SCHEDULING EDGE AND IN-TRANSIT COMPUTING RESOURCES FOR STREAM PROCESSING APPLICATIONS

by

ALI REZA ZAMANI ZADEH NAJARI

A dissertation submitted to the School of Graduate Studies Rutgers, The State University of New Jersey In partial fulfillment of the requirements For the degree of Doctor of Philosophy Graduate Program in Computer Science

Written under the direction of Manish Parashar and approved by

________________________

________________________

________________________

________________________

New Brunswick, New Jersey OCTORBER, 2019
The exponential growth of digital data sources has the potential to transform all aspects of society and our lives. However, to achieve this impact, the data has to be processed in a timely manner to extract insights that can drive decision-making. With the increasing availability of Internet of Things (IoT) devices, and potential applications that make use of data from such devices, there is a need to better identify appropriate data processing techniques that can be applied to this data. The computational complexity of these applications and the complexity of the requirements on the data processing techniques often derive from the capabilities of current IoT devices and the need to integrate data streams across multiple IoT devices. This results in larger data sizes and loads on the computing infrastructure.

While cloud computing is able to provide flexible computing and storage services,
important factors such as bandwidth provisioning between different components, leveraging network computing resources along the data path, and utilizing heterogeneous resources based on their geographic location to deploy the workflows are not supported in current cloud computing approaches and models. In fact, due to the data movement costs, traditional approaches that rely on moving data to remote data centers for processing are no longer feasible. Instead, new approaches that effectively leverage distributed computational infrastructure and services are necessary. Specifically, these approaches must seamlessly combine resources and services at the edge, in the core, and along the data path as needed.

To address these challenges, this dissertation explores various approaches to eliminate or alleviate the impact of limited resources to process large amounts of data. First, programming network resources using Software Defined Network (SDN) capabilities is added to the cloud federation to gain control over networking infrastructure. Second, having control over network computing resources (in-transit resources) enables us to provision latent resources at network nodes and process data using such resources by considering location and network properties of the resource and the flow of data. Finally, a novel data delivery approach is developed that uses heterogeneous resources located at the edge of the network and along the data path to process big data streaming applications and deliver the processed data to users while considering users constraints.

The main contribution of this dissertation is the integration of SDN and cloud federation that enables the provisioning of the in-transit network resources and provides more information about networking resources for deployment. Another contribution of this dissertation is the design and development of a large-scale publish/subscribe messaging system for data movement using CDN nodes to seamlessly route the data between components and automatically deploy stream-oriented workflows on geo-distributed heterogeneous nodes.

The validation of these works is done through a series of experiments based on real scenarios and applications. Three main applications have been considered as use cases of these works: (1) Data gathered from instrumented build environment applications, (2)
Video analytics applications for data collected by video surveillance cameras (3) Real-time data from Ocean Observatory Initiatives (OOI). Heterogeneous geographically-distributed resources with different capabilities, availability, and network condition are utilized to prove our hypothesis and show the effectiveness of our approach. The results demonstrate the potential impact of SDN, edge/in-transit processing, and approximate computing on various cloud federation aspects such as completion ratio of the jobs, quality/accuracy of the processed data, and utilization of resources.
Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Dr. Manish Parashar, for his continuous support of my Ph.D studies, for his patience, motivation, and immense knowledge. His guidance and supervision helped me during my research and writing of this dissertation. I could not have asked for a better advisor and mentor for my Ph.D studies.

Besides my advisor, I would also like to thank Dr. Ulrich Kremer, Dr. Srinivas Narayana, and Dr. Adrien Lebre for serving on my Ph.D. committee and for their advice and feedback along the way. Their comments, ideas, and insights helped me in strengthening my dissertation.

I would also like to thank my collaborators whom I have had the pleasure to work with during my graduate studies (in alphabetical order): Dr. Moustafa Abdelbaky, Dr. Ashiq Anjum, Dr. Daniel Balouek-Thomert, Dr. Javier Diaz-Montes, Eduard Gibert Renart, Dr. Ioan Petri, Dr. Omer Rana, Dr. Ivan Rodero, J. J. Villalobos and Mengsong Zou. I have learned a great deal from all of you.

Finally, I would like to express my deepest gratitude and appreciation to my family for all that they have done for me. Particularly, I would like to thank my parents, Hamidreza and Faegheh, and my sister Asal. I am where I am today because of you. Last but not least, I would like to thank my lovely wife Elham for all her help, for always believing in me, and being by my side throughout this journey.
Dedication

To my parents, for their endless love, support, and encouragement.
# Table of Contents

Abstract ........................................................................................................... ii  
Acknowledgements ................................................................................................... v  
Dedication .............................................................................................................. vi  

List of Tables ......................................................................................................... xi  
List of Figures ......................................................................................................... xiii  

1. Introduction ...................................................................................................... 1  
   1.1. Motivation ................................................................................................... 3  
      1.1.1. Scientific applications and workflows ................................................. 3  
      1.1.2. Internet of Things ................................................................................ 4  
      1.1.3. Large Scale Observatories ................................................................. 5  
   1.2. Crosscutting requirements from emerging applications ...................... 7  
      1.2.1. Distributed data processing in multi-cloud paradigm ....................... 8  
      1.2.2. Exploiting landscape of the resources .............................................. 9  
   1.3. Challenges .................................................................................................. 11  
      1.3.1. Exploiting network resources for wide-area data processing ........ 11  
      1.3.2. Coordination and orchestration of the resources for streaming ap-  
          plications ................................................................................................. 12  
      1.3.3. Data processing under application constraints .................................. 13  
   1.4. Dissertation Focus ..................................................................................... 14  
      1.4.1. Dissertation Components ................................................................... 14  
      1.4.2. State of the art .................................................................................... 15  
   1.5. Contributions ............................................................................................. 16
## Related Work

2.1. Stream Processing and Streaming Engines ............................................. 18  
   2.1.1. Edge Computing ................................................................. 20  
   2.1.2. In-Transit Processing ......................................................... 24  
2.2. Wide-Area Analytics ................................................................. 25  
   2.2.1. Multi-Cloud Data Processing ................................................. 27  
   2.2.2. Cloud Federation .............................................................. 28  
2.3. Network Abstraction Techniques ................................................. 28  
   2.3.1. Software Defined Network .................................................. 28  
      2.3.1.1. SDN Enabled Cloud Computing ..................................... 31  
   2.3.2. Network Function Virtualization (NFV) ................................. 32  
2.4. Quality of Service for Workflow Applications ................................. 33

