = T_min = C_min = ∞
20:         p_target = null
21:     return A, T, C
```

and data transfer) on edge cloud resources. The processing will continue by adding paths and optimize time T and cost C.

Algorithm 6 presents the steps for resource provisioning and workflow scheduling. The algorithm receives inputs from task groups G as a result of the estimation process, and available computation resources D of clouds and edges. Infinite values for time T and cost C are assumed because a minimization problem is being approached. The scheduling algorithm can be expressed as follows. In line 6, the function *findExecutionPaths* constructs workflow execution paths P based on the tasks dependencies structure; each path should include all tasks needed to reach the end of workflow execution (i.e., the last task). Next, the algorithm will iterate over all execution paths to find the one with minimum execution time T and cost C on edge cloud system D.

The algorithm starts with the first unallocated path p which has at least one unscheduled task, in other words, $[p - A] \neq \emptyset$. Remaining tasks $[p - A]$, if any,

are computed in the edge cloud system D using the function *computePathInEC*, line 10, which is represented in Algorithm 7. The path with minimum time and cost p_{target} will be selected and scheduled based on the scheduling strategy S_{target}. Next, selected path p_{target} is removed from the list, and total time T and cost C are updated. Finally, the algorithm returns scheduled tasks A and optimization object variables T and C.

Algorithm 7 describes the steps of computing the optimized provisioning plan for an execution path p. No collaboration between edge and cloud resources is assumed, thus path tasks can only provision at one layer. The algorithm provides two main functions. The first is to schedule tasks in either edge or cloud, lines 4 and 7. Execution time and cost are calculated based on the available resource configuration. The second function *computeTransferTimeAndCost*, lines 5 and 8, calculates the amount of time and cost to migrate data considering the target provisioning environment (edge or cloud) and computing nodes which host already processioned dependent tasks. Time, cost and provisioning are returned. Accordingly, the capacity of the edge nodes as well as the quality of bandwidth between edge and cloud nodes have significant impact on provisioning and scheduling decisions.

4.5 Performance Evaluation

This section presents the performance evaluation of the proposed hybrid workflow scheduling techniques. It begins with a discussion of the experimental setup, including a hybrid workflow design from existing IoT application benchmarking, and edge cloud resource configuration setup. Then, the experimental results are discussed and insights from different analysis perspectives are provided.

Algorithm 7 Compute Execution Path in Edge Cloud System

```
1:  procedure COMPUTEPATHINEC(T, D)
2:      T = C = ∞
3:      S = null
4:      ET_e, EC_e, S_e = scheduleInEdge(T, D)
5:      TT_e, TC_e = computeTransferTimeAndCost()
6:      OptimCost_e = (ET_e + TT_e).(EC_e + TC_e)
7:      ET_c, EC_c, S_c = scheduleInCloud(T, D)
8:      TT_c, TC_c = computeTransferTimeAndCost()
9:      OptimCost_c = (ET_c + TT_c).(EC_c + TC_c)
10:     if OptimCost_e < OptimCost_c then
11:         T = ET_e + TT_e
12:         C = EC_e + TC_e
13:         S = S_e
14:     else
15:         T = ET_c + TT_c
16:         C = EC_c + TC_c
17:         S = S_c
18:     return T, C, S
```

4.5.1 Experimental Setup

For experimentation, we adopted a state-of-the-art IoT benchmarking tool to construct multiple hybrid workflow structures. Shukla et al. [1] proposed an IoT-based benchmark to formulate an application structure and dependencies between stream and batch tasks. They evaluated dataflow tasks by constructing them into sub-tasks and performed micro-benchmarking by running each sub-task in a single-core machine. In this thesis, we used Shukla et al.'s benchmarking results to define initial model parameters as well as build workflows for experiments. Three workflows were constructed: small, medium (horizontal-scale) and large (vertical-scale). A scalability factor was included to enable the study of the relationship between workflow structure and optimization parameters. Scalability means adding more stream tasks to allow more data stream inputs. In horizontal scaling, more batch tasks are added at each processing level, whereas in vertical scaling more processing levels are added to increase the workflow depth. Table 4.1 presents the main characteristics of these workflows.

For each model parameter, simulation was carried out 30 times, and average values

were used for comparing the performance of the C-GDHW, PSO-based and full-mode techniques. Model parameter values were varied in a range of 20% to 180% for windows size and arrival rate, and in a range of 10% to 90% for throughput. Relative percentages for the default workflow configuration were proposed due to the variation in parameter values among workflow tasks.

We extended CloudSim [157] simulator to reflect the adopted resource model as an edge cloud computing environment. Three data centers were added for two cloud providers and edge resource layer. At each data center, different VM configurations, including processing power and cost were assumed. VM configurations were applied for CloudA and CloudB, based on configurations provided by AWS and Microsoft Azure, respectively. Table 4.2 provides the resource model configuration setup for the experiments. We used two edge resource setups, high and low capability. The former was used for resource estimation evaluation, while the latter was used for edge capability analysis in the context of hybrid workflow computation. Furthermore, we considered a stable data transfer rate between edge and cloud servers as 50 MB/s, and between cloud servers as 100 MB/s.

In this chapter, the performance of the proposed gradient-based resource estimation technique, C-GDHW is evaluated in comparison to other two existing techniques, namely, PSO-based and full-mode. To achieve a fair and stable comparison, the cluster-based technique, C-HWPS, is applied for provisioning and scheduling hybrid workflows based on the outcomes of the estimation stage. The first existing technique is our previous work using the PSO technique for resource estimation [164]. PSO was utilized to search workflow configuration space to find an optimized solution. The second existing technique is the full-mode technique, in

Table 4.1: Hybrid Workflows Characteristics

Workflow	**#Stream Tasks**	**#Batch Tasks**	**Scale Mode**
Small	12	17	Equal
Medium	20	45	Horizontal
Large	36	73	Vertical

TABLE 4.2: Resource types for edge and cloud systems

Provider	Type	CPU Cores	Price per hour ($)
CloudA, CloudB	large	2	[0.093, 0.117]
	xlarge	4	[0.186, 0.232]
	2xlarge	8	[0.371, 0.465]
Edge (Low capability)	small	1	$0.002
	medium	2	$0.070
Edge (High capability)	large	4	$0.090
	xlarge	8	$0.103

which resource estimation is performed with a semi-optimization process at queuing system utilization level. In contrast to C-GDHW and PSO-based techniques, the full-mode technique does not apply arrival rate and window size reductions for stream tasks. Thus, resource estimation is performed to a hybrid workflow with maximum execution mode.

We proposed the semi-optimization model to evaluate the performance of the other two techniques in tuning stream execution parameters to optimize workflow execution time and cost. To the best of our knowledge, there is no model in the literature in the scope of hybrid workflow scheduling that is compatible with the proposed workload model or the proposed workflow structure.

Three experiments were conducted. The first one compared the performance of the C-GDHW estimation and scheduling technique with those of the nominated techniques (PSO-based and full-mode). The second experiment was designed to examine the stability of the proposed optimizer in maintaining workflow execution cost with workflow complexity regarding the percentages of windows size and arrival reductions. The third experiment sought to identify the contribution of edge resource capability to optimization objectives (time T and cost C).

4.5.2 Results and Discussions

This section presents and discusses the results of applying the resource estimation and scheduling framework to three different hybrid workflow structures.

4.5.2.1 Resource Estimation Evaluation

This section describes an evaluation of the efficiency of C-GDHW in optimizing hybrid workflow execution time T and cost C based on the variation in model parameters (stream window size ω, stream arrival rate λ, and execution throughput τ) and workflow scalability (vertical and horizontal). Next, each optimization case based on each parameter variation is discussed in turn.

The window size is the time needed for processing and aggregating the incoming data stream. Figures 4.4, 4.5 and 4.6 show the results of varying window size on execution time and monetary cost. The C-GDHW model demonstrates high stability and improved control over the workflow execution time with the increase in window size length for both horizontal and vertical scalability.

For large workflows, in Figure 4.6, the optimizer demonstrates potential cost saving for adding more resources to confront the increase in data processed by stream tasks in lengthy windows intervals. With the longest window interval, cost is reduced by reaches a maximum of 37% compared to the PSO technique. This is due to C-GDHW's ability to group a larger number of tasks and effectively perform periodic resource provisioning effectively. This feature allows IoT applications to produce more data by adding more IoT devices (horizontal scaling) or dividing the incoming data load on applications (vertical scaling).

The arrival rate refers to the number of stream inputs in a given time interval. Figures 4.7, 4.8 and 4.9 show the results of comparing the responsiveness of the three models to the variation in stream arrival rate. The C-GDHW optimizer determines steady control over the incoming arrival rate under the constraints of the queuing system and throughput. For all workflow scenarios, the optimizer was adequate to preserve the execution time at a steady level. In contrast, the PSO optimizer's behaviour shows fluctuation in optimizing the execution time, particularly with long-term execution workflow, as shown in Figure 4.9.

An additional advantage of the proposed optimizer is the linearity behaviour in provision more VMs (related to the monetary cost) to overcome high rate streams.

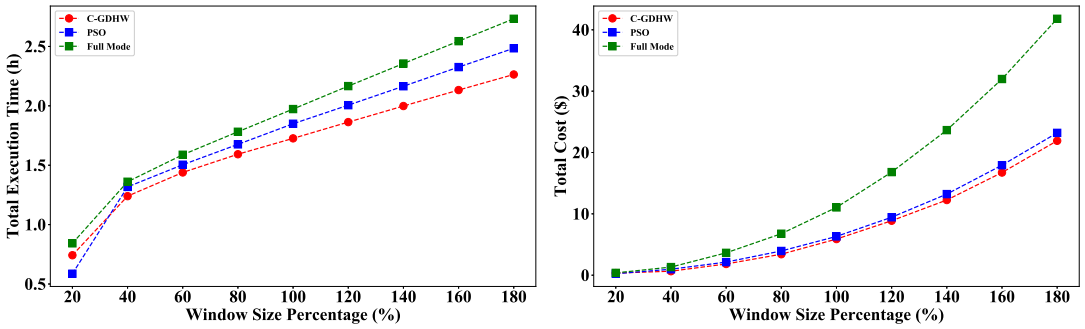

FIGURE 4.4: Window size variation impact on the small workflow

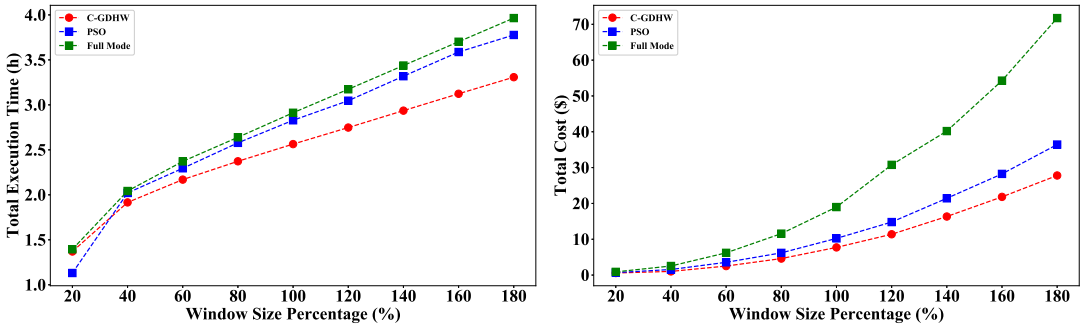

FIGURE 4.5: Window size variation impact on the medium workflow

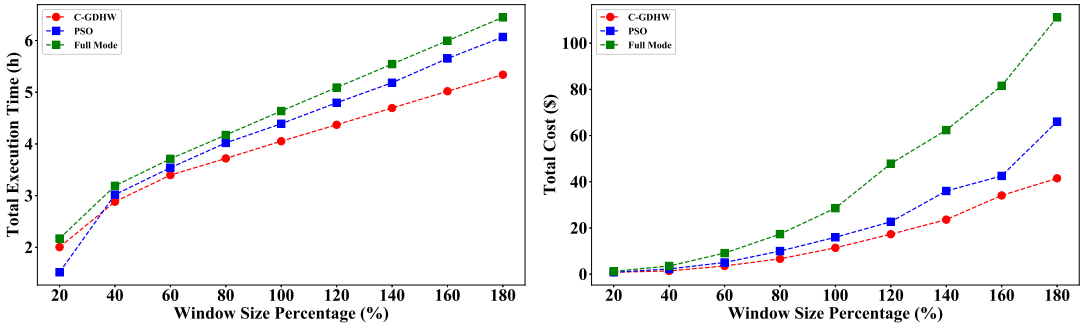

FIGURE 4.6: Window size variation impact on the large workflow

This behaviour can be interpreted with the efficiency to utilize the throughput relaxation to allocate stream tasks in nearby edge nodes.

One important conclusion is that the C-GDHW optimizer reduces cost as the number of stream tasks increases. This can be shown by comparing the performance of the proposed optimizer and the PSO technique. For small workflow with 12 stream tasks, Figure 4.7 shows that both optimizers demonstrate relatively similar cost-saving.

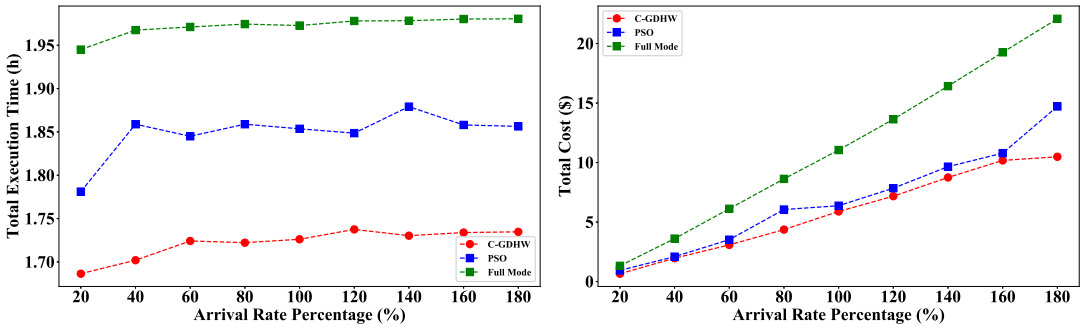

FIGURE 4.7: Arrival variation impact on the small workflow

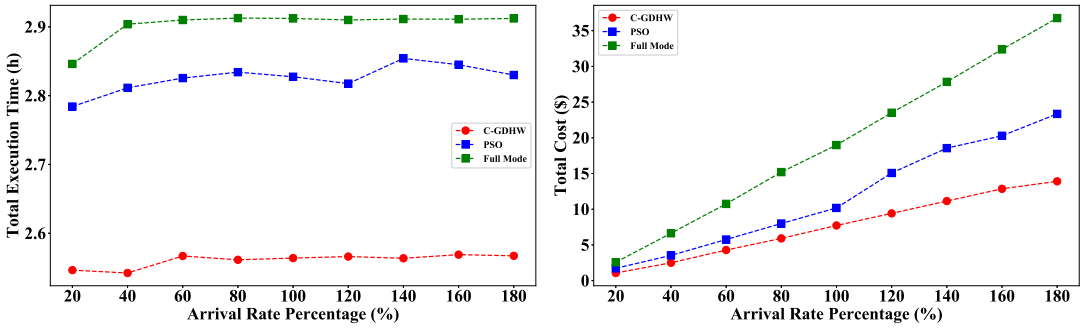

FIGURE 4.8: Arrival variation impact on the medium workflow

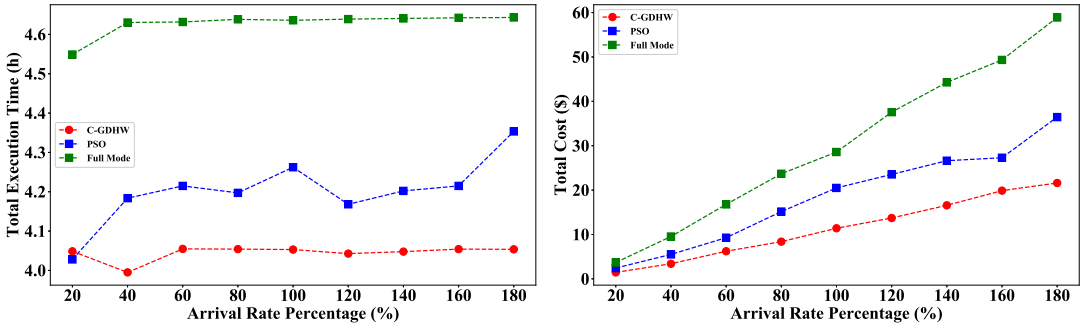

FIGURE 4.9: Arrival variation impact on the large workflow

In contrast, with 36 stream tasks for a large workflow in Figure 4.9, the proportional difference in cost reduction increase as the arrival rate rises. The reduction increased from 35% to 70% in the case of 40% and 180% arrival rate percentages respectively.

The Throughput determines the percentage of data to be processed to accomplish a workflow execution objective. This parameter is a real business-related selection, and differs from one application to another. In this work, we studied

the influence of application throughput on execution time and cost. The proposed optimizer was capable of reducing the influence of high throughput, particularly for the monetary cost parameter.

For instance, Figure 4.12 shows that with large workflow, the C-GDHW optimizer provides a near-linear execution time increase. Likewise, in the same figure, the optimizer produced a potential cost saving compared to PSO of 46% with the highest throughput of 90%. Hence, the proposed technique is able to handle the increase in throughput while reducing the time and cost. Some scientific workflows do not require collecting huge data for developing accurate predictive models. For instance, modeling of rare phenomena prediction such as Earthquakes and floods.

4.5.2.2 Adaptability Analysis

We studied the C-GDHW optimizer's performance in reducing stream task arrival rate and window size to optimize workflow execution cost. The reduction level has a direct impact on the workflow execution outcome from an application perspective. For example, in predictive analysis, the amount of data plays a vital role in training prediction models and producing highly accurate results. On the other hand, processing more data imposes an additional overhead in terms of computation cost. Thus, it is essential to provide a balanced estimation to handle the trade-off between execution performance and cost.

Figure 4.13 demonstrates the optimizer's adaptability in providing a stable arrival rate and window size reduction for a large hybrid workflow. For example, with variation in throughput, the optimizer to preserves a stable reduction for the arrival rate and linear reduction for the windows size in response to the exponential cost increase.