## A Model to Leverage the Combined Use of Computational Capabilities Available at the Network Edge and between Computing Resources  

3.1. Introduction ................................................................. 35  
3.2. Video Analytics Use Case ..................................................... 37  
   3.2.1. Scenario 1: Processing at Capture Node (Edge) ....................... 39  
   3.2.2. Scenario 2: Processing at In-transit Nodes ......................... 39  
3.3. Resource Federation Model for Video Analytics ............................. 40  
3.4. Problem Formulation .......................................................... 42  
3.5. Scheduling Optimization Strategy ............................................. 45  
3.6. Configuration of Testbed ....................................................... 47  
3.7. Evaluation ................................................................. 50  
   3.7.1. Experiment 1 – Deadline based on completion time ............... 53  
   3.7.2. Experiment 2 – Deadline based on video size ...................... 57  
   3.7.3. Experiment 3. Validation of the Model ............................... 59  
   3.7.4. Experiment 4. Analytical Evaluation ................................. 61
5.5. Evaluation1 ................................................................. 102
  5.5.1. Workflow ............................................................. 102
  5.5.2. Experimental Setup and Scenario .............................. 103
  5.5.3. Results ............................................................... 104
5.6. Evaluation2 ............................................................ 111
5.7. Discussion ............................................................. 112
5.8. Relevant Publications ................................................ 113

6. Conclusion ..................................................................... 115
  6.1. Summary ................................................................. 115
    6.1.1. Leveraging SDN to exploit and provision in-transit resources . 116
    6.1.2. Using edge and in-transit resources to increase quality of solution 116
    6.1.3. Scheduling of the stream-oriented workflows on edge and in-transit
           nodes ................................................................. 117
  6.2. Prospectives ............................................................ 118
  6.3. Future Work ............................................................ 119
    6.3.1. Scalability ........................................................... 120
  6.4. Relevant Publications ................................................ 121
### List of Tables

3.1. Computational Resource Properties. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. .............................................. 49

3.2. Video Stream Analysis Time and Characteristics obtained from [2]. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. ....................... 51

3.3. Number of accepted jobs – Modifying Number of Workers. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 2x C means we doubled the number of workers in the cloud site; 2x I means we doubled the number of workers in the in-transit sites; and 2x C&I means we doubled the number of workers in both, the cloud and the in-transit sites. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10. .............................................. 64

3.4. Number of accepted jobs – Modifying Performance of Workers. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 2x C means we doubled the performance of workers in the cloud site; 2x I means we doubled the performance of workers in the in-transit sites; and 2x C&I means we doubled the performance of workers in both, the cloud and the in-transit sites. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10. .................. 66

4.1. Infrastructure Scenarios. ©2017 IEEE (reprinted with permission) Zamani et al. [3]. .............................................. 80

4.2. Resource Properties. ©2017 IEEE (reprinted with permission) Zamani et al. [3]. .............................................. 80
4.3. Job Information. ©2017 IEEE (reprinted with permission) Zamani et al. [3] .......................................................... 83


List of Figures

1.1. Traditional Data Processing System. Data is generated at the digital sources, processed at central cluster and delivered to the users. 7

1.2. Three disjoint approaches. Edge at the proximity of data source, in-transit is provisioned and leveraged by SDN and NFV techniques, multi-cloud infrastructures are the main source of computation. 11

1.3. Main components of this dissertation. 15

3.1. High Level Stream Processing Workflow. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 38

3.2. Video Analytics Scenario – adapted from [2]. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 38

3.3. Federated architecture that exposes edge and in-transit resources. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 40

3.4. Customized High Level Stream Processing Workflow. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 46

3.5. Infrastructure. Solid lines indicate network links of 20MB of bandwidth. We assume each SDN router is co-located with computational resources. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 48

3.6. Value Distribution for Each Job Type. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 51

3.7. Summary of experimental results. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10. 53
3.8. Summary of experimental results. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. Deadlines are: QCIF = 48s, CIF = 48s, 4CIF = 120s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10.

3.9. Summary of model validation results for 12-42-150 deadlines. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10. Each column represents a set of experiments, where Real means experimentally obtained and Model means analytically obtained.

3.10. Summary of model validation results for 48-48-120 deadlines. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10. Each column represents a set of experiments, where Real means experimentally obtained and Model means analytically obtained.

3.11. Summary of experimental results – Modifying Bandwidth. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10.

4.1. Energy optimization scenario. ©2016 IEEE. Reprinted with permission, Zou et al. [5].

4.2. Infrastructure Setup. Solid and dashed lines indicate high and low bandwidth links, respectively. ©2017 IEEE (reprinted with permission) Zamani et al. [3].

4.4. Average Accuracy. ©2017 IEEE (reprinted with permission) Zamani et al. [3]. .............................................................. 85

4.5. Idle time Overheads per job. ©2017 IEEE (reprinted with permission) Zamani et al. [3]. High standard deviation existed because various approximation techniques have been selected and they have wide range of execution times. .............................................................. 86

4.6. Average Cost. ©2017 IEEE (reprinted with permission) Zamani et al. [3]. 86

5.1. Workflow pipeline. ©2018 IEEE (reprinted with permission) Zamani et al. [4]. .............................................................. 92

5.2. The time to reach an optimal solution for different number of workflow stages and resources. ............................................. 95

5.3. Services Involved in Data Delivery and Processing. ......................... 96

5.4. Subscription-Based Data Movement Using Distributed Kafka Clusters. ©2018 IEEE (reprinted with permission) Zamani et al. [4]. ......................... 98

5.5. Comparing Regular Stream Processing with Combining Requests Approach. Worker Nodes Haven’t Been shown Here for Simplicity. ........... 102

5.6. Sliding Window and Image Pyramids. ©2018 IEEE (reprinted with permission) Zamani et al. [4]. .............................................................. 103

5.7. Infrastructure Consisting of Producer, Edge, In-transit and Core Resources. ©2018 IEEE (reprinted with permission) Zamani et al. [4]. .... 103

5.8. Utilization And Number of Streams without Approximation, Edge and In-Transit Resources. ©2018 IEEE (reprinted with permission) Zamani et al. [4]. .............................................................. 105

5.9. Number of Streams Being Delivered to Users Over Time for Different Minimum Acceptable Data Resolutions. ©2018 IEEE (reprinted with permission) Zamani et al. [4]. .............................................................. 106

5.10. Comparing Acceptance Ratio in Baseline and Edge/In-Transit Enabled Scenarios for Different Qualities. ©2018 IEEE (reprinted with permission) Zamani et al. [4]. .............................................................. 106
5.11. Utilization of Resources for Different Resolutions ©2018 IEEE (reprinted with permission) Zamani et al. [4]......................... 107
5.12. Maximum utilization of the core resources for various scenarios and resolutions. .................................................. 108
5.13. Effect of Sudden Change in Bandwidth on Number of Streams and Average Resolution of the streams. ©2018 IEEE (reprinted with permission) Zamani et al. [4]........................................... 109
5.14. Number of Users and Average Resolution at Runtime for Various Deadlines and Resolution of 80%. ©2018 IEEE (reprinted with permission) Zamani et al. [4]................................. 110
5.15. Acceptance Ratio for baseline and data merging scenarios. Acceptance ratio decreases when number of data products increases. ............... 112
5.16. Number of users being served throughout the experiment ............ 113
Chapter 1

Introduction

The exponential growth of digital data sources has the potential to transform all aspects of our society. This growth can have an impact on the quality of human life by fundamentally transforming our ability to understand and manage our environment. For instance, in case of applications related to the management of crisis and national disasters such as hurricanes or earthquakes, real time processing of the weather and geo-science information can have a critical impact on human lives [6].

To achieve this impact, data has to be processed in a timely manner in order to extract insights that can drive decision making. Due to the recent exponential growth in the amount of data, traditional approaches that rely on moving data to remote centralized data centers for processing are no longer feasible [7]. Hence, there has been recent interest in moving away from such centralized, large-scale data centers to a more distributed multi-cloud setting (as demonstrated by significant interest in cloud federation and interoperability efforts) [8, 9]. Such a multi-cloud environment is often formed by a network of more than one cloud platform, in which each one delivers a specific application service. A multi-cloud can be comprised of a public, private, and hybrid clouds to achieve specific goals [10].

Distributed resources within a multi-cloud setting are connected to each other using network links owned by network providers. Network providers are increasingly becoming potential sources of general purpose computation by minimizing the amount of network-specialized hardware hosted in their data centers and moving towards the use of commodity hardware [11]. This strategy follows state of the art networking approaches such as Software Defined Networks (SDN) and Network Functions Virtualization (NFV). Understanding how the availability of commodity servers within such
“network data centers” can contribute towards data processing would enable an effective way to extend the boundaries of a cloud system from a high-end, often localized data center, to multiple distributed data centers that can process data while it is in transit from source to destination [5, 12].

Aside from in-transit resources, with the maturity of the Internet of Things (IoT) paradigm, data sensing can now be combined with data processing and analytics on the same device. This is referred to as edge computing [13]. As IoT devices increase in function and capability, existing systems designed to process data gathered from IoT devices and remote sensors can leverage resources at the proximity of the data source and en-route from data source toward the data centers. Such a perspective assumes that IoT devices and in-transit network nodes, over which such data is channeled, can be used to support data processing along with the data centers to which this data is transmitted [14].

In this regard, new approaches that effectively leverage distributed computational infrastructure and services are necessary. Since such infrastructures and services are geographically distributed, data is exposed to network resources while it moves between computational sites. Hence, it is crucial that these approaches seamlessly combine resources and services at the edge, in the core, and along the data path as needed.

In this dissertation, we propose approaches for enabling and realizing a fluid ecosystem where distributed resources and services are provisioned at the edge and in-transit nodes to support emerging data intensive applications and stream oriented workflows. We propose a conceptual architecture that leverages ideas from software-defined networks, edge and in-transit processing, and approximate computing in order to effectively process and deliver processed data to the users and applications. We deploy the workflow stages on resources en-route from source to destination. In addition, we demonstrate how this proposed architecture enables run-time management of data quality and provides end-to-end QoS for users at various geographic locations.

**Thesis Statement:** As edge, in-transit and clouds are typically decoupled from each other, designing a framework that uses them together in an integrated manner as part of an application workflow is challenging. In this dissertation, we are going to
address this challenge by developing a mathematical scheduling model to support edge and in-transit data analysis in cloud federation. In addition, we design and develop a subscription-based data streaming framework to deploy and execute stream oriented workflows, and a runtime management layer that leverages edge and in-transit resources to adjust the resolution and provide end-to-end QoS using on-demand control loops. The validation of this work is done through a series of experiments on three main applications: (1) EnergyPlus, (2) Image processing for surveillance cameras, and (3) Ocean Observatory Initiatives (OOI). The results demonstrate the effectiveness of our approach on increasing job acceptance ratio and utilization of the resources by almost 25% and 16%, respectively.

In the remainder of this chapter, we give an overview of the emerging applications that we consider in this dissertation. We explain the necessity of novel approaches to incorporate and provision various services and geo-distributed heterogeneous resources to process big data applications. Then, we present the challenges that are addressed in this dissertation followed by the summary of the state-of-the-art approaches and the key contributions of this dissertation.

1.1 Motivation

In this section, we use various applications from diverse areas that motivate the need to develop new approaches that effectively leverage heterogeneous resources in-transit from source to destination and improve the performance of workflows and applications running on geographically distributed resources.

1.1.1 Scientific applications and workflows

Computing has been an enormous accelerator to science and has led to an information explosion in different fields [15]. The increase in the number of devices and their accuracy has caused the amount of data generated as a result of scientific experiments to grow at a very fast rate. Although the size of the data is very large, the ability to draw conclusions and take actions based on the data relies on the data to be processed,
analyzed and visualized in a very short amount of time.