4.5.2.3 Edge Capability Analysis

Figure 4.14 shows the correlation between the capability of edge resources, and workflow execution time and cost with the variation in stream arrival rate. Edge

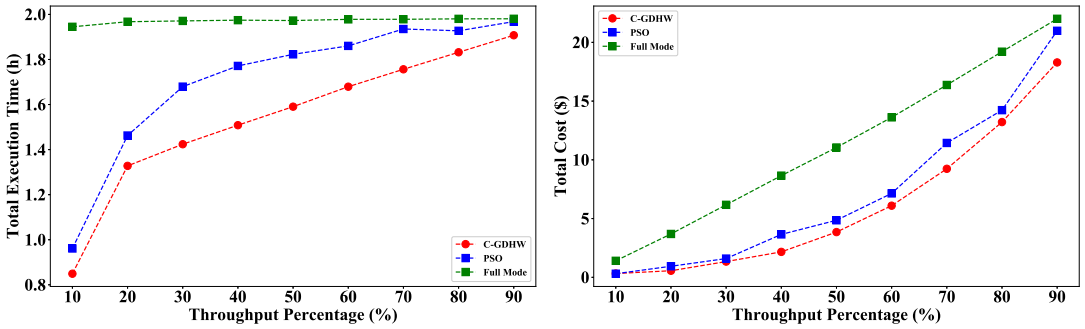

FIGURE 4.10: Throughput variation impact on the small workflow

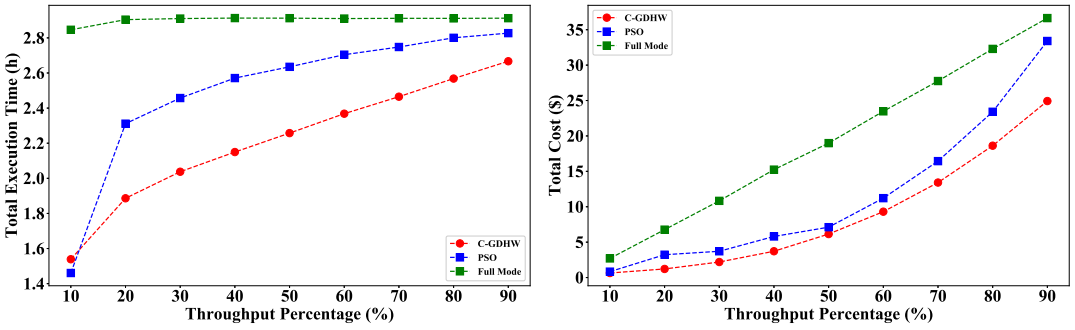

FIGURE 4.11: Throughput variation impact on the medium workflow

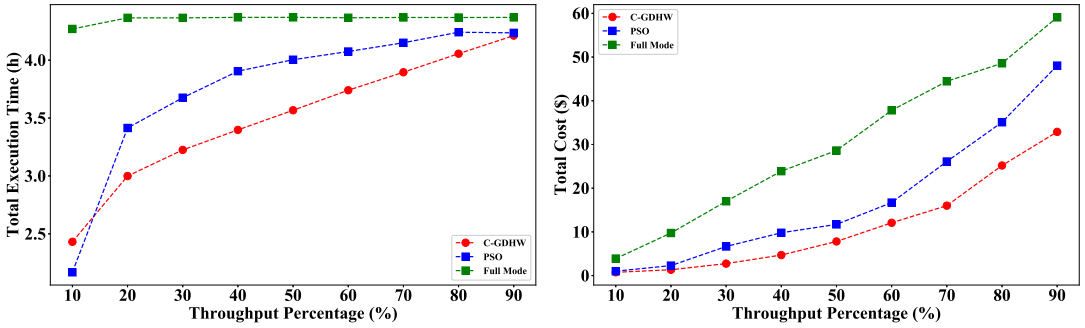

FIGURE 4.12: Throughput variation impact on the large workflow

capability indicates the computation performance of edge resources. Table 4.2 provides details about low and high edge resource configuration (low capability refers to the small and medium VM types, and high capability refers to large and xlarge VM types). Despite variation in stream arrival rate, results indicate a steady workflow execution time with low and high edge capability, with 15% average difference and minimal contribution of the workflow complexity.

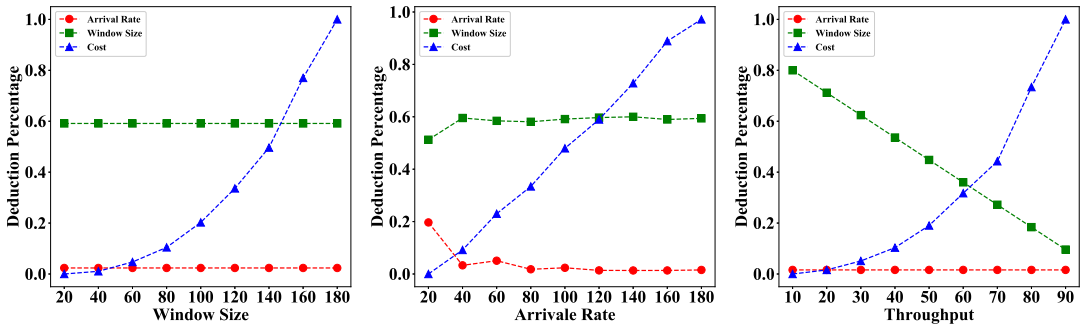

FIGURE 4.13: Arrival rate and window size reduction with cost increase: large workflow

On the other hand, the figure implies a linear increase in monetary cost, particularly for the large hybrid workflow scenario. The cost increases gradually from 56% (lowest arrival rate) to 8% (highest arrival rate). Overall, the graph shows clearly that as stream arrival rate increases, dependence on high-performance edge machines becomes costly compared to low-performance machines. This conclusion will motivate researchers to move toward commodity edge computing.

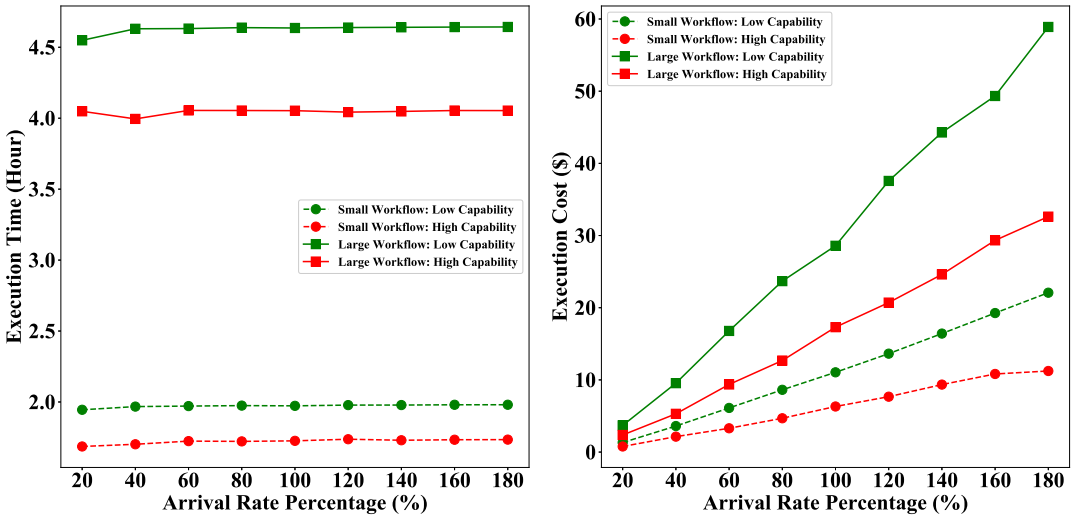

FIGURE 4.14: Compare the variation of execution time for small and large workflows based on edge capability and change in arrival rate

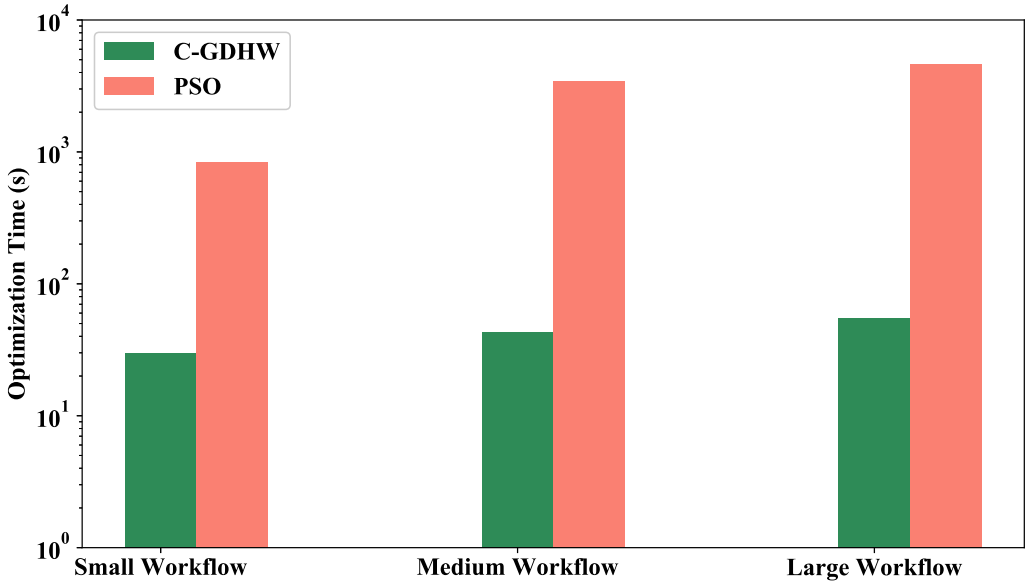

FIGURE 4.15: Optimization time of PSO and C-GDHW for different hybrid workflows (Log Scale)

4.5.2.4 Optimization Time Analysis

The increase in optimization time can have a significant impact on the scheduling process, particularly for large workflows. In a more complex scenario, the scheduling process could be repeated at short intervals, to provide accurate decisions about model parameters, such as arrival rate and window size. Figure 4.15 shows the advantage of the proposed estimation algorithm, over those from our previous work using the PSO algorithm, in sustaining the optimization time with variation in workflow complexity and number of tasks.

The optimization time complexity increases by 45%, 55% and 60% for small, medium and large workflows respectively. The low optimization complexity time of the GDS-based optimizer suggests that linear optimization models should be adopted to solve complex workflow estimation and scheduling problems.

4.6 Summary

This chapter presented hybrid workflow scheduling framework for an edge cloud computing resource model. The framework includes two stages. In the first stage, we propose a resource estimation algorithm based on a linear optimization approach with GDS, and in the second stage, a cluster-based provisioning and scheduling technique for hybrid workflows in heterogeneous edge cloud resources was introduced. The aim was to achieve a multi-objective optimization of execution time and monetary cost under constraints of deadline and throughput. In comparison to two nominated scheduling techniques, namely, PSO and full-mode, the proposed technique was able to control the execution of hybrid workflows by efficiently tuning several parameters, including stream arrival rate, processing throughput and workflow complexity. The proposed scheduler reduced cost and time by a minimum of 35% and a maximum of 70% compared to the other nominated techniques. Moreover, the GDS-based technique significantly reduced the optimization complexity compared to the PSO technique, by 50% on average.

This chapter proposal provided limited effort to utilize the capability of edge computing. We assumed workload migration to the cloud in the case of edge failure. The next chapter describes experiments with hybrid workflow provisioning and scheduling in cooperative edge cloud computing. It includes an improvement to the GDS technique to handle online resource estimation without prior knowledge about the input workflow structure and constraints. The current version of the GDS technique does not support online estimation for unknown workflow structures.

Chapter 5

Hybrid Workflow Provisioning and Scheduling on Cooperative Edge Cloud Computing

This chapter extended the work presented in chapter 4 and presents a hybrid workflow scheduling framework for cooperative edge cloud systems. A cooperative model was adopted to resolve the issues of latency-sensitive application as well as to improve resource utilization at the edge layer. In addition, the GDS-based resource estimation technique was extended to meet the requirements of unpredictable workflow structures. In experimental evaluation, the cooperative model reduced cost by 40% compared to the non-cooperative model. In addition, the results demonstrated the cooperative model's ability to maximize data migration within the edge layer, and accordingly reduce data transfer to the cloud layer.

5.1 Introduction

For decades, cloud computing has proved stable solutions for applications at different scale, robustness, and cost levels. Nevertheless, and despite cloud computing being the most convenient paradigm for resource-intensive applications, achieving

latency constraint has become a significant challenge. Pushing continuous data streams from a large number of IoT devices to cloud servers raises substantial latency and timely processing concerns. By 2021, the volume of data generated from IoT devices that needs to be processed will far exceed the capacity of central clouds [165]. Thus, it is vital to perform computation closer to end users, for example, in radio base stations and other systems on the edge of access networks a style of computing known as edge computing. Results provided in Chapter 4 demonstrate that edge resources can be involved in executing hybrid workflows. The ultimate advantage of edge computing is to bring computation closer to data sources. Thus avoiding large data transfers to the cloud to reduce transfer time and cost.

However, the availability and low efficiency of edges servers are decisive factors in quality achievement [121]. Moreover, the collaboration between edge and cloud computing models allows reliable integration between stream and batch applications in the context of IoT processing. However, in comparison to cloud resources, edges are often limited in terms of computation and storage capability, and availability (subjects to power and network connectivity). In addition, edge nodes are heterogeneous in both storage and computation capacities. Workload balancing to align the heterogeneity of edge node, causes exponential computation complexity for resource allocation and scheduling algorithms [166]. To improve the performance of edge resources, a collaborative schema can be adopted.

The main question addressed in this chapter is "how integration between collaborative edges and the cloud can resolve the issues of hybrid workflows, and resources utilization without increasing the complexity of resource allocation and task scheduling algorithms". This chapter provides the answer in two ways. The first is modeling the integration between stream and task applications at a low-level to accomplish the QoS requirements for each application model, and the second is proposing a scheduling framework to execute hybrid workflows on cooperative edge cloud computing.

The rest of the chapter is structured as follows. Section 5.2 discusses the related work on workflow resource estimation, provisioning, and task scheduling. A detailed description of cooperative edge cloud system model is provided in Section 5.3. Section 5.4 presents a detailed description about hybrid workflow scheduling in a cooperative edge cloud system. Section 5.5 reveals the experimental findings and provides insights into the main results followed by chapter summary in Section 5.6.

5.2 Related Work

This section contains a review of related research disciplines of resource estimation and provisioning, workflow scheduling in edge cloud and cooperative edge cloud systems. Workflow scheduling major research directions in academia and telecommunication industry, and therefore there is an extensive body of related research body. Many algorithms and techniques have been proposed for scheduling optimization on computing environments, including grid [44], cloud [45, 46], multi-cloud [47], and edge [48]. Common challenges of adopting edge computing in workflow scheduling are scalability, self-adaptively, and reliability [112]. In IoT platforms, it's challenging to extend the functionalities of IoT devices [48], and techniques of service replication and migration [167] are needed to achieve resource/service reliability. Edge computing is a means of overloading nodes with computationally intensive workloads.

Many works have adopted scheduling techniques for latency-sensitive workflows on edge computing. The literature review presented in Chapter 2 discussed various techniques of adopting cooperative edge systems to overcome issues like application and service latency [113, 115, 117], low system stability [115], edge resources utilization [122], application responsiveness and accuracy [124, 128], energy consumption [116] and reliability [126]. For complex workflows, such as hybrid workflows, dynamic distributed and decentralized computing model, such as edge cloud, is convenient for reliable and scalable execution [126]. With this model, the need

arises for developing schedulers that can intelligently partition workflows and allocate the partition.

To summarize, the literature contains various methods of implementing edge cloud and cooperative edge cloud computing to resolve issues of latency-sensitive and resource-intensive applications. The following points can be concluded. First, edge cooperation is limited to edge data center level, and cooperation mechanisms are not well defined or structured. Second, hybrid workflow scheduling is not a feature of the literature, and the integration between stream and batch processing is not clearly associated with the scheduling process. Finally, the impact of stream and batch processing parameters tuning (e.g., how change in stream arrival rate or processing throughput could affect scheduling behaviour and optimization decisions) has not been studied.

5.3 A Cooperative Edge Cloud Computing System

The cooperative edge cloud is an integrated computing model utilized to run complex applications such as real-time data analytics [168], visual guiding services using a wearable cameras [169], on-demand gaming [170] and real-time video streaming [171]. The cooperative model extends the edge cloud resource model presented in Section 4.3, and incorporates three main layers, IoT, cooperative edge, and multi-cloud. The work outlined here assumes that at a certain level, resources among all layers are accessible to each other. Figure 5.1 presents the system model of cooperative edge cloud computing.

- *Layer 1 (IoT).* An IoT device is sensing hardware capable of transmitting data streams over the Internet via a variety of network interfaces like WiFi, Long Term Evolution (LTE), 4G and 5G. IoT devices include wireless sensors, software, actuators, and computer devices. They can be embedded

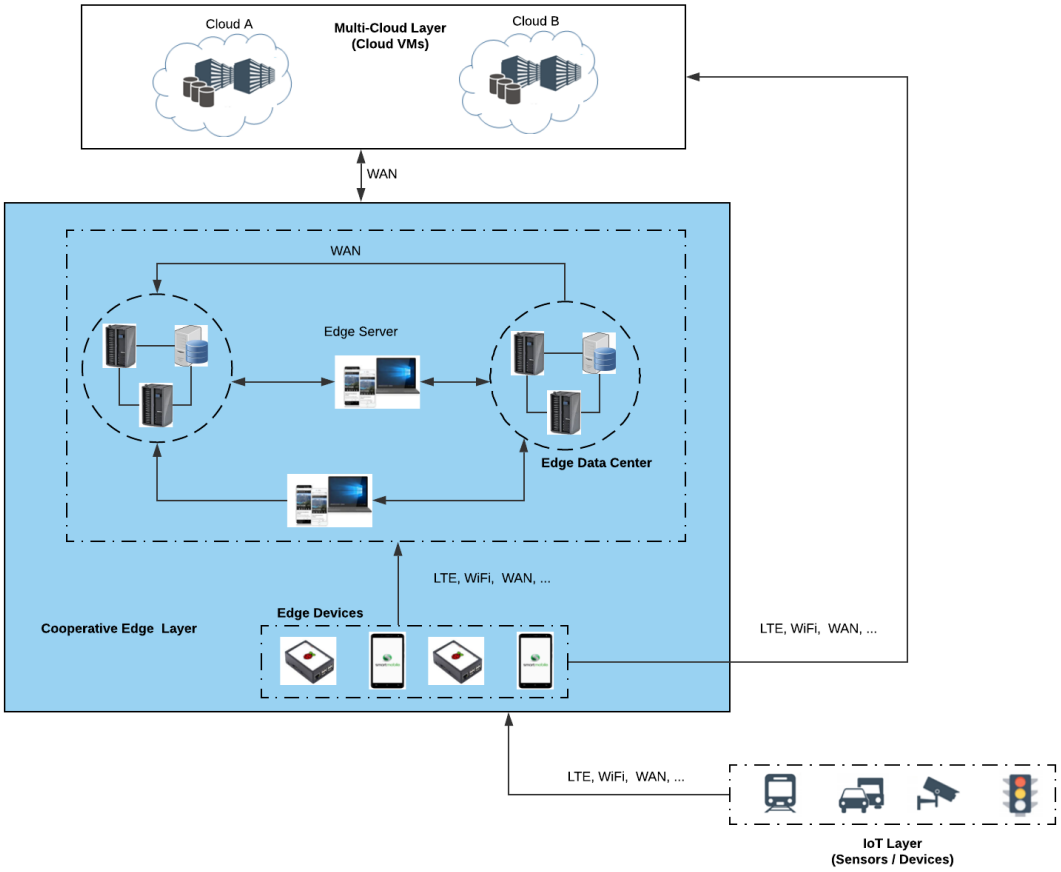

FIGURE 5.1: Cooperative Edge Cloud Computing System Model

into mobile devices, industrial equipment, environmental sensors, medical devices, and more.

- *Layer 2 (Cooperative Edge Nodes).* This layer is composed of two resource components, namely, edge devices and edge data centers. Edge devices are non-stationary computation nodes perform the closest layer to IoT data sources. These devices are limited in computation and storage capabilities, but able to handle some pre-processing tasks in the stream processing pipeline. On the other hand, edge data centers are powerful and interconnected edge nodes, that can undertake resource-intensive tasks, and are able to communicate with other data centers for sharing data and computation. Overall, the edge layer offers computation closest to data sources (IoT devices and sensors), thereby reducing data transfer time, and performing pre-processing in a timely-manner constraint [20].

The edge cooperation schema is applied when task waiting time is high; the computation queue buffer is full, the task is computation-intensive and exceeds edge capacity, and cumulative generated data is large and needs to be migrated to a capable edge. Edge data centers can be operated by individuals, organizations or cloud service providers.

- *Layer 3 (Multi-cloud Services):.* The third layer consists of powerful resources which are adequate to carry out complex services in domains like machine learning, business intelligence, and interactive visualization.

5.4 Hybrid Workflow Scheduling on Cooperative Edge Cloud Computing

Hybrid workflow scheduling is utilized to estimate resources and plan resource allocation for tasks based on the required (QoS) parameters, and is responsible for selecting optimal virtual machines for workflow execution using a preferred scheduling algorithm. Section 3.3 provides detailed modeling needed for hybrid workflows, which includes the mathematical modeling to estimate the number of servers for stream and batch tasks. Figure 5.2 presents a high-level abstraction for a scheduling framework in a cooperative edge cloud system. The framework is described as follows.

1. *Workflow submission.* A user submits a hybrid workflow to the workflow manager with sufficient configuration of structure, data sources, and preferred QoS constraints, which may include minimum throughput and maximum execution deadline. It is assumed that the workflow manager has no prior knowledge about the incoming workflow.