In this context, workflows have emerged as a paradigm for representing and managing complex distributed scientific computations and accelerating the pace of scientific progress. They provide an orchestration of the data flow across the individual data transformations, and a mechanism to execute them in a distributed environments [16]. They are used in many areas of science such as biology, astronomy, chemistry and geoscience. Workflows can be composed of multi-step computational stages such as calibration, data transformation, visualization, anomaly and object detection, and result collection, that should be applied to the huge volume of data generated from scientific instruments and devices.

For instance, the Korean Superconducting Tokamak Advance Research (KSTAR) project [17] generates 3 TB of data in less than 2 minutes and timely analysis of the generated data to detect anomalies that could damage the instrument is necessary. The execution of this workflow requires a massive number of computing resources that are not collocated with the research center. Thus, cloud computing can play an important role by enabling scientists to provision their required resources on-demand using a pay-per-use basis and execute their workflows using public/private cloud facilities. However, this model requires massive data transfers between experimental sites and cloud resources, which is expensive in terms of time, energy and cost [18]. Hence, the computational models that seamlessly integrates in-situ, edge and in-transit resources closer to the data sources can potentially accelerate scientific discoveries and reduce the overall execution and data movement costs.

1.1.2 Internet of Things

With the growing ubiquity of intelligent devices, more data than ever is being trafficked across global networks. This has given rise to the “Internet of Things (IoT)” [19]. At present, over 20 billion IoT devices are in use around the world, and this number is expected to increase in the coming years as Cisco predicted that will be more than 50 billion Internet-connected objects by 2020 and Huawei estimated that there will be 100 billion devices connected to the Internet by 2025 [20, 21].
The IoT supports a vast ecosystem of applications and programs such as smart cities, wearables, connected cars and industrial Internet, all of which produce and process enormous amounts of data. Data are generated as data streams, i.e. machine-generated data produced at high speed rates, meaning that they are constantly generated and need to be aggregated, processed and stored locally or in remote data stores and databases. Such data streams possess different characteristics such as rate, size and resolution, which highlight the need for design and development of adaptable methods for data processing [22]. Applications need to combine information from various devices and data streams and use historical data stored in databases to produce accurate results. Aside from data, IoT applications hold various computation complexities, from simple comparisons to advanced image processing and object detection algorithms.

The seamless interaction between devices (technological interoperability) and correct sharing of information between them (semantic interoperability) put an increased pressure on communicational resources [23] that requires solutions that leverage resources at the proximity of such devices to reduce this pressure. Also, the complexity of the IoT and the applications that utilize it to expand our lives have demanded new forms of cyberinfrastructure and data processing techniques. Through the construction of the novel computational infrastructures, researchers will be able to ensure fast and high-quality data processing and delivery as data demands increase.

1.1.3 Large Scale Observatories

Open, large-scale scientific facilities are an essential part of the science and engineering enterprise. Large scale observatories are nationally/internationally funded shared-use resources designed to provide the scientific community with open access to data and data products from geographically distributed sensors and instruments [24].

These facilities provide shared-use infrastructure, instrumentation, and data products that are openly accessible to a broad community of researchers and educators. They consist of thousands of sensors and instruments distributed across the globe to measure and capture data which will be used by scientists and engineers. The raw data captured from the devices are transferred and stored in gigantic storage units and
magnetic tapes. The data are (pre)processed and delivered to the users for further processing. Examples of such observatories include the National Ecological Observatory Network (NEON) ¹, the Large Synoptic Survey Telescope (LSST) ², and the Ocean Observatories Initiative (OOI) ³.

While these facilities provide reliable and pervasive access to raw data and data products, users typically have to download the data of interest and then process them, typically using local resources. Consequently, transforming these data and data products into insights requires local access to powerful computing, storage, and networking resources. These requirements can significantly limit the impact of the data, especially for researchers, educators, and students who do not have access to such capabilities. Due to the increase in size of the data and computational complexity of the applications, such computing models are no longer feasible and sustainable. Hence, developing a system that automatically integrates resources (computing, network and storage) from public/private clouds in order to process the workflows and applications is essential.

Aside from the mentioned requirements, these systems should hide the complexities of the workflow deployments by automatically provisioning the appropriate resources from the pool of available resources, leverage the resources based on users’ need, and assign the workflow stages to selected resources. Such systems will be able to remove scientists from actively being involved in processing and delivery cycles. The new systems should also be able to address the growing demand of stream oriented workflows [25] and provide Quality of Service (QoS) for the users located around the globe for extended period of time (as long as users want to receive updated data from instruments). In such systems which consist of various wide-area connected devices and resources, long term end-to-end QoS monitoring and providing users with fast and reliable results are essential.

¹National Ecological Observatory Network: http://www.neonscience.org
²Large Synoptic Survey Telescope: https://www.lsst.org
³Ocean Observatories Initiative: http://oceanobservatories.org
1.2 Crosscutting requirements from emerging applications

In most of traditional batch and stream processing environments, end-to-end data delivery is performed in three concrete steps: (i) Downloading the raw data from data sources to the centralized data centers, (ii) Using centralized well-provisioned resources to transform the data and (iii) Delivering the results to the users [26]. The schematic overview of these models is depicted in Figure 1.1.

![Figure 1.1: Traditional Data Processing System. Data is generated at the digital sources, processed at central cluster and delivered to the users.](image)

The limitations caused by a centralized model of data processing are as follows:

1. **Uploading**: Network resources are usually not as scalable as computing resources. Hence, injecting all of the data generated from numerous data sources to a central location causes network resource limitations. As a result, in this data processing model, providing QoS for the users and (near) real-time data delivery is almost impossible. When the inbound network of a data center is congested, the system would not be able to accommodate more requests and deliver the result in a timely manner. Consequently, even though there are computing resources available to process more requests, bandwidth limitation reduce the systems’ ability to address more requests.
2. Transformation/processing: As workflows are getting more complex and compute-intensive, more powerful resources are needed to process each request [27]. As the number of such requests increases, the resources within one data center are not sufficient to satisfy all of the requests from a large number of users [28, 29]. As a result, a single data center is not sufficient to process the requests. Hence, part of the requests should be outsourced/redirected to remote resources and data centers. Moreover, in current frameworks, the users and scientist are actively part of the processing cycles. The user’s involvement in date processing can result in a non-optimized application execution.

3. Data delivery: In most of the current systems, users are required to manually download results of their requests/applications/workflows. This process can get cumbersome and error prone specially in streaming applications. Hence, a new system has to be designed that minimizes the manual steps in data processing and delivery needs to be designed.

In all of the emerging applications mentioned in Section 1.1, the increase in size of the data and number of devices necessitates the using of resources outside the scope of one cluster or data center, specifically, where data needs to be processed within a specific time interval. In addition, geographic distribution of data sources motivates scientists to explore new methods that can leverage geo-distributed resources to process such data [30, 31]. These limitations and requirements highlight the need to: (i) process data using resources existing in multiple locations, (ii) leverage public and private clouds on-demand (multi-cloud environment), (iii) use resources near the data source (edge resources) and in-transit to process data with less latency and shape the data as it moves between geographically distributed resources.

1.2.1 Distributed data processing in multi-cloud paradigm

In the last decade, clouds are becoming an increasingly popular system to execute large-scale data intensive scientific and business applications [32]. They offer on-demand access to computing utilities and a pay-as-you-go business model.
Beside utilizing single cloud and data center for computations, there has been recent interest in creating “multi-cloud” ecosystem. Users and application developers are moving toward the use of distributed data processing techniques to deploy and execute their applications and workflows [33]. Such infrastructures are generally distributed and comprised of the integration of capabilities from a variety of different Cloud providers [34]. There are several advantages in moving away from centralized environments toward multi-cloud fashion. The main benefit of such systems is the on-demand aggregation of resources at multiple locations that increases the ability of the system to process an increase amount of data using on-demand remote resources when local resources are limited. In this case, users and applications can rely on multiple vendors’ infrastructures. Hence, the system will be fault tolerant and has better security and robustness [10].

Distributed data processing using resources at multiple locations causes large amount of data transfers between computing infrastructures. These data transfers are costly and time consuming, especially when time and budget are bounded by users.

1.2.2 Exploiting landscape of the resources

Combining IoT and cloud computing capabilities enables the creation of smart environments that can respond to real-time events, by (a) combining services offered by multiple stakeholders (i.e. those that are at the network edge with services provided within a data center) and, (b) providing scalable resources to support a large number of users in a reliable and decentralized manner. They need to be able to operate in both wired and wireless network environments and deal with constraints such as access devices or data sources with limited power and unreliable connectivity. The cloud application platforms need to be enhanced to support (a) the rapid deployment of services by providing domain specific programming tools and environments and (b) the seamless execution of applications harnessing capabilities of multiple dynamic and heterogeneous resources to meet quality of service requirements of different users.

Data centers managed and operated by network providers form a significant part of the current Internet infrastructure, since there are large numbers of such data centers
that are almost ubiquitous across the world. These data centers may not be as powerful as computational data centers that are hosted by cloud providers or traditional high performance computing (HPC) providers. However, their ubiquity and the fact that we have to necessarily use them, when moving data over the Internet, makes them a useful source of pervasive computing within the network. Understanding how the availability of commodity servers within such “network data centers” can contribute towards data processing enables an effective way to extend the boundaries of a cloud system – from a high end, often centralized data center, to multiple distributed data centers that can process data while it is in transit from source to destination. Figure 1.2 presents the resources that can be leveraged to process data as it moves from the producer toward the consumer.

In this dissertation, IoT devices and devices located near data source are categorized as edge nodes or edge clouds. Despite the fact that there are millions of IoT devices around the world, these resources are not well-provisioned and they are designed to perform simple computations on small datasets. Furthermore, these resources are limited and expensive. Although these resources have limited capabilities, since these resources are at the proximity of data sources, they have very low latency and high bandwidth connections to the data sources. Therefore, they can be leveraged to process/pre-process data and filter the data to save network bandwidth resources. In contrast, the resources located at the core of the infrastructure are located within well-provision data centers. They are the main source of computation for most applications. Such resources are relatively inexpensive and highly available for various applications/users. Due to the fact that they are usually far away from data sources, they have high latency and low bandwidth connection to them. In-transit resources are located between edge and core resources. Their price and bandwidth are higher than core resources and lower than edge resources.
Figure 1.2: Three disjoint approaches. Edge at the proximity of data source, in-transit is provisioned and leveraged by SDN and NFV techniques, multi-cloud infrastructures are the main source of computation.

1.3 Challenges

1.3.1 Exploiting network resources for wide-area data processing

Network operators are increasingly becoming providers for general purpose computation infrastructure [35]. They are minimizing the amount of network-specialized hardware hosted in their data centers and moving towards the use of commodity hardware. This strategy is supported in recent efforts in Software Defined Networking (SDN) and Network Functions Virtualization (NFV). SDN, in particular, is an approach devised to simplify network management through abstraction of lower-level functionality. Specifically, SDN separates control plane (where to send data) from data plane (data forwarding functions). This enables running the software-based control plane on commodity servers and to leveraging the latest-generation of processors which are faster than embedded-class processors in most switches [36]. On the other hand, NFV goes a
step further and extends the as-a-service cloud model to offer networking functions on-demand using virtualization techniques. This approach promises a reduction in capital expenses and a rapid deployment and delivery of new functionality [37].

As shown in Figure 1.2, three areas of edge processing, in-transit processing and multi-cloud data processing can potentially be involved in data processing. However, designing a framework that leverages all of these approaches is challenging because these approaches are decoupled from one another. The aggregation of the edge and network resources, exposing those resources for the workflow execution, provisioning and leveraging them to process parts of the data, and applying workflow stages require coupling between network providers and cloud resource providers that does not exist in the literature [38]. These tasks require leveraging novel networking approaches, such as SDN and NFV, and utilizing a central orchestrator or controller that manages routing tables and other network functions. Such an orchestrator has knowledge of the overall network architecture that it for instance being used to optimize traffic routes based on utilization and available bandwidth. Hence this controller needs to be extended to enable the management of computational capabilities available at each network data center. It is worth mentioning that the scalability and elasticity of SDN controllers can be achieved by implementing them in a “logically centralized and physically distributed” manner [39].