2. *Workflow Profiling.* In the work presented in Chapter 4, the availability of a submitted workflow usage function was assumed. This function describes the relation between resource estimation parameters (arrival rate λ, window size ω, and throughput τ), and total number of cores R and workflow

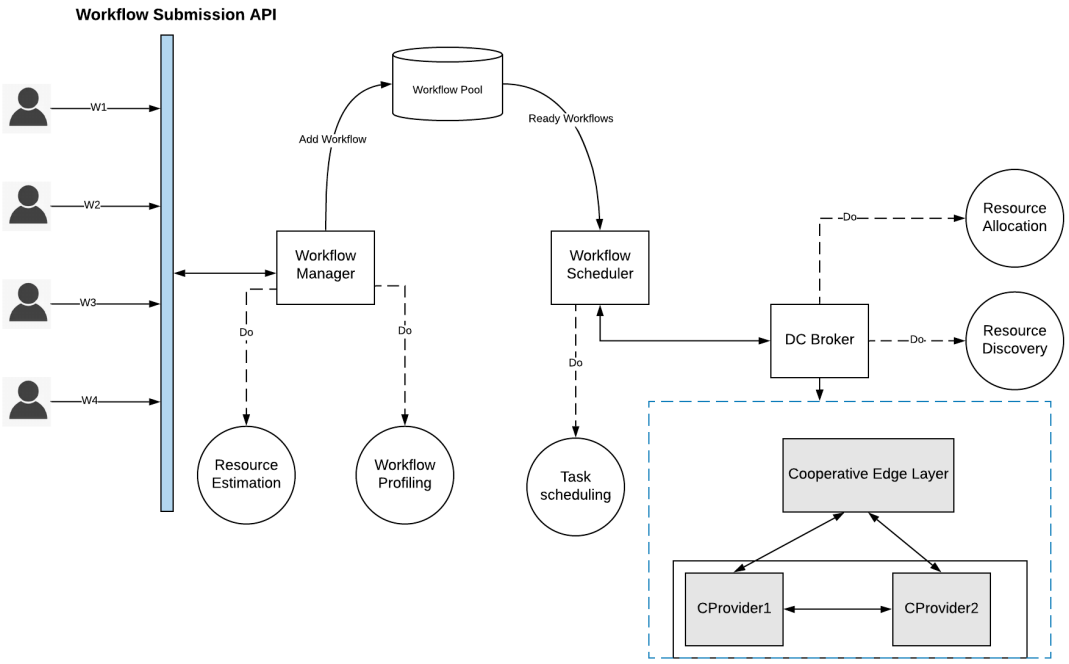

FIGURE 5.2: Workflow Scheduling Framework on Cooperative Edge Cloud Computing

execution time E. However, this assumption is not realistic with high variation on hybrid workflow application models, in IoT and data analytics as examples. Thus, the workflow manager will perform an online profiling process to understand the workflow behaviour corresponding to the mentioned parameters.

3. *Resource estimation.* In previous chapter , we proposed a multi-objective optimization technique to reduce number of computation units (cores) R and workflow execution time E. The Gradient Descent technique [172] (C-GDHW) performs a linear search for tuning stream parameters considering constraints of throughput, waiting time and deadline. Specifically, the aim is to find the optimal combination of problem input parameters, that is, arrival rate λ, aggregation window size ω, and workflow throughput τ.

   In this chapter, an online resource estimation based on the workflow profiling is adopted. The main challenge in applying the C-GDHW algorithm is the unavailability of an exact approximation for the workflow usage function, and thus, calculate the gradient at a given point. The estimation process yields

two outcomes. The first is a workflow execution groups G which formulates the dependencies among workflow tasks, and the second is the estimated number of cores R at group level. The technique adopted to handle the challenge is illustrated in next section.

4. *Resource discovery and allocation.* The next step is to allocate resources in the cooperative edge cloud environment. The estimation process determines resources required to execute the workflow and satisfy user QoS requirements. The edge cloud cooperative model unites heterogeneous resources with different computation capability, storage capacity, and communication protocols. The resource discovery implies investigating the current status of resources at the edge layer. For edge devices, the framework collects details about computation capability (number of available cores) and network quality. For edge data centers, the framework communicates with the resource manager to fetch resource details.

   Data collected during the discovery process allows the framework to build resource allocation plans P. Each plan represents an allocation strategy to combine resources from different resources layers, that is, edge devices and data centers, and multi-cloud providers, to accomplish estimation outcomes. The point of constructing alternative allocation strategies is to reduce the complexity of the scheduling process in optimizing the computation time and cost.

5. *Task scheduling.* We adopted a cluster-based technique to schedule workflow tasks, which relies on constructed groups G. The technique aims to select the best strategy from resource allocation strategies P from the previous step by performing a multi-objective optimization process to reduce the workflow execution time T and monetary cost C. The scheduling process is discussed later in this section.

5.4.1 Hybrid Workflow Resource Estimation with a Gradient Descent Approximation Technique

Resource estimation using a gradient descent technique is explained in Chapter 4. To summarize, the process aims to find the configuration tuple $[\lambda, \omega, \tau]$ for each stream task, by which, the optimum number of cores R and maximum execution time E is achieved. The gradient-based technique applies an iterative process of finding a local minimum (descent) of a differentiable function $F(x)$. Each step involves a proportional movement to the negative of the gradient of the function at the current point. The movement generates a new point, which represents an updated version of the stream configuration. Next, the cost of estimation function is calculated, and the process is stopped when no improvement is achieved.

To apply the gradient-based technique, a workflow estimation function $F(x)$, to calculate the joint value of the number of cores R and the maximum execution time E, must satisfy two conditions: differentiability and convexity. Differentiability is mandatory to accomplish the line search, while convexity guarantees the existence of a function global minimum. In previous work, we assumed prior knowledge of the cost estimation function $F(x)$, and thus, it can be evaluated in offline mode.

The assumption of prior knowledge of the cost estimation function $F(x)$ is not achievable for all workflow structures. To attain a generic estimation approach for hybrid workflows, the previous technique was extended to address the online estimation mode. The proposed technique is referred to Adaptive Constraint-based Gradient Descent search for Hybrid Workflows (AC-GDHW). The applicability of online gradient approximation is subjected to two observations:

- The cost estimation function $F(x)$ is differentiable on each estimation parameter domain. The function derivative exists at each point in its domain. All combinations of estimation parameters have estimation values, thus, the function never has a jump discontinuity.

- According to differentiability, the estimation function can be approximated locally by linear functions. Having this fact, the function derivative (gradient) at a certain point x_i can be approximated locally.

Algorithm 8 shows the steps to find an approximation of the function $F(x)$ gradient at point x. The approximation function *ApproximateGradient* receives three parameters, namely, the cost estimation profiling space S, target point x, and target estimation parameter var. The algorithm starts with finding the point that best fits to the target point x on the profiling space S. We used the Euclidean distance to find the closest point in a 3-dimensional space S. Moving to the var dimension, the algorithm locates the closest two points on the same dimension from left and right directions, xL and xR, respectively. The next step is calculating the function $F(x)$ derivative at point x by finding the slope in the two directions using the function $CallculateDerivative$. The derivative value of x is the average of slope values from left and right, $Derivative_L$ and $Derivative_R$, respectively. The algorithm is applied in all workflow stream tasks to construct the optimized configuration tuple.

After setting stream configurations, the estimation technique is applied to find the number of cores c required to execute each workflow task. The last step at this stage is constructing the workflow execution groups $G = \{G_1, G_2, ..., G_n\}$. Each group G_i includes all tasks which can executed concurrently, $G_i = \{t_{i,1}, t_{i,2}, ..., t_{i,n}\}$ where $t_{i,j}$ is a stream or batch task, and n is the number of group tasks. Mathematical notations used in the estimation modelling is presented in Table 5.1. A group G_i has the following properties.

Algorithm 8 Find Function Gradient from Profiling Space

1: **procedure** ApproximateGradient (S, x, var)
2: $x_0 = FindClosestPoint(S, X)$
3: $xL, xR = FindPointsLeftRight(S, x)$
4: $Derivative_R = CallculateDerivative(x_0, xR, var)$
5: $Derivative_L = CallculateDerivative(x_0, xL, var)$
6: $Derivative_x = (Derivative_L + Derivative_R)/2$
7: **return** $Derivative_x$

TABLE 5.1: Mathematical notations used in resource estimation modelling

Notation	**Description**
G	A set of all constructed groups.
G_i	A group of concurrent tasks.
EST_i	Earliest start time of a task t_i
EFT_i	Earliest finish time of a task t_i
D_i	Total data generated by group G_i.

$$G_i = \{R_i, EST_i, EFT_i, D_i\} \tag{5.1}$$

$$R_i = \sum_{j=1}^{n} c_j \tag{5.2}$$

$$R = \sum_{i=1}^{n} R_i \tag{5.3}$$

$$E_i = EFT_i - EST_i \tag{5.4}$$

$$E = \sum_{i=1}^{n} E_i \tag{5.5}$$

$$D_i = \sum_{j=1}^{n} d_j \tag{5.6}$$

$$\min \quad (R.E) \tag{5.7}$$

5.4.2 Hybrid Workflow Provisioning and Scheduling Framework on Cooperative Edge Cloud Computing Environment

In previous chapter, C-HWPS framework is adopted to provision demanded resources and schedule tasks on edge cloud computing environment. The cluster-based approach aims to perform resource allocations and scheduling for workflow groups G from the previous stage. In this chapter, C-HWPS is extended as Cluster-based and Cooperative Hybrid Workflow Provisioning and Scheduling Cooperative

(CC-HWPS), to allow provisioning and scheduling in a cooperative edge cloud computing system, which includes three types of computation resources: edge devices, edge data centers, and cloud resources. The provisioning and scheduling framework CC-HWPS is described as follows.

1. *Resource Discovery.* The first step is fetching the current status of edge nodes. For edge devices E_{dv}, the profiler keeps track of details including, available number of cores, network interface, and connectivity with edge data centers and cloud. For edge data centers E_{dc}, the profiler collects details about number of offered computation nodes, maximum storage, and the communication mechanism with other edge data centers and cloud.

2. *Resource Provisioning.* One critical issue to consider when scheduling in a cooperative edge cloud environment is edge resources heterogeneity and load balancing [130]. The cooperative model illustrates two objectives: increased edge resources utilization and reduced the data communication between edge devices and data centers. To achieve and balance these objectives, the scheduler should distribute task groups over edge and cloud resources. In this step, the resource provisioner will produce potential group' execution strategies $P = \{P_1, P_2, ..., P_k\}$. A strategy P_i includes three types of computation nodes: edge devices E_{dv}; an arrangement of edge data centers E_{dc} in which an edge data center E_{dc_i} can communicate with at least another data center E_{dc_j} or a cloud data center C_{dc_i}; and cloud data center C_{dc}. Mathematical notations used in the resource provisioning and scheduling modelling is presented in Table 5.2.

   A computation node N_i is modeled as:

$$N_i = \{\beta_i, \Upsilon_i, \Gamma_i, \zeta_{\beta_i}, \zeta_{\Upsilon_i}, \zeta_{\Gamma_i}\} \tag{5.8}$$

3. *Group Scheduling.* Tasks within a group G_i are independent and can be executed concurrently; each group can be referred as a cluster of tasks. The cluster-based technique increases resource utilization and efficiency because

TABLE 5.2: Mathematical notations used in workflow scheduling modeling

Notation	**Description**
β_i	Communication bandwidth between computation data centers/servers.
Υ_i	Computation power of a server.
Γ_i	Storage capacity of a data center/server.
ζ_{β_i}	Bandwidth cost between computation data centers/servers.
ζ_{Υ_i}	Processing cost a server.
ζ_{Γ_i}	Storage cost a data center/server.
PT	Processing time for a group G_i.
$PC.$	Processing cost for a group G_i.
TT	Total transfer time to send data between two data centers/servers.
$TC.$	Total transfer cost to send data between two data centers/servers.

it facilitates the usage of powerful and high-capacity machines. The scheduler's mission is to find the execution strategy that jointly optimizes the execution time T and monetary cost C of running all workflow groups G. The optimization function of the scheduling problem is provided as follows.

$$\min \quad (T.C) \tag{5.9}$$

The formulas calculating the total execution time T and cost C are s follows.

$$PT_i = \sum_{j=1}^{n} ComputeTime(\Theta_j) \tag{5.10}$$

$$PC_i = \sum_{j=1}^{n} ComputeCost(\Theta_j, \zeta_{\Upsilon_j}, \zeta_{\Gamma_j}) \tag{5.11}$$

$$TT = \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{\delta_{j,k}}{\beta_k} + \epsilon_{j,k} \tag{5.12}$$

$$TC = \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{\delta_{j,k}}{\beta_k} * \zeta_\beta \quad j \neq k \tag{5.13}$$

$$T = \sum_{i=1}^{n} PT_i + TT \tag{5.14}$$

$$C = \sum_{i=1}^{n} PC_i + TC \tag{5.15}$$

In details, the formulation is described as:

- A group G_i is partitioned into a set of subgroups $\{\Theta_1, \Theta_2, ..., \Theta_n\}$ according to strategy P_k. The notation Θ_j refers to the list of tasks that can be executed in computation node N_j. The $ComputeTime$ function is responsible for provisioning the computation node N_j if the node represents an edge E_{dc} or cloud data center C_{dc}. The function returns the execution time for the longest task.

- Based on the provisioning plan provided in the first step, the function $ComputCost$ calculates the cost of executing a subgroup Θ_j in computation node N_j with respect to processing cost ζ_{Υ_j} and storage cost ζ_{Γ_j}.

- The data transfer time TT is the sum of all data transfer processes between computation nodes N_j and N_k to transfer amount of data $\delta_{j,k}$ with latency $\epsilon_{j,k}$. The data transfer cost TC is calculated in the same manner. The cost of moving data within the same computation node/cluster is neglected.

- The total time T to run a hybrid workflow is the sum of processing time for all groups and the time to exchange data between computation nodes. Similarly, the total cost C to run a hybrid workflow is the sum of processing costs for all groups and the cost of the exchanging data between computation nodes.

Algorithm 9 performs workflow scheduling. It receives inputs of task groups G as a result of the estimation process, and resource allocation strategies S from the resource provisioning step. Scheduling algorithm can be expressed as follows. In step 1, lines 2-6, the algorithm initializes objective variables, T and C. In step

Algorithm 9 Cluster-based and Cooperative Hybrid Workflow Scheduling on Cooperative Edge Cloud (CC-HWPS)

```
1:  procedure CC-HWPS(G, S)
2:      R = {}
3:      T = C = ∞
4:      T_temp = C_temp = ∞
5:      G_target = null
6:      for s in S do
7:          for g in G do
8:              PT, PC = computeGroupTimeCostWithStrategy(g, s)
9:              T_temp = T_temp + PT
10:             C_temp = C_temp + PC
11:         TT, TC = computeDataTransferTimeCost(s)
12:         T_temp = T_temp + TT
13:         C_temp = C_temp + TC
14:         if T_temp.C_temp < T.C then
15:             T = T_temp
16:             C = C_temp
17:             s_target = s
18:         T_temp = C_temp = ∞
19:     return R, T, C
```

2, lines 6-18, the algorithm iteratively examines the provisioning and scheduling of workflow groups based on allocation strategies S, the group with minimum $T_{temp}.C_{temp}$ is selected. T_{temp} is the processing time of group g with strategy s and C_{temp} is the processing cost with the same strategy.

The function $computeGroupTimeCostWithStrategy$, line 8, calculates the processing time and cost of executing group g within strategy s in the context of the cooperative edge cloud. Algorithm 10 shows the steps of this computation logic. After calculating the computation time and cost with the given strategy s, the next step is calculating the communication time TT and cost TC based on the amount of data transferred between computation and storage servers, in which task predecessors to group g tasks are located, line 12. It is worthwhile mentioning that each iteration progresses a sup-optimization problem to approach the clustering optimization behaviour. Next, lines 14-17, the strategy s is selected if it provides total time T and cost C reduction. The algorithm stops after examining all allocation strategies S. Finally, the algorithm returns scheduled tasks G, and

Algorithm 10 Compute the time and cost of executing workflow group with a resource allocation strategy

```
1: procedure COMPUTEGROUPWITHSTRATEGY(g, s)
2:     T = C = 0
3:     V = getGroupTasks(g)
4:     while V ≠ ∅ do
5:         for r in s do
6:             AvaillMIPS = getResourceMIPS(r)
7:             A = ∅
8:             v = getNextUnallocatedTask(V)
9:             ToAllocateMIPS = getTaskMIPS(v)
10:            while ToAllocateMIPS < AvaillMIPS do
11:                v = getNextUnallocatedTask(V)
12:                ToAllocateMIPS = ToAllocateMIPS + getTaskMIPS(v)
13:                A = A ∪ v
14:                V = V − v
15:                if V = ∅ then
16:                    Break
17:            if A ≠ ∅ then
18:                provisionAndAllocateTasks(A, r)
19:                T = T + computeTime(A, R)
20:                C = C + computeCost(A, R)
21:    return T, C
```

optimization object variables, T and C.

Algorithm 10 shows, in a high-level abstraction, the steps in allocating task group g with a strategy allocation s in a cooperative edge cloud model. The first step is allocating group tasks V. While there are unallocated tasks, the algorithm tries to find a computation node r which accomplishes the computation requirements of a task v. To attain high resources utilization, computation nodes in strategy s are ordered by their processing power (MIPS).

5.5 Performance Evaluation

This section presents the performance evaluation hybrid workflow scheduling on a cooperative edge cloud system . It firstly provides details about the experimental setup, including structuring and constructing hybrid workflows from existing IoT application benchmarking, and setting-up the cooperative edge cloud system.

Then, the experimental results are discussed and insights about edge cooperation performance are highlighted.

5.5.1 Experimental Setup

In the previous sections of this chapter, an online scheduling framework for hybrid workflow in cooperative edge cloud system was proposed. The online scheduling involves no pre-knowledge about the current status of the edge resources, which are obtained at the time of scheduling by the resource discovery system. The edge status includes details about resource availability, capacity and cooperation between edge resources is considered. At application level, the variation in model parameters, and how this variation contributes to the optimization behavior. A hybrid workflow is an integration of stream and batch tasks in the form of a DAG structure. An IoT dataflow is an example of a hybrid workflow. To construct hybrid workflow structures, we followed the setup described in Chapter 4. Small, medium horizontal-scale, and large vertical-scale were constructed. A scalability factor was included to enable the study of relationship between workflow structure and optimization parameters. Table 5.3 presents the main characteristics of these workflows.

For each model parameter, simulation was carried out 30 times, and average values were used for comparing the performance of running hybrid workflows on three computing systems: cooperative edge cloud, non-cooperative edge cloud, and cloud-only. Following is the description and assumptions related to each computing system.

- Cooperative edge cloud. In the cooperative model, edge resources work collaboratively by sharing and exchanging computation and data workloads. An edge device or data center can pass allocated task(s) to nieghbours in case of computation or storage shortcomings.

- Non-cooperative edge cloud. In the non-cooperative model, computation or storage collaboration is not assumed. Unhanded computation and data workloads are sent directly to the cloud.

- Public Cloud or Cloud-only. In this model, all actions related to workflow execution are undertaken by a cloud provider without usage of any edge devices.

We extended CloudSim [157] to reflect the adopted resource model as an cooperative edge cloud computation environment. Three types of resources were modelled: cloud, edge data centers and edge devices. This modeling is aligned with the system model provided in Figure 5.1. Table 5.4 provides the resource model configuration setup of the experiment. In addition, the edge cooperation status is simulated to reflect the resource discovery service which includes the computation capability of edges, accessibility and communication quality between edges. The the edge cooperation status is examined in prior of workflow execution.