1.3.2 Coordination and orchestration of the resources for streaming applications

Although there have been improvements in the development of tools to process large amounts of data, processing and managing such data using distributed environment in a timely manner is still an unsolved challenge, especially in streaming application where the data source(s) are located at various geographic locations and are continuously generating data [40].

The main challenge in this area is the coordination and orchestration of infrastructures for streaming applications and instant extraction of actionable insight from the data [41, 42]. Design and implementation of a framework that aggregates the resources
at the edge and en-route from source(s) and destination(s) to process streaming application is very challenging because: (a) users are located in different geographic locations (possibly far away from data sources) and there are many components, services and assets involved in the processing of wide-area streaming applications/workflows, (b) resources are distributed in various geographic locations and they have various characteristics such as cost, computing power, energy consumption and availability. Hence, the coordinator needs to gather all of the information such as cost, performance, availability and utilization, from the resource providers and coordinate numerous resources to deploy the applications on the heterogeneous geo-distributed resources along the data path. Since the data is constantly generated from data sources, the coordinator should have updated information about the resources, adjust the execution space at runtime (dynamic scheduling), and constantly optimize application metrics, e.g., delay, accuracy and quality of the result, based on the current status of the data processing cycles.

1.3.3 Data processing under application constraints

In many scenarios, users and applications are required to receive processed/transformed data with several particular constraints such as deadline, budget and quality. We refer to these constraints as Quality of Service (QoS). However, guaranteeing on-time data delivery within specific constraints imposed by the users in environments composed of heterogeneous resources such as network links, virtual machines and bare metal servers requires service/resource coordination. Also, due to the large number of resources and devices, composing appropriate services and resources from various possible solutions and satisfying users’ objectives is a challenge [43]. Moreover, in environments where data is continuously generated and processed via complex workflows, providing QoS for a long period of time and avoiding QoS degradation require comprehensive monitoring [44]. Hence, the system should be able to examine and monitor application metrics at runtime which include accuracy, timeliness and resolution, and leverage available resources to improve them to reach better QoS.
1.4 Dissertation Focus

The ideas and techniques that are presented in this dissertation are designed to increase our ability to handle large amounts of data. The focus of this dissertation is on opportunistically combining the techniques mentioned above. Specifically, we propose methods that provision and leverage the available resources located within the data path, from the source toward the destination, process the data partially/completely, filter it before it reaches the final destination, and use the edge and in-transit resources for quality of service monitoring. Programmatically controlling the state of the execution environment and the continuous orchestration/composition of the resources at various geographic location are the other targets of this dissertation. In the experimental evaluation of this dissertation, we show the effectiveness of our model in satisfying more requests when computational capabilities are limited. Also, we demonstrate the benefits of edge and in-transit resources in increasing the utilization of data centers by sending more valuable information toward data centers. We also confirm the advantages of approximate computing on edge nodes by showing that the workflows can be executed with higher accuracy when edge clouds are enabled.

1.4.1 Dissertation Components

As shown in Figure 1.3, there are several steps that we need to take in order to design a framework that schedules the workflows on heterogeneous edge and in-transit resources. In Chapter 3, we explore the SDN capabilities which enables us to extract resources within in-transit network nodes. We also use these resource to partially process the data as it moves between the nodes in the cloud federation system. In Chapter 4, our framework considers users' provided approximation techniques at the edge and in-transit nodes to satisfy end-to-end quality of service. Finally, in Chapter 5, a mathematical model is designed to schedule workflows over edge and in-transit nodes. In the same Chapter, we show how to use resources along the data path to apply workflow stages. A subscription based data streaming framework that moves data effectively between the nodes and applying workflow stages on streaming data is also presented in Chapter 5.
Moreover, in Chapter 5, we dynamically establish on-demand feedback control loops using in-transit nodes to control data quality and satisfy end-to-end QoS for users.

Figure 1.3: Main components of this dissertation.

1.4.2 State of the art

Recently, the topics of cloud computing, IoT and distributed data processing have been the target of many research papers. Specifically, with the evolution of data intensive applications, many researchers have focused on effective models and approaches to process data using distributed resources and services.

The work related to this dissertation can be categorized into six main topics. A detailed discussion on the literature for each is provided in Chapter 2. Related main components and a brief explanation of each topic are listed below:

- **Edge computing**: This is the practice of using resources at the proximity of data source to partially/completely process data. By leveraging resources near the data sources, systems can use the prevalence of ubiquitously connected smart devices near the data sources and shape the data with very low latency [45].

- **In-transit processing**: This is a technique that uses the resources between two nodes (source and destination) to process the data and add value to the data while it moves [46]. Each node with the processing capabilities at the network data centers can get the data, process it and pass it to the next node.

- **Approximate computing**: This topic consists of a group of computation techniques that produce inaccurate and yet acceptable results while preserving computation and communication resources. These techniques can be categorized into
several well-known subjects such as precision scaling, task dropping, lossy compression, and using artificial neural networks [47].

- **Wide area analytics:** This area focuses on the processing of data using resources connected via wide-area links and the Internet. The basic idea for wide area analytics is to move away from traditional centralized data processing environments. This is mainly beneficial for the cases where data sets are very big and generated from distributed data sources [48, 49].

- **Scheduling:** Scheduling is the process of arranging and controlling the workloads and assigning them to available resources. Scheduling is mainly divided into two categories, i.e. static and dynamic scheduling, that has been explored in many research papers [50, 51].

- **Monitoring:** The efficient management and operation of complex infrastructures relying on proper monitoring [52]. This is a series of actions that control the execution and progress of a workload/workflow.

The existing frameworks that are widely used today for distributed data processing focus mainly on one or two of the mentioned topics and are mainly proposed to increase the performance of big data applications. On the other hand, in this dissertation, we design and develop a framework that advances these categories in several ways, individually, and incorporates all of them in one unified framework to satisfy end-to-end QoS requested by a wide range of users at multiple geographic locations.

### 1.5 Contributions

In this dissertation, we propose solutions and techniques that effectively add value to the data while it moves toward the destination and reduces data size near the data sources to overcome bandwidth limitations. We develop an analytical model to improve the use of computational resources at the network edge and within network data centers to support data transformation and analysis from source to destination. This dissertation makes the following contributions:
1. Integration of Software Defined Network (SDN) technology within the cloud federation to get information from the network resources.