Table 5.3: Hybrid Workflows Characteristics

Workflow	**#Stream Tasks**	**#Batch Tasks**	**Scale Mode**
Small	12	17	Equal
Medium	20	45	Horizontal
Large	36	73	Vertical

Table 5.4: Resource types for edge cloud system

Provider	**#Cores**	**Processing Cost ($/Hour)**	**Bandwidth (MB/s)**	**Bandwidth Cost ($/GB)**
Cloud DC	2 4 8	[0.80, 0.90]	[60.0, 80.0]	[0.14, 0.20]
Edge DC	2 4 8	[0.30, 0.35]	[20.0, 40.0]	[0.07, 0.08]
Edge Devices	2 4	[0.20, 0.25]	[20.0, 25.0]	[0.03, 0.04]

In this chapter, the proposed resource estimation technique, the adaptive gradient-based, AC-GDHW, and the provisioning and scheduling technique, CC-HWPS are used to evaluate the performance of three computing environments; cooperative edge cloud, non-cooperative edge cloud, and cloud-only to run hybrid workflows. Technically, schedulers proposed in chapters 3 and 4, HWRPO and C-HWPS, respectively, are special cases from CC-HWPS to schedule hybrid workflows on cloud-only and non-cooperative edge cloud systems, respectively. The experiment was performed to determine the contribution of edge cooperation on reducing the execution time T and monetary cost C to executing hybrid workflows.

5.5.2 Results and Discussions

This section discusses the results of applying the resource estimation and scheduling framework to three hybrid workflow structures.

5.5.2.1 Edge Cooperation Evaluation

This section evaluates the efficiency of CC-GDHW in optimizing hybrid workflow execution time T and cost C based on the variation of model parameters (stream window size ω, stream arrival rate λ, and execution throughput τ) and workflow scalability (vertical and horizontal). Since an online scheduling mode was adopted, each data point for execution time and cost is an average of multi-iterations workflow execution.

The window size is the time needed for processing and aggregating the incoming data stream. Figures 5.3, 5.4 and 5.5 show the results of varying window size on execution time and monetary cost for the three workflow structures. As window size increases, the time to capture data streams increases. This challenges the scheduler to handle the cumulative collected data from IoT devices. Here it is we assumed that stream processing is basically allocated to edge devices for the sake of reducing streams collection delay from IoT devices, and to benefit from the non-stationarity advantage for high data collection coverage. However, edge

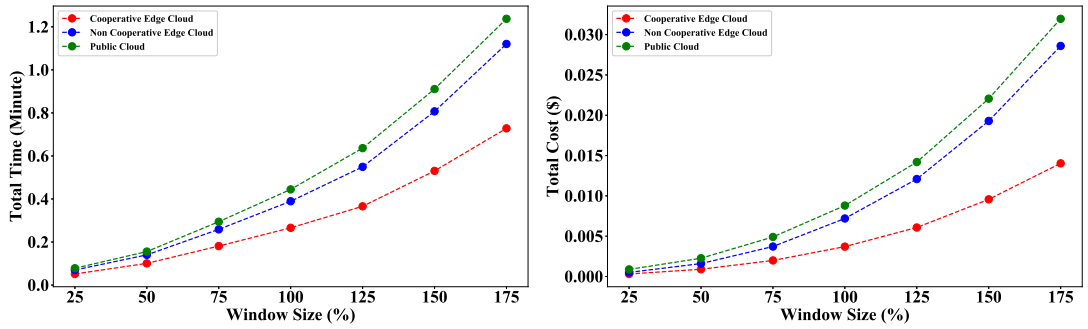

FIGURE 5.3: Window size variation impact on small hybrid workflow scheduling

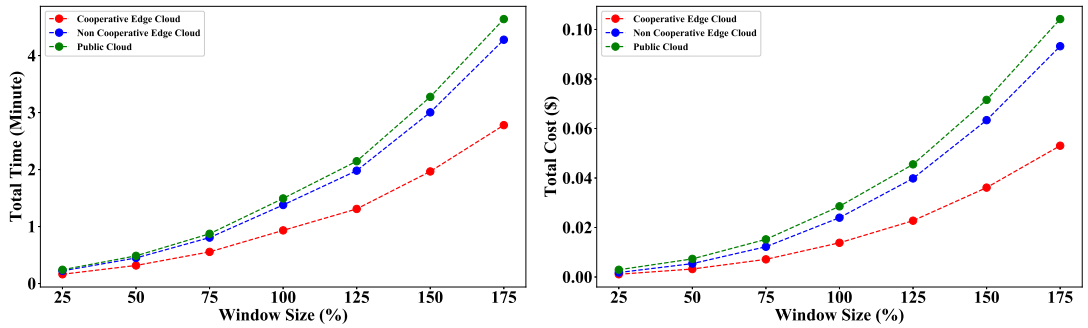

FIGURE 5.4: Window size variation impact on medium hybrid workflow scheduling

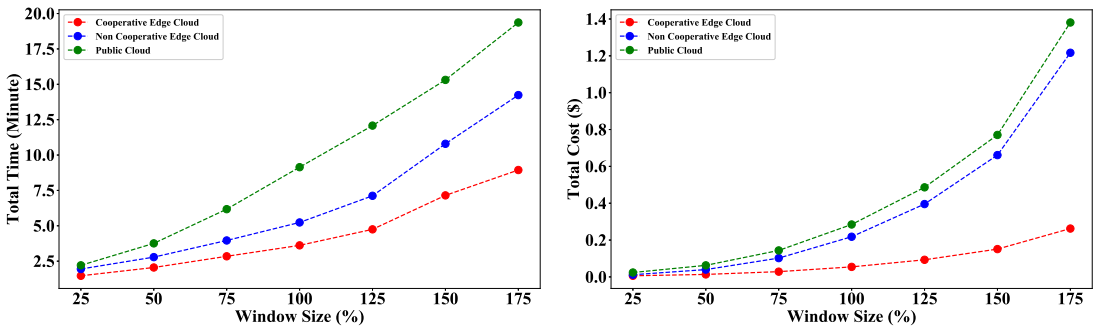

FIGURE 5.5: Window size variation impact on large hybrid workflow scheduling

devices are limited in the computation and storage capabilities. The cooperative model allows them to pass computation and data loads to more powerful resources with reduced overhead on communication networks, particularly, with non-free network interfaces. The advantage of the cooperative scenario is clear for complex workflow structure (Figure 5.5). In comparison to the non-cooperative scenario, the cooperative scenario produced a significant time and cost savings of 50% and 85% compared with the non-cooperative scenario, respectively.

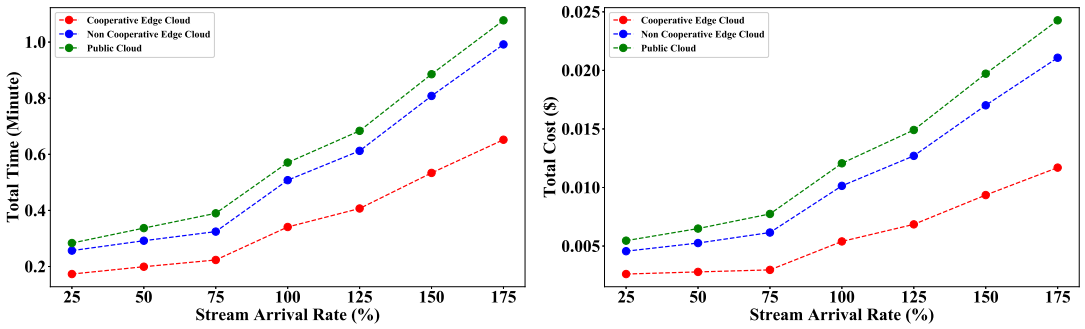

FIGURE 5.6: Arrival rate variation impact on small hybrid workflow scheduling

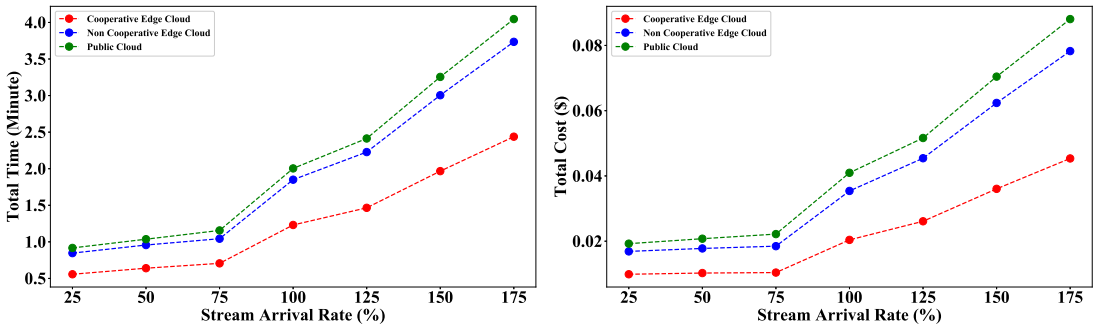

FIGURE 5.7: Arrival rate variation impact on medium hybrid workflow scheduling

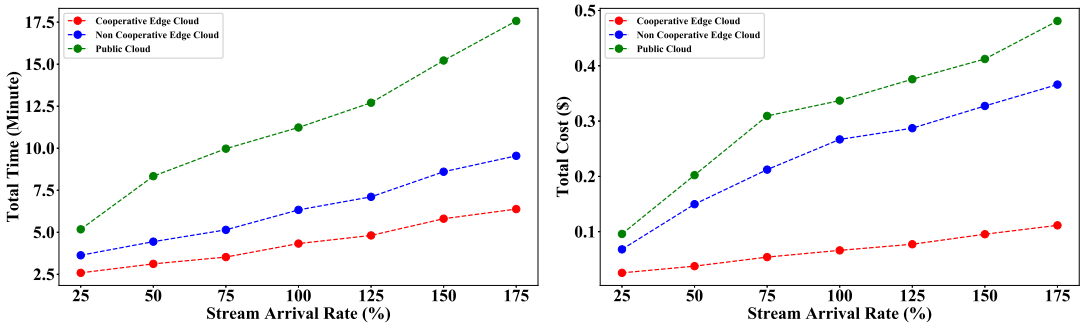

FIGURE 5.8: Arrival rate variation impact on large hybrid workflow scheduling

The impact of network overheads with large data transfers explains the poor performance of the non-cooperative scenario in optimizing hybrid workflow execution. With a non-complex workflow structure, Figure 5.3 shows that the scenario reduced time and cost by 10% compared to the public cloud scenario. The reduction percentage decreases as the workflow application becomes more complicated.

The arrival rate refers to the number of stream inputs in a given time interval. Figures 5.6, 5.7 and 5.8 show the results of comparing the responsiveness

of the three models to the variation in stream arrival rate for the three workflow structures. The increase in stream rate involves receiving more data within the aggregation window time while preserving the throughput level, set at 60% for this experiment. In addition, the arrival rate λ has direct correspondence to the number of servers (cores) in a queuing system to keep system utilization at acceptable level. For an arrival rate of less than 100%, the scheduler was able to control the execution time and cost because stream tasks are provisioned as a group. However, for public cloud and non-cooperative computing systems, handling high speed streams is challenging. For example, with a large workflow, Figure 5.8 shows the execution cost increased by a maximum of 500% and 200%, for public cloud and non-cooperative edge systems, respectively. On the other hand, the cooperative edge system distributes the workload overhead with a 50% cost increase. This demonstrates the cooperative model's ability to utilize edge resources without migrating computation and data workloads to the cloud.

The **throughput** determines the percentage of data to be processed to accomplish a workflow execution objective. This parameter is a real business-related selection, and differs from one application to another. In this work, the influence of application throughput on execution time and cost was assessed. Figures 5.9, 5.10 and 5.11 show the impact of throughput variation on the three workflow structures. One conclusion is that the scheduler must adjust arrival rate and window size parameters to achieve minimum throughput. Meanwhile, a high throughput incurs additional time and cost with the cloud-only system; edge-based systems are more efficient at processing more data streams.

5.5.2.2 Analysis of Edge Cooperation Impact on Data transfer time and cost

The main motivation for using cooperative edges is to allow maximum cooperation between edge resources to reduce the amount of data and computation workloads sent to the cloud. This section outlines the results of analyzing the contribution of edge cooperation to data transfer time and cost.

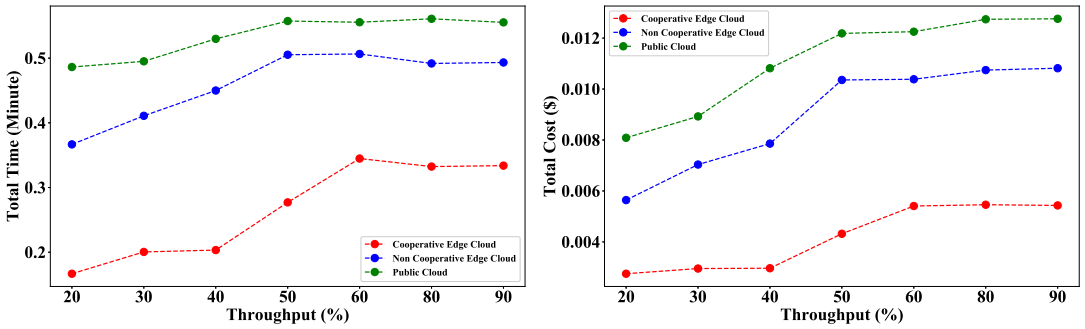

FIGURE 5.9: Throughput variation impact on small hybrid workflow scheduling

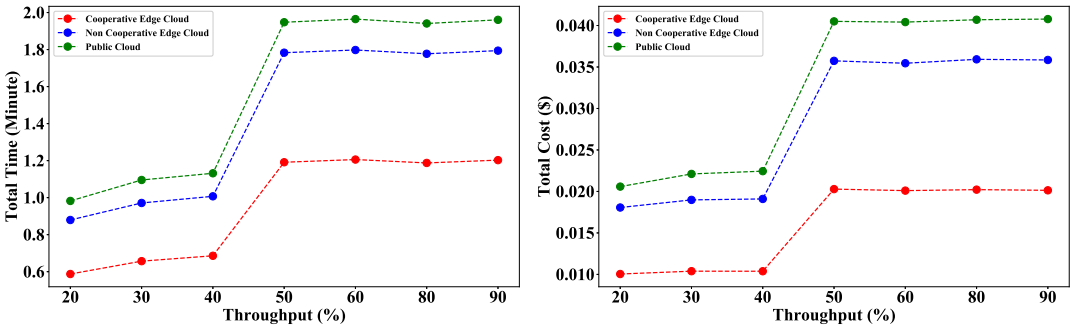

FIGURE 5.10: Throughput variation impact on medium hybrid workflow scheduling

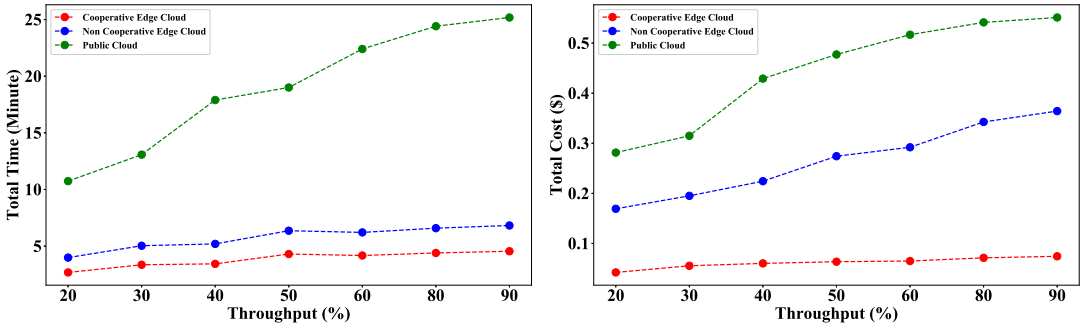

FIGURE 5.11: Throughput variation impact on large hybrid workflow scheduling

Figure 5.12 shows the results of running a large hybrid workflow with varying of window size in cooperative and non-cooperative edge cloud systems with respect to data transfer time and cost. We selected the window size parameter because it is closely related to the amount of data generated during workflow execution. Results show that the cooperative model demonstrates significantly reduces data transfer time and cost as the window size increases. Figure 5.13 gives a clear quantitative

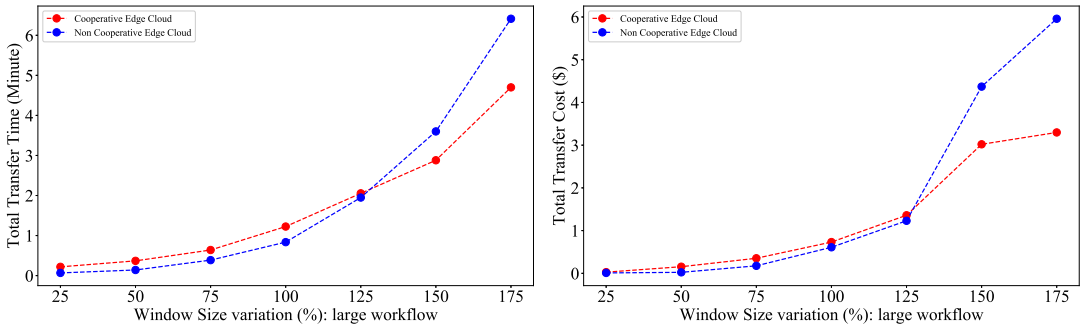

FIGURE 5.12: Edge cooperation impact on data transfer time and cost

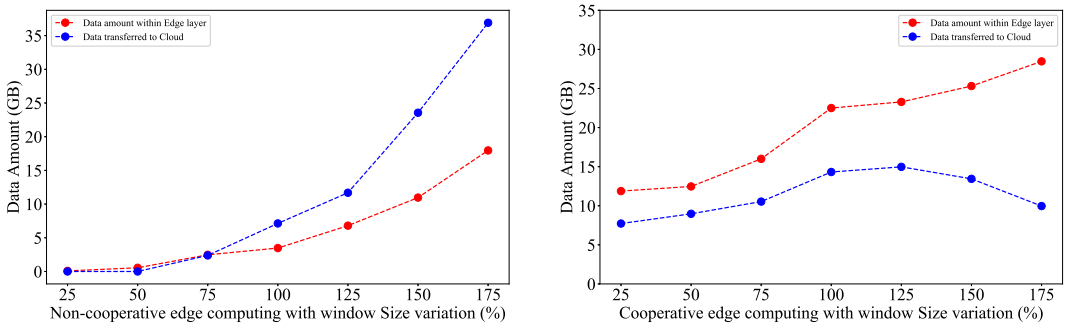

FIGURE 5.13: Edge cooperation impact on amount of data migrated

illustration of this behaviour. In the case of the non-cooperative model, the amount of data sent to the cloud increases exponentially, whereas the cooperative model and as the amount of generated data is increased, the cooperative edge layer is able to process a larger proportion of the data and reduces the communication with the cloud.

This scenario can support many applications, particularly with the engagement of massive number of data sources. Applications in smart cities, traffic monitoring and social sensing can benefit from adopting cooperative computing systems by integrating and enabling communication among resources at the edge layer to reduce dependency on public clouds, and thus achieved improved service quality, cost savings and greater data privacy.

5.6 Summary

We presented a hybrid workflow scheduling framework on cooperative edge cloud computing resource model. The cooperation implies communication between edge resources for computation and data migration under constraints of resource capability and data dependency. In this chapter, we compared the cooperative edge-cloud model with other two resource models, namely, non-cooperative edge-cloud and cloud-only models. The cooperative model reduces cost by 40% compared to the non-cooperative model. In addition, results demonstrated the cooperative model's ability to maximize the data migration within the layer, and accordingly reduce data transfer to the cloud layer. Furthermore, this chapter shows how the optimization approach, GDS, could be improved such that, it is able to perform online estimation regardless of workflow structure.

This chapter and the previous two chapters provided algorithms, techniques and frameworks to handle resource estimation and provisioning, and task mapping in the context of hybrid workflows. This simulation-based experimental work relies on verified data profiling for hybrid workflow task types. The next chapter provides a detailed description of the simulation environment.

Chapter 6

Implementation and Simulation Environment

This chapter provides a detailed description of the proposed framework for hybrid workflow scheduling in cooperative edge cloud computing systems. The framework of a cooperative edge cloud computing deployment architecture and its prototype, 'CoopEdgeCloudSim' is implemented by extending the classes of the CloudSim simulator in order to simulate resource provisioning and hybrid workflow scheduling framework in an edge cloud system.

6.1 Introduction

Internet of Things technology has been deployed to enable real-time tracking and monitoring services in many application domains, including e-healthcare, smart cities, social collaboration and interactive learning. According to McKinsey, 43 billion IoT devices will be connected by 2023, and will be vital components of 25% of the global economy [173]. As the increase in number of connected IoT devices increases, the amount of data generated by IoT devices will grow. Collected data can be small, such as a single metric of a machine's health, or large datasets generated by video surveillance cameras. IDC estimates that the amount of data

created by these connected IoT devices will grow at a annual growth rate of 28.7% over 2018-2025 [174].

The huge and rapidly growing number of sensing devices leads the transformation of traditional business models into automated digital platforms via emerging real-time connectivity, geographic data collection, and analytics capabilities. As highlighted already in this thesis, the adoption of IoT technology introduced emergent challenges for integrating hybrid application models of stream and batch applications. Stream processing enforces timely sensitive handling for incoming data with reasonable throughput, while batch processing involves the incorporation of large amounts of resources to meet deadline and service quality. According to Cisco, the growth rate of data generated from IoT devices requiring processing will far exceed the capacity of central clouds in 2021 [165].

Chapter 1 contains a discussion of the challenges of scheduling hybrid workflows, and how features like complexity, dynamicity and QoS variation are critical for obtaining resource provisioning and workflow scheduling plans. In chapter 5, we provided an efficient scheduling framework on cooperative edge cloud to align the demands of powerful computation and timely processing with a high system utilization computing system. The framework provides three core services. The first service is to provide an estimation for the required amount of resources to run a workflow under constraints of throughput and deadline. The second service is to provision these resources on edge cloud system, and the third is to map workflow tasks to provisioned resources.

The best way to evaluate the framework is to implement and test services in a real environment. However, the adopted resource model presents difficulties for real environment evaluation due to the complexity of configuring a stable computation environment at a large scale and sufficient resource variation. Moreover, scheduling optimization outcomes can be affected by communication performance between resources at all layers (cloud, edge and IoT), making it hard to produce evaluation results that meet the core research objective of studying hybrid workflow scheduling on an edge cloud system. In addition, the real environment setup

is expensive in terms of time and cost. All these challenges prevented the evaluation in a real environment; instead, a simulation framework is provided to provide more control over the environment setup and the application of related resource provisioning and scheduling policies.

To provide more realistic awareness to the simulation and modelling of IoT-enabled applications on edge cloud environment, we performed a low-level configuration modelling to convey high abstraction of computation and data communication behaviour for environment elements. The simulation in the context of edge cloud computing incorporates (i) resource capabilities of computation and storage; (ii) networking interfaces to model communication for stationary and non-stationary devices; (iii) resource availability in terms of capability and accessibility; and (iv) resource collaboration based on pre-defined schema and policies. In this thesis, we adopted a simulation-based experimental methodology to implement the framework services by extending the CloudSim [157] framework and adding corresponding components. This involves designing and building CoopEdgeCloudSim to simulate scheduling and execution of hybrid workflows in a cooperative edge cloud system.

This chapter describes the design and implementation aspects of the simulator, and is organized as follows. Section 6.2 discusses the workflow management system and its components and services. Section 6.3 provides details about the scheduling framework simulation and highlights the main simulator components. Framework validation is in Section 6.4 and a chapter summary is presented in Section 6.5.

6.2 Workflow Management System

Figure 6.1 presents a high level architecture of the proposed workflow management system, which includes all service layers for communicating with end users, submitting workflow applications for scheduling, and interacting with resources.

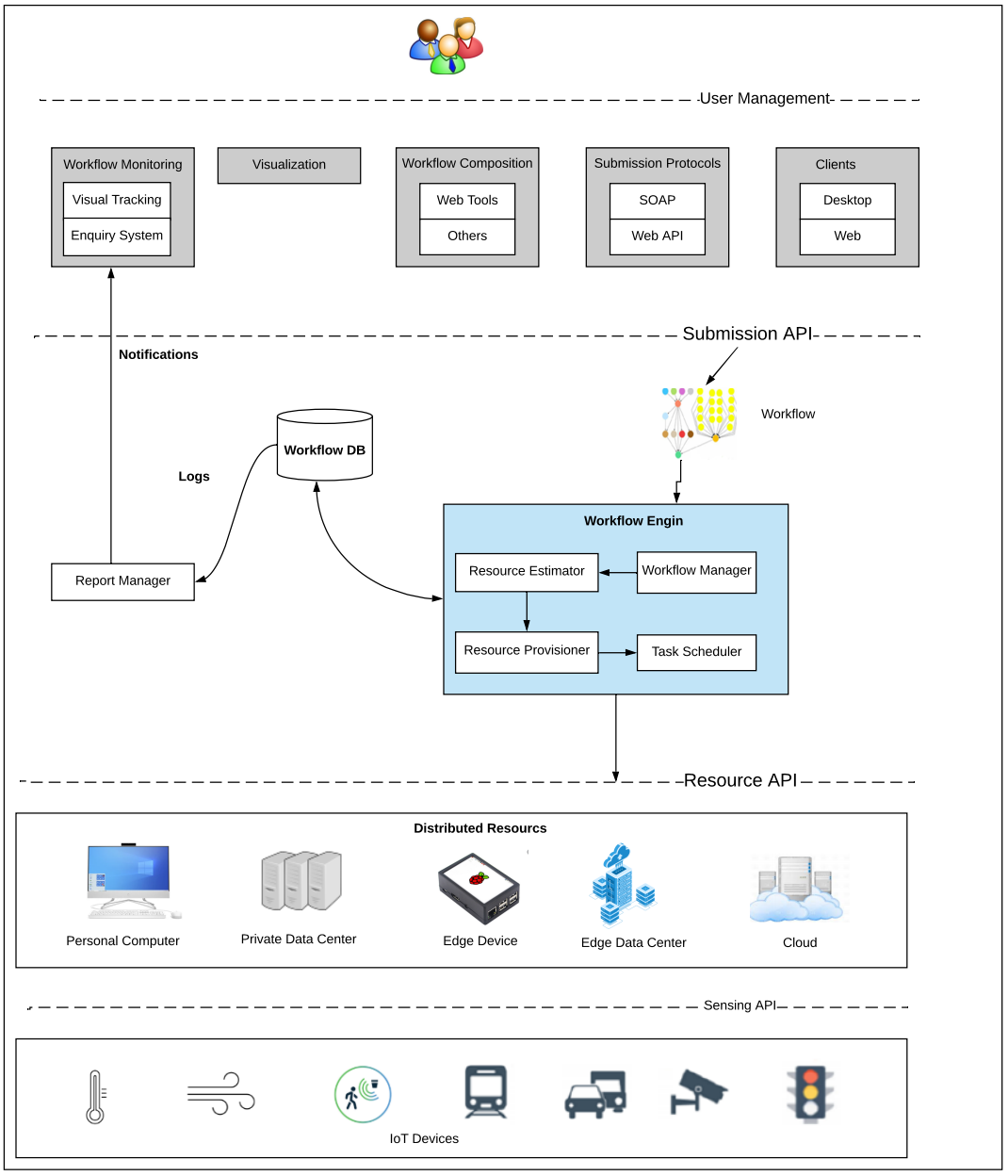

FIGURE 6.1: Hybrid Workflow Scheduling Management Architecture

6.2.1 User Management

The workflow scheduling process starts with a client submitting a scheduling request. The user management interface provides APIs to facilitate user interaction with the system in an abstract view.

Submission Service

Hybrid workflows have marginally complex structures of interactions and dependencies between stream and batch tasks. For stream applications, the user needs to identify the data sources, data rate, aggregation time interval, minimum throughput and application type. For both applications, the user should set the deadline and application type. The submission service provides desktop and web user interfaces (UIs) to accelerate the workflow creation process through offering configurable templates. For advanced users, the submission service enables writing the workflow configuration in JSON format. Figure 6.2 shows an example of stream and batch tasks definitions.

In addition, the user can compose and integrate multiple workflow applications in one submission. After submitting the workflow submission. setting-up the


```
    "_apptype":"B"
},
{
    "_id":"2",
    "_label":"A",
    "_type":"STREAM",
    "_servicrate":"0.005",
    "_aggFactor":"0",
    "_deadline":"0",
    "_throughput":"0.9",
    "_wsize":"20",
    "_srvTimedist":"UNIFORM",
    "_srvdistparam":"0.0006,0.0007",
    "_arrTimedist":"UNIFORM",
    "_arrdistparam":"0.00020,0.00021",
    "_pre":"1",
    "_post":"3,4,5",
    "_apptype":"E"
},
{
    "_id":"3",
    "_label":"B",
    "_type":"BATCH",
    "_servicrate":"0.714",
    "_aggFactor":"1500",
    "_deadline":"14",
    "_throughput":"0.9",
    "_wsize":"0",
    "_srvTimedist":"UNIFORM",
    "_srvdistparam":"",
    "_arrTimedist":"UNIFORM",
    "_arrdistparam":"",
    "_pre":"2",
    "_post":"5",
    "_apptype":"T"
},
{
    "_id":"4",
```


FIGURE 6.2: Task definition in JSON format

workflow configuration, the user can send it to the workflow execution engine for execution.

Resource Services

System manager monitors and tracks system resources. Based on the current usage level, the manager can set-up resource provisioning to reduce upfront costs. In addition, the system enables the manager to communicate with private resource owners (at the edge layer) to update the cooperation status by adding or releasing edge data centers and devices.

Workflow Monitoring Service

The system affords the client live monitoring of workflow execution. The monitoring service offers data about the ongoing cost of running the workflow, execution output status and data collection reports from input devices. Based on this data, the client can request updates to the workflow submission to reduce the cost, add additional data sources to improve output quality, or increase the number of execution iterations, etc.

6.2.2 Workflow Engine

The workflow engine provides the core workflow scheduling services, including managing workflows and performing initial compilation, estimating required resources based on client configuration, resource provisioning and allocation at cloud and edge layers, task scheduling by mapping tasks to resources, and finally running workflows.

Workflow Manager

The workflow manager receives the client workflow in XML file format and converts it to a workflow object. The conversion involves the following:

1. Building the workflow and set-up data dependencies through parsing the XML file according to the workflow template. Table 6.1 provides a description for main stream and batch properties.

TABLE 6.1: Hybrid workflow task properties

Property	**Description**	**Task Type**
Type	Task type	Stream and Batch
Arrival Rate	Stream rate at receiver	Stream
Service Time	Processing rate	Stream and Batch
Throughput	Percentage of processed data	Stream and Batch
Window size	Time length to collect streams	Stream
Deadline	Maximum execution time	Batch
Aggregation Factor	Number of data units to produce an output	Batch
Pre Tasks	Preceding tasks	Stream and Batch
Post Tasks	Succeeding tasks	Stream and Batch

2. Running and validating the queuing system for each stream task. In this thesis the execution of a stream was modelled as a $M/G/c$ queuing system [149], in which the arrivals are Markovian (modeled as a Poisson process), service times have a general distribution and there are multiple servers. Queuing system validation implies satisfying the minimum system utilization and maximum stream waiting time in the queue.

3. Workflow profiling. The workflow engine applies the GDS technique to estimate the amount of resources required and the maximum workflow execution time. The profiling process works iteratively to log workflow execution results, which include, total cost, allocated resources, execution time and a workflow configuration file. The workflow manager uses data profiles to approximate the workflow execution function, which will be used later by the resource estimator.

Resource Estimator

The resource estimator implements functions to predict the amount of resources run a hybrid workflow. Algorithm 11 shows the high-level GDS-based estimation process. The process is considered valid when every task is allocated to a group. For complex hybrid workflows, a group relaxation strategy was applied to allow adding tasks that go beyond group boundaries at a certain threshold within the interval [0.01, 0.05]. For each task, the gradient value is calculated and configuration is updated for resource estimation. Section 5.4 provides more details about

GDS-based resource estimation.

Resource Provisioner

The resource provisioner is responsible for managing computation environment resources. In this thesis we adopted a dynamic provisioning strategy to allocate required resources based on resource estimation outcomes. The edge cloud system offers three types of resources: cloud data centers, edge data centers and edge devices. The provisioner's mission is to prepare resource allocation plans based on the current status of system resources. The following functions are associated to the provisioner:

1. Resource discovery. This function is about invigilating the availability and the capacity of system resources. The discovery process can be described as follows.

   (a) An edge device can be stationary or non-stationary and it is considered available if it has: i) the minimum computation capacity in terms of free cores and memory (depending on the type of application); ii) the minimum battery energy (in the case of a non-stationary device); and iii) the minimum connectivity threshold to access edge data centers.

   (b) The provisioner communicates with the edge center gateway to collect information about data center resource availability and connectivity status with other data centers (edge and cloud) and edge devices.

Algorithm 11 GDS-based resource estimator

```
1: procedure RunEstimator
2:     while ! AllTasksGrouped do
3:         IncreaseGroupRelaxation()
4:         for each Task ∈ StreamTasks do
5:             TConfig = LoadTaskConfiguration()
6:             CalculateGradient(TConfig)
7:             UpdateTaskConfig(TConfig , Task)
8:             UpdateEstimation()
9:         AllTasksGrouped = GroupTasks(StreamTasks)
```

(c) The discovery process is only performed on authenticated devices.

2. Resource allocation planning. Various parameters determine the overall time and cost of running a workflow based on a resource provisioning plan: resource capability, network type and quality, and execution trustworthy. Thus, the provisioner will construct alternative execution plans based upon the connectivity status between edge data centers and devices.

3. Cloud resource provisioning. Along with resource allocation plans, the provisioner also provides details about cloud VMs to guide the scheduler in mapping tasks to public clouds. After deciding on mapping some tasks to the cloud, it is the provisioner's mission to allocate cloud VMs.

Task Scheduler

The task scheduler receives task groups from the resource estimator and allocation plans from the resource provisioner. Subsequently, the scheduler needs to find the optimized mapping strategy which provides the minimum of execution time and cost in a joint optimization manner. The mapping is a group-based process in which group tasks are enclosed in a bounded execution interval. This allows the scheduler to control the execution scale at group level and apply relaxation strategies.

6.3 Implementation and Prototyping

To implement the prototype of the proposed workflow management architecture, we built a simulation environment to implement all the services and functions required to achieve the desired functionalities of resource management, user communication, resource scheduling and task mapping. The environment allows the evaluation of hybrid workflow scheduling techniques and frameworks in edge cloud systems. In that matter, the CoopEdgeCloudSim is developed on the top of the CloudSim simulator [157], which is the most well-known simulator for building customized distributed systems in an event-based simulation execution.

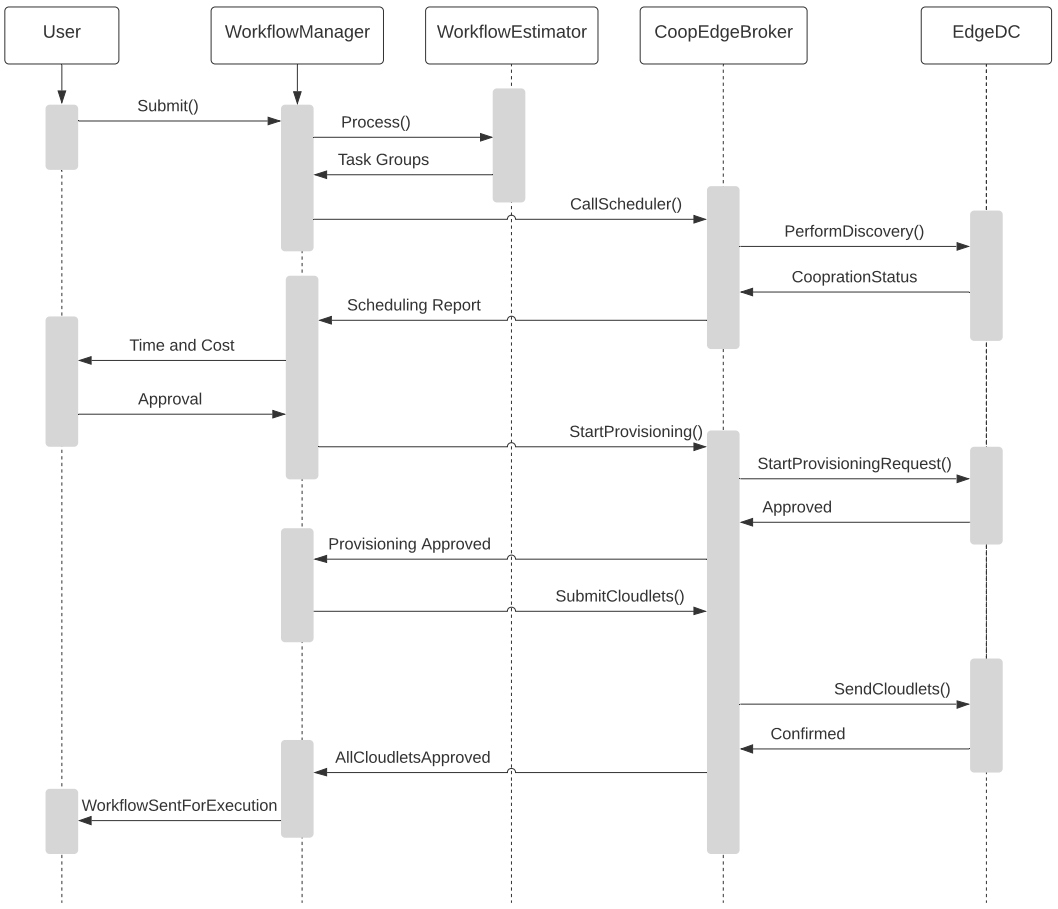

FIGURE 6.3: The sequence diagram for running a workflow

For the implementation of CoopEdgeCloudSim, we extended some of the CloudSim classes, as presented in Figure 6.4. Entities that extend the SimEntity class can exchange messages, and schedule events (SimEvent) based on pre-defined command/event tags. The class diagram hierarchy for the CoopEdgeCloudSim simulator is shown in Figure 6.5. The following section outlines the extended classes.

Workflow Manager

As mentioned earlier, the scheduling process is handled online and is started once the workflow execution request is received. The workflow manager extends the SimEntity class to allow event messaging with the resource broker (CoopEdgeDatacenterBroker) entity. The broker sends a message with "process workflow acknowledgment" tag indicating that resource discovery is finished. Once the acknowledgment is received, the workflow manager pulls the waiting workflow and

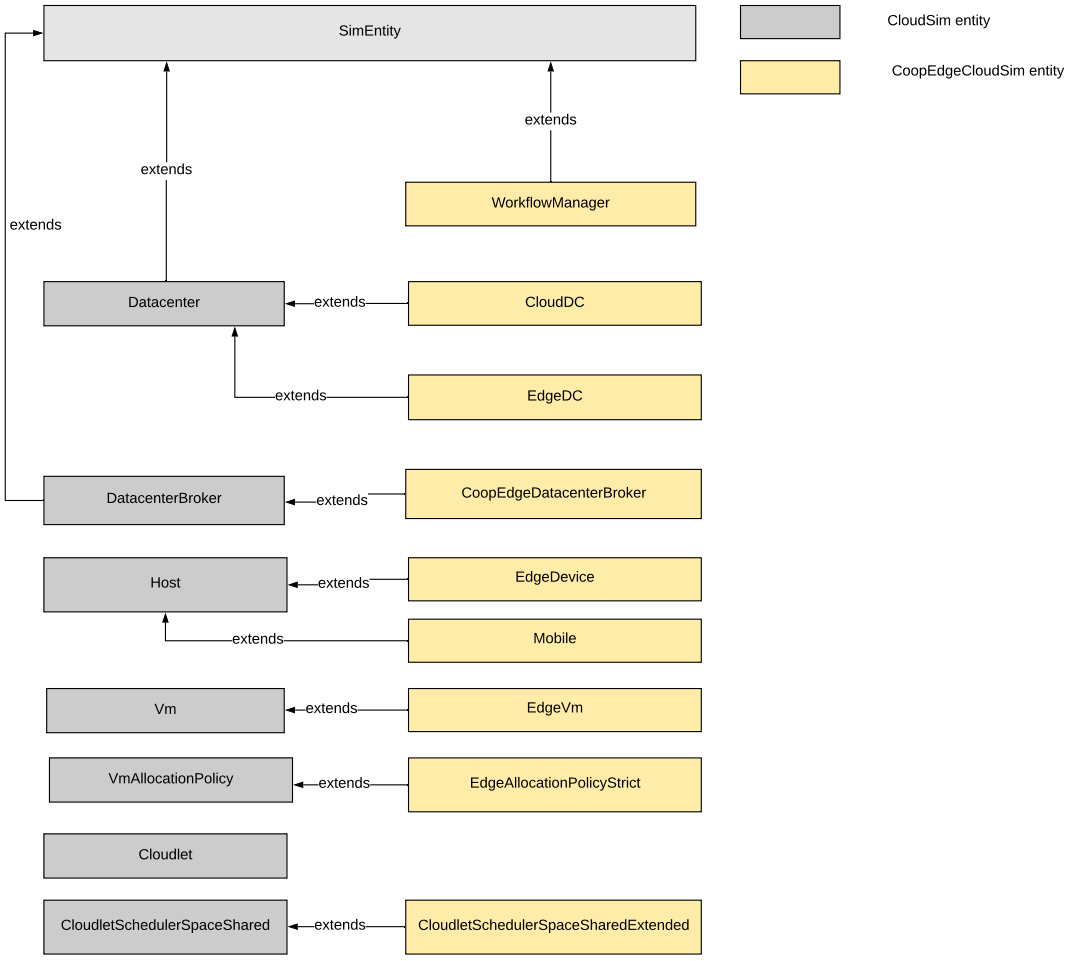

FIGURE 6.4: CoopEdgeCloudSim simulator entities

fires the estimation process to produce execution groups under constraints of queuing system validity for all stream tasks and ability to construct the critical path (CP) that encloses all tasks. Next, an allocation request is sent to the broker. Figure 6.3 shows a typical workflow execution request scenario.