2. Using SDN to provision network resources to process the data as it moves between main resources and reduce the queuing overheads.

3. Developing a mathematical scheduling model to support in-network data analysis in federated ecosystems.

4. Design and development of a subscription-based data streaming framework to deploy and execute stream oriented workflows.

5. A runtime management layer that leverages edge and in-transit resources to adjust the resolution and provide end-to-end QoS using on-demand control loops.

1.6 Outline

The rest of this dissertation is organized as follows. In Chapter 2, we provide a brief background about the specific techniques and technologies that we leveraged in this dissertation. We also provide a literature review of related works. In Chapter 3, we highlight the needs to leverage SDN technology in cloud based systems in order to control network infrastructure and provision computing resources and process the data while it moves. In Chapter 4, we focus on the QoS prospective of our approach. We present our approach to leverage edge resource to increase the quality of result (QoR). In Chapter 5, we provide a mathematical scheduling optimization technique to schedule workflow stages on edge and in-transit nodes. Also, we demonstrate the use of intermediate nodes to control and adjust data resolution at runtime. Finally, we conclude the dissertation in Chapter 6 by outlining future research in this area.
Chapter 2
Related Work

The work presented in this dissertation is based on several well-known and important concepts such as stream processing, edge and in-transit computing, and wide area analytics. Also, emerging networking techniques have been widely considered and deployed in our framework. In this section, we will provide a comprehensive literature review of the related topics.

2.1 Stream Processing and Streaming Engines

Stream processing is a programming paradigm aiming to process large amounts of data continuously generated by data sources such as sensors, mobile and IoT devices, and scientific instruments. These data sources (stream producers) are connected to the Internet and public or private networks [44, 53, 54]. Such data are usually pushed to data centers for data analytics and processing. Hence, data can be processed using enormous amounts of computing resources that exist within such well-provisioned data centers. In order to process such data streams, several systems and frameworks have been proposed by researchers, which we will cover below.

For many years, batch processing was a very hot topic in the data processing area where scientists have mostly focused on processing data using techniques such as MapReduce [55] that is a programming model for batch processing on large data sets. Batch systems such as Hadoop [56] and Apache Spark [57] have been designed to leverage MapReduce to process large volumes of data stored in Hard Disk or RAM memory.

There have been efforts to change batch processing ecosystems and make them suitable for streaming data. For instance in [58], authors proposed online MapReduce that
enables programs to be modified and executed as a streaming application. They also proposed pipelined MapReduce to increase level of parallelism in processing the data generated as a stream. As a result, the completion time of requests decreased compared to regular MapReduce execution. Aside from programming models, engineers have changed the existing frameworks to perform stream analytics by introducing the concept of mini batching, which divides the computations to a series of small batch components [59]. Although these systems are designed based on modifications on batch processing frameworks, since the main design of such systems is based on batch processing, they do not work effectively in a distributed stream processing space. These systems have high overheads and produce results with very high latency that is not desirable for streaming environments.

In terms of systems that are designed for stream processing ecosystems, the very first distributed stream processing engine was Aurora [60] that was designed for single site stream processing. Aurora represents Directed Acyclic Graph (DAG) based workflow systems. The newer version of Aurora was introduced as Borealis [61]. Unlike Aurora, Borealis is a fully distributed processing engine that runs on servers within multiple sites. A key component that was added in Borealis is a module that runs on each server which is responsible for deciding if a query should run locally or remotely. Borealis dynamically revises query results, modifies query specifications, and provides high availability and fault tolerance features.

Apache Storm [62] is another distributed stream processing engine that performs event-by-event computations and processes data streams in real-time. Within Storm, there exists a node/server called Nimbus which receives the codes from users and distributes them across available workers. In case of failures, which sometimes happen within the worker nodes, Nimbus can assign new workers to the corresponding stream. Apache S4 [63] is another fully distributed streaming engine that was developed by Yahoo. The processing units within S4 are called Processing Elements (PEs) and users are responsible to provide PEs with computational information and assign them to the nodes.

Apache Samza [64] is another type of streaming engine. In this engine, Apache
Yarn [65] has been utilized for resource allocation. Also Samza uses Apache Kafka [66] for the distributed messaging functionality, which is a distributed messaging system and a streaming platform. The messages are produced and consumed through Kafka, which provides persistence model for messages. Tasks within each stream produce data for the Kafka cluster or consume the data from it. The framework proposed in this dissertation leverages Kafka for subscription-based data streaming functionalities.

Aside from the mentioned well-known systems, researchers have also proposed new techniques and frameworks to process data streams effectively. For example, Santos et al. [67] have demonstrated the benefits of distributed stream processing engines. Their streaming engine called DiAI is based on two main components: 1) JStream, which is event processing unit and, 2) DiAIM, which is responsible for the management of distributed services. Moreover, in the same work, it has been shown that bandwidth limitations can be resolved by edge pre-processing and pre-filtering, which inspired us to investigate edge processing techniques. Their paper has shown the effectiveness of DiAI using a manufacturing use-case. However, the edge pre-processing and filtering techniques that are considered in their use case are superficial.

Although the proposed systems are designed to move away from batch processing and toward streaming processing environments, they mostly fail to effectively support geo-distributed streaming applications because their deployment layer have not considered network bandwidth and resource locations, data aggregation, data filtering, and QoS service monitoring. Hence, in this dissertation, we exploit edge and in-transit resources to filter data near the data sources and process the streams as they move between the components.

2.1.1 Edge Computing

Regarding edge computing, the vast majority of prior works have focused on the fact that edge resources have lower latency compared to core/cloud/data center resources and can start processing data earlier. These papers also showed that data filtration or aggregation at the edge of the network can reduce bandwidth utilization.