CoopEdgeDatacenterBroker

The CoopEdgeDatacenterBroker class represents the user (the workflow manager) proxy needed to access the computing system services. It has a range of responsibilities to perform, including i) preparing provisioning groups and execution paths; ii) performing resource discovery and preparing execution plans; iii) calling the workflow scheduler; iv) sending the scheduling plan and the estimated cost and time to the workflow manager for user confirmation; and v) sending VM provisioning and cloudlet execution requests to data centers.

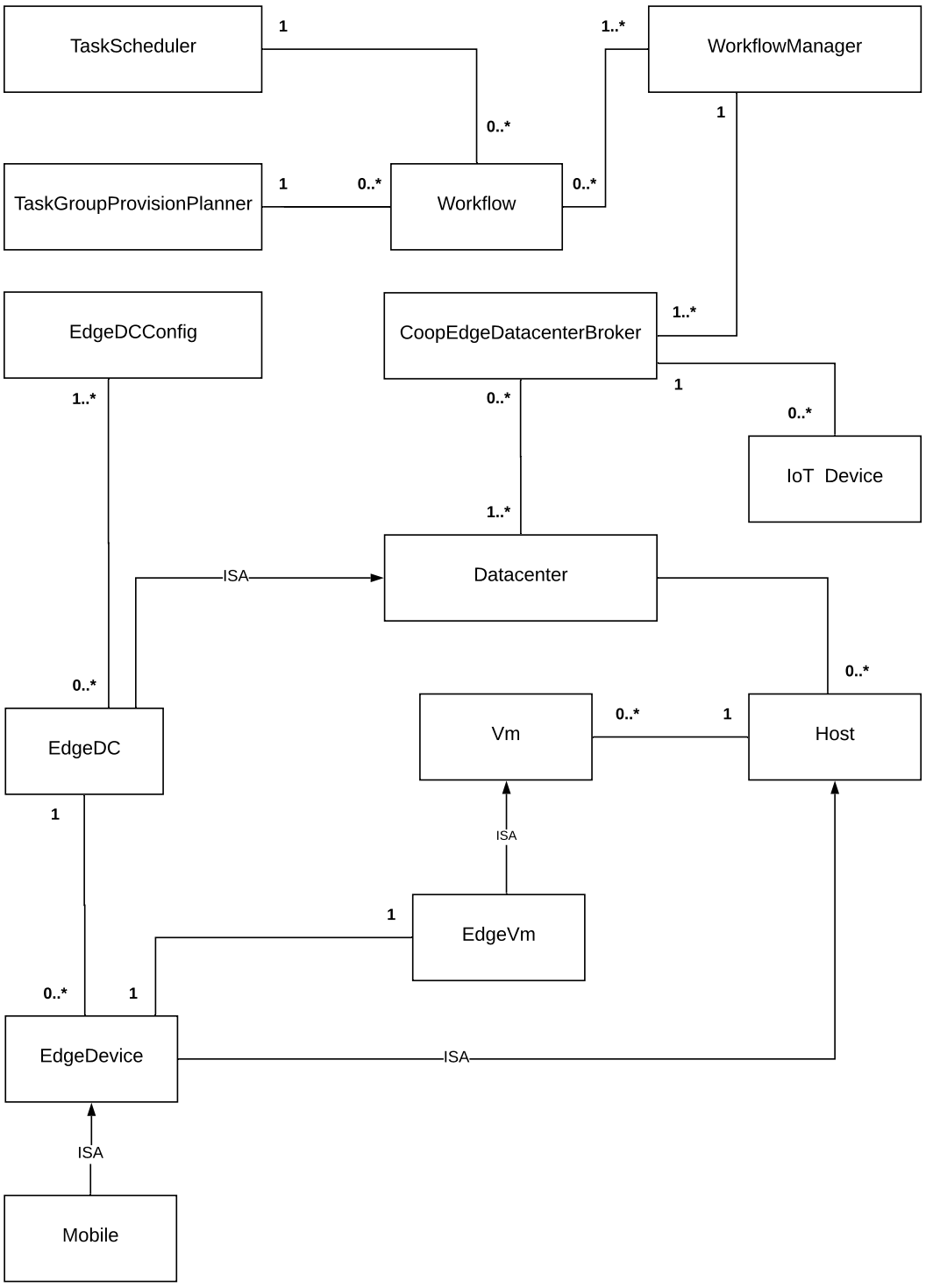

FIGURE 6.5: The class diagram hierarchy for CoopEdgeCloudSim

EdegDC

The EdgeDC class manages the functionality of the edge datacenter system. The class includes functions to handle the services for two edge datacenter types, that

is, interconnected (stationary with LAN networking) and distributed (of stationary and non-stationary) edge devices. The class provides services for resource provisioning with various allocation strategies, resource usage monitoring, facilitating access to internal assets by defining accessible service proxies and allowing tracking and monitoring for non-stationary devices (mobiles and IoT sensors) in terms of accessibility, power and energy, networking quality and location.

The EdgeDC class includes continues monitoring the connectivity status of the edge devices. This supports the resource discovery service provided by the resource broker class. To simulate the discovery process, we created "EdgeDCConfig" class that resembles the behaviour of edge resources in terms of capability, availability and cooperation with other devices.

EdgeDevice

An EdgeDevice is any accessible and verified device capable collecting and processing data streams from IoT devices and also communicating with an EdgeDC for the purposes of workload migration and data storage. The EdgeDevice class extends the Host class to allow the tracking of features like mobility, location, energy, connectivity and the cooperation network. In addition, an EdgeAllocationPolicyStrict allocation policy is applied on edge devices in a way that each device is only associated with one VM (EdgeVm).

IoTDevice

The IoTdevice class models the core features of IoT devices, and it senses, generates and sends data to the next network hop. An IoT device has the features of type (based on generated data type), communication interface and network type, stream generation rate and connectivity status.

6.4 Framework Evaluation

This section provides the validation for the simulation framework accuracy. The validation process aims to determine the framework stability in estimating the

optimization parameters, that is, the execution time and cost. For stability assessment, the validation experimentation was undertaken using a medium-scale workflow. Figure 6.6 shows the workflow structure, which consists of 20 stream and 45 batch tasks. In addition, we adopted the edge cloud resource model provided in Table 5.4. One cloud datacenter, three edge data centers and number of (application-defined) edge devices articulated were compound to form the cooperative edge cloud system.

The cooperative system is controlled by the connectivity and trustworthy among computation nodes at the edge layer. To simulate edge cooperation, a set of parameters was randomized to indicate the allowance for these nodes to migrate computation and data. Various parameters were used, such as number of edge devices, local connection, local bandwidth, enabled-network interfaces, battery-level and level of connectivity trust. This mechanism imitates the real behaviour of the collaborative edge cloud system and allows the assessment of framework stability with different cooperation states. Tables 6.2 and 6.3 present the results of running the medium-scale workflow on three computing systems and recording execution time and cost, respectively. With 95% confidence interval (CI), results demonstrate the scheduling framework stability as stated by the low measurement variance and low standard error. Furthermore, the cooperative system required higher number of iterations to attain a satisfactory level of stability. The Boxplots in Figures 6.7 and 6.8 shows how it was more difficult to produce stable estimation

Table 6.2: 95% Confidence interval validation results - execution time

Computing System	**95% CI (Minutes)**	**Std. Error**	**#Iterations**
Cooperative Edge cloud	3.04 - 3.73	0.167	28
Edge cloud	3.14 - 3.82	0.164	22
Cloud only	10.11 - 14.29	1.020	18

Table 6.3: 95% Confidence interval validation results - execution cost

Computing System	**95% CI ($)**	**Std. Error**	**#Iterations**
Cooperative Edge cloud	1.68 - 2.58	0.22	28
Edge cloud	1.91 - 2.78	0.21	22
Cloud only	9.03 - 12.11	0.68	18

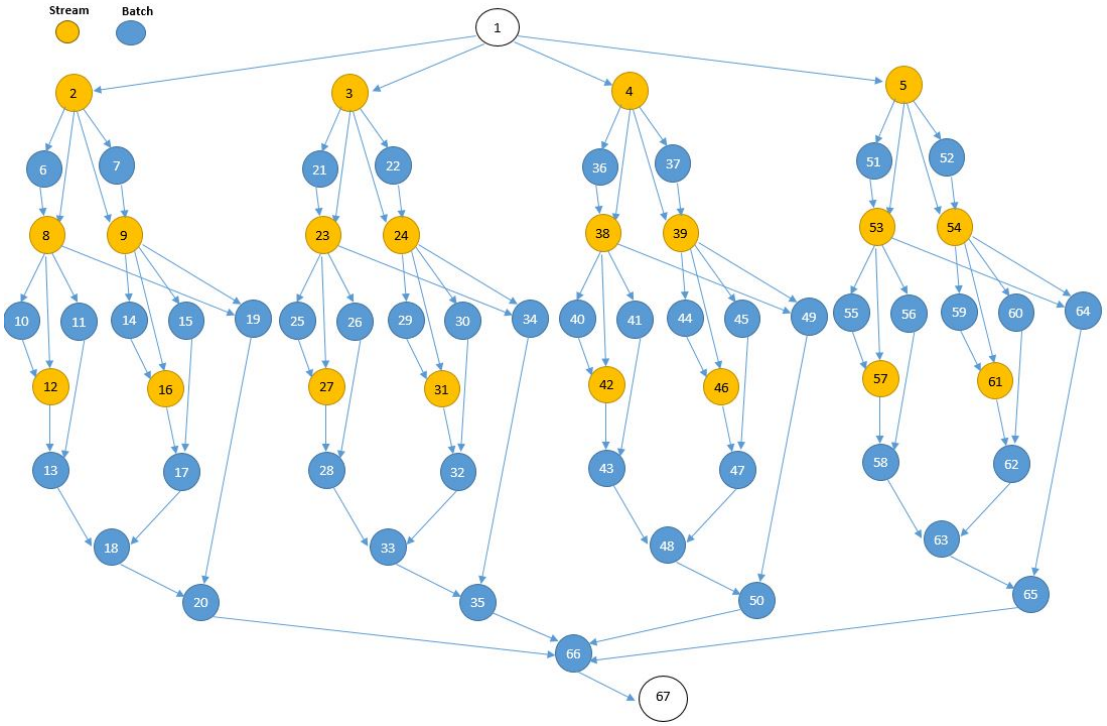

FIGURE 6.6: The sequence diagram for running a workflow

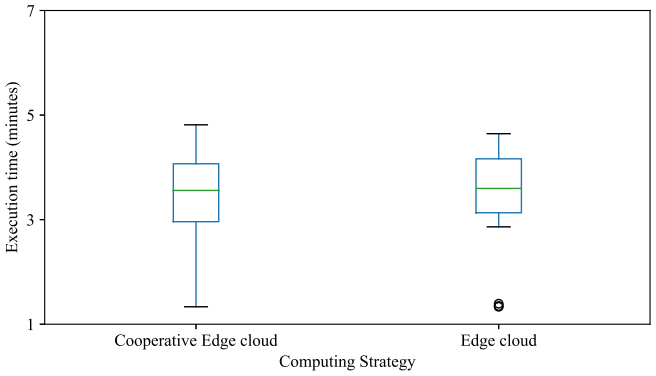

FIGURE 6.7: Execution time variation - window size scenario

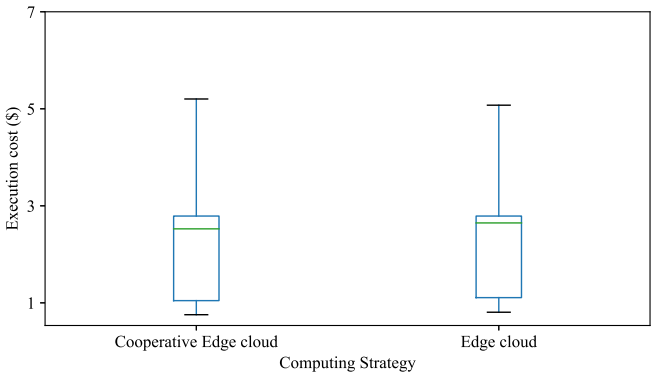

FIGURE 6.8: Execution cost variation - window size scenario

over mean values compared to uncooperative model. In conclusion, the evaluation results demonstrated a reasonable framework stability for executing hybrid workflows.

6.5 Summary

This chapter described the architecture of the proposed hybrid workflow scheduling scheduling framework. Firstly, the main components and services of the high-level scheduling management system are described. The system provides services in four levels: user interaction, workflow engine, computation resource and IoT devices. The workflow engine performs the core system services of resource estimation, resource discovery, resource provisioning and task mapping. The engine architecture is implemented as CoopEdgeCloudSim, an extension of the well-known CloudSim simulator to simulate the hybrid workflow scheduling on cloud and edge cloud environments. The CoopEdgeCloudSim simulator allows the evaluation of a variety of resource management policies (for estimation, discovery and provisioning) and task mapping techniques on different computing environments.

The framework evaluation shows stable framework behaviour with low estimation error. However, research on cooperative edge systems is still at an early stage, and there is considerable room for further improvements to enhance user satisfaction and achieve accepted levels of edge usage utilization. The next chapter outlines the main findings of the research, insights into future directions related to hybrid workflows and edge environments, and concludes the thesis.

Chapter 7

Discussion and Conclusions

This chapter discusses the main findings of research on hybrid workflow scheduling for edge cloud computing systems. The key future research directions and open research challenges are highlighted and elaborated.

7.1 Discussion

The emergent of Internet of Things is driving a search for new and sophisticated forms of service models to benefit from the engagement of a large set of objects (machines, devices, etc.) which are capable of connecting and share data with minimal human interference. Intelligent solutions are needed autonomous and reliable applications on many domains such as intelligent video surveillance, crowdsensing, health monitoring and the Internet-of-Vehicles [4]. Certainly, at large-scale and over the long term, data generated from IoT devices creates critical challenges for traditional computing systems. This data mostly feeds iterative and cyclic application services that, in turn, represent a complex structure of processing paradigms that rely on a concrete integration between real-time, near real-time and batch processing models. This thesis refers to this processing structure as a hybrid workflow. A hybrid workflow is the integration of stream and batch data processing models in one data processing pipeline. Algorithms and techniques are

proposed for hybrid workflow scheduling in different computing systems to answer two main questions: how to model an application structure that illustrates how batch tasks are communicated and collaborated respect to differences on their specifications, constraints, and structure?, and what is the contribution of the computing environment to conquer the challenges of hybrid workflow scheduling?

In chapter 1, we discussed the concept of hybrid workflows and showed the divergence between stream and batch processing with respect to service quality measurements and techniques used to achieve the satisfactory level of service. Batch processing is eager for data correctness and completeness, while stream processing can achieve the desired level of efficiency even with a high blocking rate, particularly at peak loads. To contextualize the problem of hybrid workflow scheduling, the chapter identified three core principles for the implementation of adequate resource provisioning and task scheduling frameworks and strategies: 1) controlling the flooding data generated at IoT infrastructure level, 2) proposing application-aligned resource management polices to augment the efficient usage of layered-resource models in resolving the issues of QoS measurement variation of integrated models, and 3) designing for improved stream processing performance through adopting fine-grained modelling with the emphasis on maximizing performance via tuning and controlling mechanisms.

Based on these principles, the chapter generalized the concept of hybrid workflows for applications which are: complex and large-scale with high number of integrated applications, iterative and batch-based delivery with short-term stream processing intervals, and flexible and adaptable for parameter tuning. However, research on the seamless interaction between stream and batch processing has advanced rapidly by virtue of the accelerated evolution of interactive systems and mobility-based services.

In chapter 2, a wide range of research on resource provisioning and task scheduling in cloud and edge computing systems was examined. For resource provisioning, the literature review that concluded an extensive research has been conducted on proactive resource estimation mechanisms, with most focused on achieving

cost-effective resource allocation planning strategies, particularly for interactive and unpredictable workloads. Researchers have begun to employ statistical and mathematical models for workload prediction and provisioning patterns recognition has emerged to provide more controlled resource estimation and reduce the overall service performance degradations and service level violations, decisions about resources provisioning must be taken automatically and in a timely manner. Moreover, the literature review demonstrated the high convergence of resource auto-scaling on cloud systems and how application-based approaches of service parallelization and decomposition are effective in accomplishing the monetary and responsive application execution constraints. On the other hand, resource provisioning on edge-wise systems is still in its infancy. Researchers need to propose adapted provisioning schemes that are convenient with respect to edge computing concerns of localization, data privacy and by-pass control, scalability and physical implications.

In addition, the literature review presented in chapter 2 provides a discussion of the chronological development of workflow scheduling on cloud computing. The literature includes a large number of scheduling techniques and algorithms offered as optimization solutions in contexts of best-effort, constraint-aware and QoS-based optimization aspects. The proposed techniques provide solutions for scheduling problems with majority and minority illustration for both batch and stream application workflows, and some researchers have extended the work to emerge edge computing to settle latency and data privacy issues of scheduling user-centric and data-intensive applications that enforce the engagement of IoT and user devices at large scale. Despite the successful implementation of edge computing to work closer to user and data layers, some key challenges hybrid workflow scheduling are not well addressed, and these are 1) the edge cooperation is limited to the edge data center level, and cooperation mechanisms are not well defined or structured, 2) hybrid workflow scheduling is not illustrated, and the integration between stream and batch processing is not clearly associated with the scheduling process, and 3) the impact of stream and batch processing parameters tuning has not been

studied. The thesis describes experimental work overcomes these issues on three computing systems: cloud, edge cloud, and cooperative edge cloud.

In chapter 3, a resource estimation and provisioning framework for hybrid workflows on cloud systems was proposed. The work described in the chapter was to investigate hybrid workflow modelling and its correspondence to enact optimized execution on cloud system. The idea is to construct an optimal workflow configuration plan through tuning stream task parameters. To find such a configuration, a meta-heuristics optimization technique, PSO, is adopted. The technique searches the space for a global solution that guarantees the optimal (minimum) function value with respect to number of processing cores and execution time. For resource provisioning, a group-based technique is utilized to reduce execution time and cost. The grouping approach is convenient for hybrid workflows because it is capable for controlling the execution of hybrid workflows by efficiently tuning several parameters, including stream arrival rate and processing throughput.

Evaluation results determined the technique's efficiency for adaptive task clustering and periodic resource provisioning to respond to the variation on problem variables, particularly, data generation rate, stream processing interval length and processing throughput. Furthermore, the analysis highlighted the sensitivity of the optimization objective to throughput constraints, and the necessity of building efficient tuning techniques to guarantee a reasonable margin of workflow execution optimization. Even though it has the advantage of attaining a global optimal solution, the adopted evolutionary technique has a limitation of high optimization time complexity with large-scale workflows that carry out a massive number of configuration plans, and accordingly may cause overall framework performance degradation. Moreover, although the chapter provides a proof-of-concept hybrid workflow provisioning framework, and some critical parameters were not perfectly studied thoroughly; network quality for migrating stream data to the cloud is an example. The work in chapter 4 handled these limitations through providing more advanced optimization methods which can support resource provisioning and scheduling for complex hybrid workflows in an edge cloud system.

In chapter 4, a two-stage hybrid workflow scheduling framework for edge cloud computing was proposed. The first stage is to estimate the amount of resources needed to run a hybrid workflow based on a linear optimization approach, the GDS, which maintains the same optimization objective of reducing the number of processing units in a bounded execution time limit. The linear search technique was applied for a three-dimensional workflow execution function in which each dimension represents a tuple of all workflow configuration parameter values among all stream tasks. The main challenge of applying the GDS approach was constructing the workflow execution function and confirming the usage of GDS on the function. For the purpose of technique evaluation, an offline approximation process was applied based on workflow execution profiling with consideration of variation on a range of configuration parameters.

On the second stage, the work on the chapter proposed a cluster-based provisioning and scheduling technique for hybrid workflows in heterogeneous edge cloud resources, which aims to achieve a multi-objective optimization of execution time and monetary cost under constraints of deadline and throughput. The chapter highlighted the role of edge computing in processing latency-sensitive workloads (stream tasks) closer to data sources and reduce the traffic to the cloud. To boost the efficiency of the scheduling framework, the task mapping process was conducted at group level can reduce the cumulative data migration between computation nodes, which is mostly subjects to the ability to maximize the number of tasks within a certain group. The scheduling optimizer considers the issues of load balancing, edge capacity and the data communication quality among the three resource layers (IoT, edge and cloud). The chapter provided a comparison between GDS-based (linear search optimization) and the PSO-based (metaheuristic optimization) techniques. Results demonstrate the ability of GDS to control the execution of hybrid workflows by efficiently tuning several parameters, including stream arrival rate, processing throughput and workflow complexity. The proposed scheduler shows a minimum of 35% and a maximum of 70% cost and time reduction compared to other nominated techniques. Moreover, the GDS-based technique

shows a significant reduction (50% on average) in the optimization complexity compared to the PSO technique.

The work described in this chapter had two limitations. The first is the offline examination of the workflow execution function. This is impractical for generalizing the optimization framework due to the diversity of hybrid workflow applications. The other limitation was the limited effort to utilize the capability of edge computing based on the assumption of workload migration to the cloud in case of edge failure. Thus, chapter 5 described the experimental work on hybrid workflow provisioning and scheduling for cooperative edge cloud computing, and also provides an improvement to the GDS technique to handle online resource estimation without prior knowledge about the input workflow structure and constraints. The current version of the GDS technique does not support online estimation for unknown workflow structures.

Chapter 5 described the adoption of cooperative edge cloud model to resolve the issues of latency-sensitive application as well as to improve the resource utilization at the edge layer. In addition, the chapter introduced an extended GDS-based resource estimation technique to align the requirements of unpredictable workflow structures. Research outcomes on cooperative edge computing concluded the complexity of building a resilient task scheduler that achieves a satisfactory level of fair load balancing and cost-efficient workload distribution for the cooperative model. This chapter presented a hybrid workflow scheduling framework on a cooperative edge cloud computing resource model. The cooperation implies communication between edge resources for computation and data migration under constraints of resource capability and data dependency.

The chapter provided comparison of the cooperative edge-cloud model with other two resource models, namely, non-cooperative edge-cloud and cloud-only. The cooperative model showed a significant cost reduction of 40% compared to the non-cooperative model. In addition, results demonstrated the cooperative model's capability to maximize data migration within the layer, and accordingly reduce data transfer to the cloud layer. Furthermore, in this chapter we improved the

optimization approach, GDS, such that it is able to perform online estimation regardless of workflow structure.

In chapters 3, 4 and 5, the thesis provides different algorithms, techniques, frameworks to handle resource estimation and provisioning, and task mapping in context of hybrid workflows. As we stated, this is a simulation-based experimental which relies on verified data profiling for hybrid workflow task types. According to results, the cooperative model showed a significant cost reduction with 40% compared to the non-cooperative model. In addition, results demonstrated the cooperative model capability to maximize the data migration within the layer, and accordingly reduce data transfer to cloud layer.

Chapter 6 provided a detailed description of the proposed framework for hybrid workflow scheduling in edge cloud computing systems. The framework of an edge cloud computing deployment architecture and its prototype, 'CoopEdgeCloudSim' which is implemented by extending the classes of CloudSim simulator in order to simulate resource provisioning and hybrid workflow scheduling framework in edge cloud system. To provide more realistic awareness to the simulation and modelling of IoT-enabled applications on edge cloud environment.

7.2 Future Directions

The research in this thesis addressed the problem of hybrid workflow scheduling on various computing systems. However, there are still some research challenges and open research areas that need further exploration.

7.2.1 Statistical approaches for Gradient Descent Search optimization for hybrid workflows

Hybrid workflows are applied in a wide range of applications that mostly involve the integration of complex structure of applications and services. In this thesis,

we experimented with the application of a GDS technique to reduce the complexity of searching workflow execution plans at large-scale. The main issue is to guarantee the best convergence and correctness of the quantized algorithm with variable workflow topologies [175]. Online resource estimation performance may suffer from variation in workflow models and the effects of unpredictable user behaviour on setting QoS measurements. The solution provided in this thesis is to perform an online workflow execution profiling to get the best execution function approximation. This does not seem a practical way to meet the mentioned challenges.

Utilizing statistical methods and learning algorithms could help future researchers to train estimation models for predicting workflow execution behaviour, and accordingly improves the estimations. For example, using Stochastic Gradient Descent [176] as a lightweight optimizer is likely to be a promising research direction for reducing online resource estimation complexity.

7.2.2 SLA-based optimization solutions in cooperative edge context

The cooperative edge system is a computing environment in which local devices can participate to benefit from the access to collaborative system services in terms of privacy preservation through edge-controlled user sensitive data processing, service reliability for critical applications like self-driving and health monitoring, and tailored context-aware services upon user location and data usage preferences. The management of user SLA should ensure a satisfactory level of user acceptance for data usage and context-awareness without violating the overall system performance. For example, Social Sensing based Edge Computing (SSEC) involves processing data collected from privately owned devices. SSEC manages privacy without a tradeoff with service performance or accuracy [177].

7.2.3 Towards decentralized service management solutions in edge cloud systems

Edge cloud computing enables communication/integration between centralized cloud systems and highly distributed edge devices which are used for computation and data storage to address the data's wide dispersion. Edge resources are heterogeneous, highly dispersed, and loosely coupled. Edge cloud computing is resource-decentralized computing where large data centers distribute computation and workloads towards the edge of the network closer to the end-users and sensors [178]. Research on data-based edge cloud architectures for workload distribution is still in its infancy. Further research is required to propose control architectures to manage how edge resources are controlled in the computing ecosystem. A distributed control approach with multi-service controllers is a promising research direction for more secure and reliable localized edge-based services. Localized-service controllers have many real-world applications in object-detection and geolocation tracking services [179, 180]. This thesis adopted a single and centralized service controller to manage and coordinate the communication between edge devices in the computing system. The massive and growing number of edge devices means their the coordination a bottleneck for service automation and resource collaboration. A future research direction is to develop techniques that provide dynamic, progressive and localized edge cloud control management to reduce communication costs and improve system repressiveness.

7.2.4 Reliable computing in edge cloud systems: a service-oriented approach

In this thesis, a hybrid workflow scheduling is defined as the management of QoS optimization for short-term and iterative application executions with hard-constrained of waiting time and throughput. Short-term service delivery workflows allow dynamic systems to adapt their behaviour. For example, short-term forecasting in transportation systems allows producing alternative routes to avoid traffic

congestion before gridlock. Practically, the realtime application processing, like online forecasting, is prone to failure for several reasons such as communication failure and insufficient resource capability. Considering heterogeneous resources is essential to achieve reliable stream processing in the edge cloud system. Reliable resource management for heterogeneous service-oriented systems has been widely identified as a critical research challenge in reference to the devastating contribution on violating user SLA caused by respect to processor and communication failures. In a nutshell, reliability indicates the likelihood of successfully completing of workflow execution [181]. Recently, many approaches have proposed to enhance service-based reliability by reducing application recovery time after failure. The work on chapter 5 involved the adoption of resource discovery mechanism to rank and select edge resources to be engaged in the cooperative edge cloud computing system based on computing capability and connectivity quality.

Reliable-based hybrid workflow scheduling is not addressed in this thesis. To ensure the quality of edge collaboration services and trusted behavior of involved devices, a trust mechanism can be developed to assess the trustworthy and the reliability assurance at service and device levels [182]. In addition, a reliable-based resource discovery strategy can be further investigated to traverse and predict possible failure cases of workflow task execution on edge nodes, this is refereed to designing node selection algorithm [183].

7.3 Summary

This chapter concluded the work on this thesis and provided insights on the main outcomes of research and development of techniques and algorithms for hybrid workflow scheduling on various computing systems. This thesis highlighted the foundation of hybrid workflow concept as a high level abstraction of combining heterogeneous application models, i.e., stream and batch, which are different in

computation constraints and service quality measurements. The chapter summarized the research work of proposing resource estimation and provisioning techniques that consider the complexity of hybrid workflow models and how the cost of resources usage can be reduced. In addition, the chapter highlighted the research provided to build scheduling techniques that benefit from the availability of sophisticated computing systems like edge cloud and cooperative edge cloud to optimize workflow execution and meet user expectations for low cost and high throughput execution outcomes. Moreover, the simulation framework which was developed to host the scheduling engine was described. Finally, this chapter highlighted and suggested some future research directions towards hybrid workflow scheduling and edge computing.

Bibliography

[1] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. Riotbench: An iot benchmark for distributed stream processing systems. *Concurrency and Computation: Practice and Experience*, 29(21):e4257, 2017.

[2] Marcos D Assunção, Rodrigo N Calheiros, Silvia Bianchi, Marco AS Netto, and Rajkumar Buyya. Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79:3–15, 2015.

[3] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[4] Cisco Visual Networking Index. global mobile data traffic forecast update, 2017–2022. *Cisco White paper*, 2019.

[5] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, pages 1165–1188, 2012.

[6] Wullianallur Raghupathi and Viju Raghupathi. Big data analytics in healthcare: promise and potential. *Health information science and systems*, 2(1): 3, 2014.

[7] Doug Laney. 3d data management: Controlling data volume, velocity and variety. *META group research note*, 6(70):1, 2001.

[8] Andrea De Mauro, Marco Greco, and Michele Grimaldi. A formal definition of big data based on its essential features. *Library Review*, 2016.

[9] Xiaolong Jin, Benjamin W Wah, Xueqi Cheng, and Yuanzhuo Wang. Significance and challenges of big data research. *Big Data Research*, 2(2):59–64, 2015.

[10] James Manyika. Big data: The next frontier for innovation, competition, and productivity. *http://www. mckinsey. com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation*, 2011.

[11] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data: the management revolution. *Harvard business review*, 90(10):60–68, 2012.

[12] Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. Toward scalable systems for big data analytics: A technology tutorial. *IEEE access*, 2:652–687, 2014.

[13] Nesime Tatbul. Streaming data integration: Challenges and opportunities. In *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, pages 155–158. IEEE, 2010.

[14] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 3(2):119–153, 1995.

[15] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493, 2015.

[16] Yan Yao, Jian Cao, Shiyou Qian, and Shanshan Feng. Decentralized executions of privacy awareness data analytics workflows in the cloud. *Concurrency and Computation: Practice and Experience*, 31(15):e5063, 2019.

[17] Peter Mell, Tim Grance, et al. The nist definition of cloud computing.(2011). *NIST special publication*, 800:145, 2011.

[18] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[19] Jyoti Sahni and Deo Prakash Vidyarthi. A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. *IEEE Transactions on Cloud Computing*, 6(1):2–18, 2015.

[20] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5): 637–646, 2016.

[21] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.

[22] Weisong Shi, Hui Sun, Jie Cao, Quan Zhang, and Wei Liu. Edge computing-an emerging computing model for the internet of everything era. *Journal of Computer Research and Development*, 54(5):907–924, 2017.

[23] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5):1125–1142, 2017.

[24] Bing Lin, Fangning Zhu, Jianshan Zhang, Jiaqing Chen, Xing Chen, Naixue N Xiong, and Jaime Lloret Mauri. A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing. *IEEE Transactions on Industrial Informatics*, 15(7):4254–4265, 2019.

[25] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh 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. ACM, 2012.

[26] Amir Vahid Dastjerdi and Rajkumar Buyya. Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116, 2016.

[27] Matú Harvan, Thomas Locher, and Ana Claudia Sima. Cyclone: Unified stream and batch processing. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, pages 220–229. Ieee, 2016.

[28] Nader Mohamed and Jameela Al-Jaroodi. Real-time big data analytics: Applications and challenges. In *2014 international conference on high performance computing & simulation (HPCS)*, pages 305–310. IEEE, 2014.

[29] David Hollingsworth et al. The workflow reference model: 10 years on. In *Fujitsu Services, UK; Technical Committee Chair of WfMC*. Citeseer, 2004.

[30] Adam Barker and Jano Van Hemert. Scientific workflow: a survey and research directions. In *International Conference on Parallel Processing and Applied Mathematics*, pages 746–753. Springer, 2007.

[31] Ustun Yildiz, Adnene Guabtni, and Anne HH Ngu. Business versus scientific workflows: A comparative study. In *2009 Congress on Services-I*, pages 340–343. IEEE, 2009.

[32] Dália Loureiro, Conceição Amado, André Martins, Diogo Vitorino, Aisha Mamade, and Sérgio Teixeira Coelho. Water distribution systems flow monitoring and anomalous event detection: A practical approach. *Urban Water Journal*, 13(3):242–252, 2016.

[33] Nada Chendeb Taher, Imane Mallat, Nazim Agoulmine, and Nour El-Mawass. An iot-cloud based solution for real-time and batch processing of big data: Application in healthcare. In *2019 3rd International Conference on Bio-engineering for Smart Technologies (BioSMART)*, pages 1–8. IEEE, 2019.

[34] RM Tyburski. A review of road sensor technology for monitoring vehicle traffic. *ITE (Institute of Transportation Engineers) Journal;(USA)*, 59(8), 1988.

[35] Michel Ferreira, Ricardo Fernandes, Hugo Conceiçao, Pedro Gomes, Pedro M d'Orey, Luís Moreira-Matias, Joao Gama, Fernanda Lima, and Luís Damas. Vehicular sensing: Emergence of a massive urban scanner. In *International Conference on Sensor Systems and Software*, pages 1–14. Springer, 2012.

[36] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[37] Alain Biem, Eric Bouillet, Hanhua Feng, Anand Ranganathan, Anton Riabov, Olivier Verscheure, Haris Koutsopoulos, and Carlos Moran. Ibm infosphere streams for scalable, real-time, intelligent transportation services. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1093–1104, 2010.

[38] Zhiyi Li, Reida Al Hassan, Mohammad Shahidehpour, Shay Bahramirad, and Amin Khodaei. A hierarchical framework for intelligent traffic management in smart cities. *IEEE Transactions on Smart Grid*, 10(1):691–701, 2017.

[39] Itorobong S Udoh and Gerald Kotonya. Developing iot applications: challenges and frameworks. *IET Cyber-Physical Systems: Theory & Applications*, 3(2):65–72, 2018.

[40] Kanwardeep Singh Ahluwalia. Scalability design patterns. In *Proceedings of the 14th Conference on Pattern Languages of Programs*, pages 1–8, 2007.

[41] Marcos Dias de Assuncao, Alexandre da Silva Veith, and Rajkumar Buyya. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, 103:1–17, 2018.

[42] Dawei Sun and Rui Huang. A stable online scheduling strategy for real-time stream computing over fluctuating big data streams. *IEEE Access*, 4: 8593–8607, 2016.

[43] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. Unlocking the potential of the internet of things. *McKinsey Global Institute*, 2015.

[44] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Workflow scheduling algorithms for grid computing. In *Metaheuristics for scheduling in distributed computing environments*, pages 173–214. Springer, 2008.

[45] AR Arunarani, D Manjula, and Vijayan Sugumaran. Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems*, 91:407–415, 2019.

[46] Mohit Kumar, SC Sharma, Anubhav Goel, and SP Singh. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, 2019.

[47] C Nandhakumar and K Ranjithprabhu. Heuristic and meta-heuristic workflow scheduling algorithms in multi-cloud environments—a survey. In *2015 International Conference on Advanced Computing and Communication Systems*, pages 1–5. IEEE, 2015.

[48] Ju Ren, Hui Guo, Chugui Xu, and Yaoxue Zhang. Serving at the edge: A scalable iot architecture based on transparent computing. *IEEE Network*, 31(5):96–105, 2017.

[49] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing ({ICAC} 13)*, pages 23–27, 2013.

[50] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12(4):559–592, 2014.

[51] Sara Kardani Moghaddam, Rajkumar Buyya, and Kotagiri Ramamohanarao. Performance-aware management of cloud resources: A taxonomy and future directions. *ACM Computing Surveys (CSUR)*, 52(4):1–37, 2019.

[52] Rafael Moreno-Vozmediano, Rubén S Montero, Eduardo Huedo, and Ignacio M Llorente. Efficient resource provisioning for elastic cloud services based on machine learning techniques. *Journal of Cloud Computing*, 8(1):5, 2019.

[53] Ming Mao and Marty Humphrey. Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 67–78. IEEE, 2013.

[54] Malawski Maciej. Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press*, 2012.

[55] Qingchen Zhang, Laurence T Yang, Zheng Yan, Zhikui Chen, and Peng Li. An efficient deep learning model to predict cloud workload for industry informatics. *IEEE transactions on industrial informatics*, 14(7):3170–3178, 2018.

[56] Ali Yadavar Nikravesh, Samuel A Ajila, and Chung-Horng Lung. Towards an autonomic auto-scaling prediction system for cloud resource provisioning. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 35–45. IEEE, 2015.

[57] Neal Wagner and Zbigniew Michalewicz. An analysis of adaptive windowing for time series forecasting in dynamic environments: further tests of the dyfor gp model. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1657–1664, 2008.

[58] Mahmood Deypir, Mohammad Hadi Sadreddini, and Sattar Hashemi. Towards a variable size sliding window model for frequent itemset mining over data streams. *Computers & industrial engineering*, 63(1):161–172, 2012.

[59] Daniel Warneke and Odej Kao. Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *IEEE transactions on parallel and distributed systems*, 22(6):985–997, 2011.

[60] Yanling Shao, Chunlin Li, Jinguang Gu, Jing Zhang, and Youlong Luo. Efficient jobs scheduling approach for big data applications. *Computers & Industrial Engineering*, 117:249–261, 2018.

[61] Teng Li, Jian Tang, and Jielong Xu. A predictive scheduling framework for fast and distributed stream data processing. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 333–338. IEEE, 2015.

[62] Dazhao Cheng, Yuan Chen, Xiaobo Zhou, Daniel Gmach, and Dejan Milojicic. Adaptive scheduling of parallel jobs in spark streaming. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[63] Kaiyue Wu, Ping Lu, and Zuqing Zhu. Distributed online scheduling and routing of multicast-oriented tasks for profit-driven cloud computing. *IEEE Communications Letters*, 20(4):684–687, 2016.

[64] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[65] Yu-Kwong Kwok and Ishfaq Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE transactions on parallel and distributed systems*, 7(5):506–521, 1996.

[66] Hamid Arabnejad and Jorge G Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, 2013.

[67] Luiz F Bittencourt, Rizos Sakellariou, and Edmundo RM Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time

algorithm. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 27–34. IEEE, 2010.

[68] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick HJ Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158–169, 2013.

[69] Sukhpal Singh and Inderveer Chana. A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of grid computing*, 14(2): 217–264, 2016.

[70] Lingfang Zeng, Bharadwaj Veeravalli, and Xiaorong Li. Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pages 534–541. IEEE, 2012.

[71] Vahid Arabnejad, Kris Bubendorfer, and Bryan Ng. Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources. *Future Generation Computer Systems*, 75:348–364, 2017.

[72] Xiangyu Lin and Chase Qishi Wu. On scientific workflow scheduling in clouds under budget constraint. In *2013 42nd International Conference on Parallel Processing*, pages 90–99. IEEE, 2013.

[73] Hamid Mohammadi Fard, Radu Prodan, Juan Jose Durillo Barrionuevo, and Thomas Fahringer. A multi-objective approach for workflow scheduling in heterogeneous environments. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 300–309. IEEE, 2012.

[74] Kahina Bessai, Samir Youcef, Ammar Oulamara, Claude Godart, and Selmin Nurcan. Bi-criteria workflow tasks allocation and scheduling in cloud computing environments. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 638–645. IEEE, 2012.

[75] Amandeep Verma and Sakshi Kaushal. Bi-criteria priority based particle swarm optimization workflow scheduling algorithm for cloud. In *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, pages 1–6. IEEE, 2014.

[76] Zong-Gan Chen, Zhi-Hui Zhan, Ying Lin, Yue-Jiao Gong, Tian-Long Gu, Feng Zhao, Hua-Qiang Yuan, Xiaofeng Chen, Qing Li, and Jun Zhang. Multiobjective cloud workflow scheduling: A multiple populations ant colony system approach. *IEEE transactions on cybernetics*, 49(8):2912–2926, 2018.

[77] Fuhui Wu, Qingbo Wu, and Yusong Tan. Workflow scheduling in cloud: a survey. *The Journal of Supercomputing*, 71(9):3373–3418, 2015.

[78] Wei-Neng Chen and Jun Zhang. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(1):29–43, 2008.

[79] Juan J Durillo, Hamid Mohammadi Fard, and Radu Prodan. Moheft: A multi-objective list-based method for workflow scheduling. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 185–192. IEEE, 2012.

[80] Xiumin Zhou, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei, and Shiyan Hu. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft. *Future Generation Computer Systems*, 93:278–289, 2019.

[81] Annie S Wu, Han Yu, Shiyuan Jin, K-C Lin, and Guy Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on parallel and distributed systems*, 15(9):824–834, 2004.

[82] Zhangjun Wu, Zhiwei Ni, Lichuan Gu, and Xiao Liu. A revised discrete particle swarm optimization for cloud workflow scheduling. In *2010 International Conference on Computational Intelligence and Security*, pages 184–188. IEEE, 2010.

[83] Maria Alejandra Rodriguez and Rajkumar Buyya. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE transactions on cloud computing*, 2(2):222–235, 2014.

[84] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 400–407. IEEE, 2010.

[85] Saurabh Bilgaiyan, Santwana Sagnika, and Madhabananda Das. Workflow scheduling in cloud computing environment using cat swarm optimization. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 680–685. IEEE, 2014.

[86] Shu-Chuan Chu, Pei-Wei Tsai, et al. Computational intelligence based on the behavior of cats. *International Journal of Innovative Computing, Information and Control*, 3(1):163–173, 2007.

[87] B Arun Kumar and T Ravichandran. Time and cost optimization algorithm for scheduling multiple workflows in hybrid clouds. *European Journal of Scientific Research*, 89(2):265–275, 2012.

[88] Maciej Malawski, Kamil Figiela, Marian Bubak, Ewa Deelman, and Jarek Nabrzyski. Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. *Scientific Programming*, 2015, 2015.

[89] Andrey Kashlev and Shiyong Lu. A system architecture for running big data workflows in the cloud. In *2014 IEEE International Conference on Services Computing*, pages 51–58. IEEE, 2014.

[90] Li Liu, Miao Zhang, Yuqing Lin, and Liangjuan Qin. A survey on workflow management and scheduling in cloud computing. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 837–846. IEEE, 2014.

[91] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[92] Aravind Mohan, Mahdi Ebrahimi, Shiyong Lu, and Alexander Kotov. Scheduling big data workflows in the cloud under budget constraints. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2775–2784. IEEE, 2016.

[93] Sadegh Mirshekarian and Dušan N Šormaz. Correlation of job-shop scheduling problem features with scheduling efficiency. *Expert Systems with Applications*, 62:131–147, 2016.

[94] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, pages 1–13, 2012.

[95] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *2010 IEEE international symposium on parallel & distributed processing (IPDPS)*, pages 1–12. IEEE, 2010.

[96] Ann L Chervenak, David E Smith, Weiwei Chen, and Ewa Deelman. Integrating policy with scientific workflow management for data-intensive applications. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 140–149. IEEE, 2012.

[97] Toktam Ghafarian and Bahman Javadi. Cloud-aware data intensive workflow scheduling on volunteer computing systems. *Future Generation Computer Systems*, 51:87–97, 2015.

[98] Nimal Nissanke. *Realtime systems*. Prentice-Hall, Inc., 1997.

[99] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[100] Klavdiya Bochenina, Nikolay Butakov, Alexey Dukhanov, and Denis Nasonov. A clustering-based approach to static scheduling of multiple workflows with soft deadlines in heterogeneous distributed systems. *Procedia Computer Science*, 51:2827–2831, 2015.

[101] Shivaram Venkataraman, Aurojit Panda, Kay Ousterhout, Michael Armbrust, Ali Ghodsi, Michael J Franklin, Benjamin Recht, and Ion Stoica. Drizzle: Fast and adaptable stream processing at scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 374–389, 2017.

[102] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *Presented as part of the*, 2012.

[103] Boyang Peng, Mohammad Hosseini, Zhihao Hong, Reza Farivar, and Roy Campbell. R-storm: Resource-aware scheduling in storm. In *Proceedings of the 16th Annual Middleware Conference*, pages 149–161, 2015.

[104] Leila Eskandari, Zhiyi Huang, and David Eyers. P-scheduler: adaptive hierarchical scheduling in apache storm. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–10, 2016.

[105] Chen Meng-Meng, Zhuang Chuang, Li Zhao, and Xu Ke-Fu. A task scheduling approach for real-time stream processing. In *2014 International Conference on Cloud Computing and Big Data*, pages 160–167. IEEE, 2014.

[106] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26. IEEE, 2016.

[107] Jie Cao, Quan Zhang, and Weisong Shi. Challenges and opportunities in edge computing. In *Edge Computing: A Primer*, pages 59–70. Springer, 2018.

[108] Karim Kanoun, Martino Ruggiero, David Atienza, and Mihaela Van Der Schaar. Low power and scalable many-core architecture for big-data stream computing. In *2014 IEEE Computer Society Annual Symposium on VLSI*, pages 468–473. IEEE, 2014.

[109] Per-Olov Östberg, James Byrne, Paolo Casari, Philip Eardley, Antonio Fernandez Anta, Johan Forsman, John Kennedy, Thang Le Duc, Manuel Noya Marino, Radhika Loomba, et al. Reliable capacity provisioning for distributed cloud/edge/fog computing applications. In *2017 European conference on networks and communications (EuCNC)*, pages 1–6. IEEE, 2017.

[110] Vincenzo Scoca, Atakan Aral, Ivona Brandic, Rocco De Nicola, and Rafael Brundo Uriarte. Scheduling latency-sensitive applications in edge computing. In *Closer*, pages 158–168, 2018.

[111] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, Wenlu Hu, Wolfgang Richter, and Padmanabhan Pillai. Cloudlets: at the leading edge of mobile-cloud convergence. In *6th International Conference on Mobile Computing, Applications and Services*, pages 1–9. IEEE, 2014.

[112] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A Rahmanian. Resource management approaches in fog computing: a comprehensive review. *Journal of Grid Computing*, pages 1–42, 2019.

[113] Fei Yin, Xinjia Li, Xin Li, and Yize Li. Task scheduling for streaming applications in a cloud-edge system. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, pages 105–114. Springer, 2019.

[114] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[115] Yan Sun, Fuhong Lin, and Haitao Xu. Multi-objective optimization of resource scheduling in fog computing using an improved nsga-ii. *Wireless Personal Communications*, 102(2):1369–1385, 2018.

[116] Dadmehr Rahbari and Mohsen Nickray. Scheduling of fog networks with optimized knapsack by symbiotic organisms search. In *2017 21st Conference of Open Innovations Association (FRUCT)*, pages 278–283. IEEE, 2017.

[117] Ruilong Deng, Rongxing Lu, Chengzhe Lai, Tom H Luan, and Hao Liang. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet of Things Journal*, 3(6):1171–1181, 2016.

[118] Xuan-Qui Pham and Eui-Nam Huh. Towards task scheduling in a cloud-fog computing system. In *2016 18th Asia-Pacific network operations and management symposium (APNOMS)*, pages 1–4. IEEE, 2016.

[119] Arkadiusz Madej, Nan Wang, Nikolaos Athanasopoulos, Rajiv Ranjan, and Blesson Varghese. Priority-based fair scheduling in edge computing. *arXiv preprint arXiv:2001.09070*, 2020.

[120] Ranesh Kumar Naha, Saurabh Garg, Andrew Chan, and Sudheer Kumar Battula. Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment. *Future Generation Computer Systems*, 104:131–141, 2020.

[121] Zhi Zhou, Xu Chen, Weigang Wu, Di Wu, and Junshan Zhang. Predictive online server provisioning for cost-efficient iot data streaming across collaborative edges. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 321–330. ACM, 2019.

[122] Aymen Abdullah Alsaffar, Hung Phuoc Pham, Choong-Seon Hong, Eui-Nam Huh, and Mohammad Aazam. An architecture of iot service delegation and resource allocation based on collaboration between fog and cloud computing. *Mobile Information Systems*, 2016, 2016.

[123] Yanling Shao, Chunlin Li, and Hengliang Tang. A data replica placement strategy for iot workflows in collaborative edge and cloud environments. *Computer Networks*, 148:46–59, 2019.

[124] Changchun Long, Yang Cao, Tao Jiang, and Qian Zhang. Edge computing framework for cooperative video processing in multimedia iot systems. *IEEE Transactions on Multimedia*, 20(5):1126–1139, 2017.

[125] Daniel Zhang, Yue Ma, Chao Zheng, Yang Zhang, X Sharon Hu, and Dong Wang. Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 243–259. IEEE, 2018.

[126] Weiwei Chen and Ewa Deelman. Integration of workflow partitioning and resource provisioning. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 764–768. IEEE, 2012.

[127] Sergej Svorobej, Patricia Takako Endo, Malika Bendechache, Christos Filelis-Papadopoulos, Konstantinos M Giannoutakis, George A Gravvanis, Dimitrios Tzovaras, James Byrne, and Theo Lynn. Simulating fog and edge computing scenarios: An overview and research challenges. *Future Internet*, 11(3):55, 2019.

[128] Roberto Beraldi, Abderrahmen Mtibaa, and Hussein Alnuweiri. Cooperative load balancing scheme for edge computing resources. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 94–100. IEEE, 2017.

[129] Lixing Chen, Sheng Zhou, and Jie Xu. Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. *IEEE/ACM Transactions on Networking*, 26(4):1619–1632, 2018.

[130] Faheem Zafari, Prithwish Basu, Kin K Leung, Jian Li, Ananthram Swami, and Don Towsley. Resource sharing in the edge: A distributed bargaining-theoretic approach. *arXiv preprint arXiv:2001.04229*, 2020.

[131] Gary White and Siobhan Clarke. Short-term qos forecasting at the edge for reliable service applications. *IEEE Transactions on Services Computing*, 2020.

[132] Archan Misra, Marion Blount, Anastasios Kementsietsidis, Daby Sow, and Min Wang. Advances and challenges for scalable provenance in stream processing systems. In *International Provenance and Annotation Workshop*, pages 253–265. Springer, 2008.

[133] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pages 42–47. IEEE, 2013.

[134] Rubén Casado and Muhammad Younas. Emerging trends and technologies in big data processing. *Concurrency and Computation: Practice and Experience*, 27(8):2078–2091, 2015.

[135] Altti Ilari Maarala, Mika Rautiainen, Miikka Salmi, Susanna Pirttikangas, and Jukka Riekki. Low latency analytics for streaming traffic data with apache spark. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2855–2858. IEEE, 2015.

[136] Stefan Bosse and Uwe Engel. Augmented virtual reality: Combining crowd sensing and social data mining with large-scale simulation using mobile agents for future smart cities. In *Multidisciplinary Digital Publishing Institute Proceedings*, volume 4, page 5762, 2019.

[137] D Jayanthi and G Sumathi. Weather data analysis using spark—an in-memory computing framework. In *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pages 1–5. IEEE, 2017.

[138] Apache Beam. An advanced unified programming model, 2018.

[139] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in

a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[140] Shen Li, Paul Gerver, John MacMillan, Daniel Debrunner, William Marshall, and Kun-Lung Wu. Challenges and experiences in building an efficient apache beam runner for ibm streams. *Proceedings of the VLDB Endowment*, 11(12):1742–1754, 2018.

[141] Sukhpal Singh and Inderveer Chana. Cloud resource provisioning: survey, status and future research directions. *Knowledge and Information Systems*, 49(3):1005–1069, 2016.

[142] Sunilkumar S Manvi and Gopal Krishna Shyam. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *Journal of Network and Computer Applications*, 41:424–440, 2014.

[143] Abdul Hameed, Alireza Khoshkbarforoushha, Rajiv Ranjan, Prem Prakash Jayaraman, Joanna Kolodziej, Pavan Balaji, Sherali Zeadally, Qutaibah Marwan Malluhi, Nikos Tziritas, Abhinav Vishnu, et al. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 98(7):751–774, 2016.

[144] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. Evaluation of cloud autoscaling strategies under different incoming workload patterns. *Concurrency and Computation: Practice and Experience*, page e5667.

[145] Harold W Cain, Ravi Rajwar, Morris Marden, and Mikko H Lipasti. An architectural evaluation of java tpc-w. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 229–240. IEEE, 2001.

[146] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156, 2014.

[147] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[148] Scott Schneider, Martin Hirzel, Bugra Gedik, and Kun-Lung Wu. Auto-parallelizing stateful distributed streaming applications. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 53–64. ACM, 2012.

[149] Leonard Kleinrock. *Queueing systems, volume 2: Computer applications*, volume 66. wiley New York, 1976.

[150] David G Kendall. Some problems in the theory of queues. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 151–185, 1951.

[151] Rahav Dor, Joseph M Lancaster, Mark A Franklin, Jeremy Buhler, and Roger D Chamberlain. Using queuing theory to model streaming applications. In *Symp. on Application Accelerators in High Perf. Computing*, 2010.

[152] Per Hokstad. On the steady-state solution of the m/g/2 queue. *Advances in Applied Probability*, 11(1):240–255, 1979.

[153] Maria A Rodriguez and Rajkumar Buyya. Budget-driven resource provisioning and scheduling of scientific workflow in iaas clouds with fine-grained billing periods. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 9(4), 2015.

[154] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[155] Konstantinos E Parsopoulos, Michael N Vrahatis, et al. Particle swarm optimization method for constrained optimization problems. *Intelligent Technologies–Theory and Application: New Trends in Intelligent Technologies*, 76(1):214–220, 2002.

[156] AWS. Amazon ec2 instance types, 2020. URL `https://aws.amazon.com/ec2/instance-types/`.

[157] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[158] Anthony D JoSEP, RAnDy KAtz, AnDy KonWinSKi, LEE Gunho, DAViD PAttERSon, and ARiEL RABKin. A view of cloud computing. *Communications of the ACM*, 53(4), 2010.

[159] Mohammad Masdari, Sima ValiKardan, Zahra Shahi, and Sonay Imani Azar. Towards workflow scheduling in cloud computing: a comprehensive analysis. *Journal of Network and Computer Applications*, 66:64–82, 2016.

[160] Shubham Jain and Jasraj Meena. Workflow scheduling algorithms in cloud computing: An analysis, analogy, and provocations. In *Innovations in Computer Science and Engineering*, pages 499–508. Springer, 2019.

[161] Hyeong S Kim, In Soon Cho, and Heon Y Yeom. A task pipelining framework for e-science workflow management systems. In *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 657–662. IEEE, 2008.

[162] Luiz F Bittencourt, Alfredo Goldman, Edmundo RM Madeira, Nelson LS da Fonseca, and Rizos Sakellariou. Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*, 30:31–54, 2018.

[163] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[164] Raed Alsurdeh, Rodrigo N Calheiros, Kenan M Matawie, and Bahman Javadi. Cloud resource provisioning for combined stream and batch workflows. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2018.

[165] Cisco Global Cloud Index. Forecast and methodology, 2016–2021 white paper. *Updated: February*, 1, 2018.

[166] Xiao Ma, Ao Zhou, Shan Zhang, and Shangguang Wang. Cooperative service caching and workload scheduling in mobile edge computing. *arXiv preprint arXiv:2002.01358*, 2020.

[167] Cihat Baktir, Cagatay Sonmez, Cem Ersoy, Atay Ozgovde, and Blesson Varghese. Addressing the challenges in federating edge resources. *Fog and Edge Computing: Principles and Paradigms*, pages 25–49, 2019.

[168] Stefan Nastic, Thomas Rausch, Ognjen Scekic, Schahram Dustdar, Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Sasko Ristov, and Radu Prodan. A serverless real-time data analytics platform for edge computing. *IEEE Internet Computing*, 21(4):64–71, 2017.

[169] Sharad Agarwal, Matthai Philipose, and Paramvir Bahl. Vision: The case for cellular small cells for cloudlets. In *Proceedings of the fifth international workshop on Mobile cloud computing & services*, pages 1–5. ACM, 2014.

[170] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th annual workshop on network and systems support for games*, page 2. IEEE Press, 2012.

[171] Ahmed Sammoud, Ashok Kumar, Magdy Bayoumi, and Tarek Elarabi. Real-time streaming challenges in internet of video things (iovt). In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[172] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[173] Markus Löffler and Andreas Tschiesner. The internet of things and the future of manufacturing. *McKinsey & Company*, page 4, 2013.

[174] IDC. The growth in connected iot devices is expected to generate 79.4 zb of data in 2025, according to a new idc forecast. 2019.

[175] Hongbing Zhou, Weiyong Yu, Peng Yi, and Yiguang Hong. Quantized gradient-descent algorithm for distributed resource allocation. *Unmanned Systems*, 7(02):119–136, 2019.

[176] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[177] Nathan Vance, Daniel Yue Zhang, Yang Zhang, and Dong Wang. Privacy-aware edge computing in social sensing applications using ring signatures. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 755–762. IEEE, 2018.

[178] Abhishek Chandra, Jon Weissman, and Benjamin Heintz. Decentralized edge clouds. *IEEE Internet Computing*, 17(5):70–73, 2013.

[179] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1067–1080, 2007.

[180] Arnab Barua, Chunxi Dong, Fadi Al-Turjman, and Xiaoong Yang. Edge computing-based localization technique to detecting behavior of dementia. *IEEE Access*, 8:82108–82119, 2020.

[181] Laiping Zhao, Yizhi Ren, and Kouichi Sakurai. Reliable workflow scheduling with less resource redundancy. *Parallel Computing*, 39(10):567–585, 2013.

[182] Xumin Huang, Rong Yu, Jiawen Kang, and Yan Zhang. Distributed reputation management for secure and efficient vehicular edge computing and networks. *IEEE Access*, 5:25408–25420, 2017.

[183] Xiangwang Hou, Zhiyuan Ren, Jingjing Wang, Wenchi Cheng, Yong Ren, Kwang-Cheng Chen, and Hailin Zhang. Reliable computation offloading for edge computing-enabled software-defined iov. *IEEE Internet of Things Journal*, 2020.