For instance, Heintz et al. [68] utilized the group aggregation technique to combine
and summarize large quantities of data, from one or more streams, at the edge. Their solution controls the transfer frequency of the aggregated data/result to a central node. They basically considered the trade-off between staleness and WAN traffic, which has been considered in our framework as well. They argued that in geo-distributed data processing ecosystems, sending all of the data to a central data center is not feasible due to bandwidth limitations. Moreover, processing all of the data at the edge is not feasible due to limited power and computing capabilities at the edge. The effect of windowed group aggregation, which decides on the amount of computation performed at the edge versus the central node, has been studied by the same group [69]. In pure batching (aggregating all the data at the edge), the network traffic is greatly reduced compared to pure streaming scenario (sending all the data to the central node). However, pure batching introduces big staleness due to scarce computing nodes at the edge. The offline and online algorithms to minimize both staleness and network traffic have been proposed in the same work. They also decide on the amount of group aggregation based on several dynamic factors such as query type and network constraints has been modeled using a caching problem. In another paper, they argued that due to the limited bandwidth between edge and central cloud data center, it is not possible for many applications to get the exact result on-time [70]. For wide area streaming applications, they considered accuracy versus timeliness trade-offs in approximate windowed group aggregation at the edge. To show the effectiveness of their approach, minimizing the error under timeliness constraints and minimizing staleness under error constraints have been studied. Our work that is presented in this dissertation complements their efforts by introducing on-demand feedback control loops to adjust data resolution at the edge.

Deng et al. [71] have demonstrated the trade-off between power consumption and delay in fog-cloud environments. Their solution divides the fog-cloud workload deployment into three subproblems, fog deployment, cloud deployment and communication delay, and solves them to achieve best energy-delay trade-off for workload deployment. To verify their theoretical analysis, authors provided extensive simulations and numerical results.
Valerio et al. [72] have explained the impact of fog computing on Hypothesis Transfer Learning (HTL). HTL is a machine learning technique where a model is trained based on learning of disjoint training sets and gathering the partial learning results to reach a unique leaning model. As the number of partial models increases, the accuracy is decreased and best accuracy is achieved when the model is trained using all of the data. The purpose of their work is to show that HTL accuracy for data generated from distributed IoT devices does not change drastically as the number of edge collectors increases. However, the network bandwidth is greatly decreased by distributed collectors at the edge (there is a trade-off between traffic and accuracy).

All of the mentioned edge processing works have focused on various trade-offs for edge-cloud deployment and have emphasized on the importance of edge deployment in energy saving or reducing latency. The work presented in this dissertation complements these papers by using edge resources to execute the tasks, increase the accuracy of the solution, and filter unwanted data close the data source. Next, we study several papers which target the scheduling aspects of edge computing.

In a paper authored by Chun et al. [73], authors came up with a new system called CloneCloud, which automatically transforms mobile applications to take advantage of the cloud environment. CloneCloud automatically changes single machine execution to distributed execution by partitioning the application. It sends a thread to execute part of the application on a remote cloud environment and reports the results back at the edge/mobile devices. The way that CloneCloud partitions the applications is based on static and dynamic analysis and profiling, which considers workload, execution condition and network characteristics. The same aspect have been studied by Kaur et al. [74], which talk about the integration of IoT and edge computing, and propose task selection and scheduling at the edge of the network using container-as-a-service (CoaaS). Task selection/scheduling is done through a multi-objective cooperative game theory approach. They rank the tasks based on the memory and CPU requirements and identify the tasks that can be executed using the containers at the edge and the core of the network. In the same work, a monitoring module has been introduced that only monitors the resource status and helps the scheduler find the free and feasible resources
for scheduling. Their simulation results showed that the energy consumption can be reduced by a small amount of SLA violation.

Pham and Huh [75] explained how fog providers should schedule and offload their applications to the cloud providers’ sites considering workflow’s execution time and cost trade-off. The workflows are defined as DAG and if the workload is outsourced to the remote resources, it would incur data transfer and computational costs at the cloud. They used Cloudsim for modeling and simulation and considered CMT variable (Cost makespan trade-off) in their evaluations. Their simulation results showed that the CMT of their approach is higher than other papers.

Jonathan et al. [7] have introduced Nebula which enables usage of edge voluntary resources in addition to dedicated resources. Nebula uses edge infrastructure for data storage and computation, which have great benefits for applications where data is highly distributed. Moreover, Nebula uses optimization techniques to do location-aware data and computation placement. Finally, their paper concluded that democratize usage of computation and storage resources at the edge and volunteer nodes can massively improve map-reduce execution. In this dissertation, we considered the bandwidth between nodes to identify if they are close to each other and uses such information to map the workflows to the nodes.

Cheng et al. [76] targets a cloud-edge based on-demand stream processing system which automatically configures and manages stream processing tasks. Their model, which is called Geelytics, enables on-demand edge analytics. The key idea in their work is a component called topology master which generates an execution plan based on the request and geographic location of the data producers, consumers, and compute nodes. Our framework also has a master nodes that makes decision about the scheduling of the workflows across geo-distributed nodes.

In contrast to the mentioned papers, we considered the QoS requested by users to schedule and deploy the workload and workflow stages across edge resources. We also optimize the edge deployment based on network usage, execution time, and the cost of execution.
2.1.2 In-Transit Processing

In-transit data processing, also known as in-network data processing, refers to the manipulation and transformation of data while data moves between source and destination. In-transit processing can be advantageous for data intensive applications [34, 46]. Various reactive strategies for in-transit data manipulation have been proposed by several research groups [77, 78]. Several studies have investigated the possibility of coupling various application level components in order to create a cooperative management framework and in-transit data manipulation for data-intensive scientific and engineering workflows [79, 80].

Papers regarding in-situ workflow systems study the problem of visualization for monitoring purposes. Recently, such systems facilitate the coupling of simulation codes with popular visualization and analysis toolkits such as VisIt [81] and ParaView [82]. This expose a broader analytics tools for scientific simulations. The performance-oriented designs for these workflows are becoming increasingly important as they attempt to balance parameters such as latency and run-time performance both for in-situ and in-transit workflow execution [80].

The idea of in-network data processing has been introduced through the topic of active networking. An active network refers to a specific capability to execute tasks within the network over active elements such as switches that have processing capabilities. In this regard, communication patterns are used to address specific user requirements [83]. Implementing patterns in such active switches according to application specific requirements and exploiting packet-based processing within the network has been a key focus of active networking research. Lefevre et al. [84] developed an active network architecture (A-Grid) to support QoS-related metrics for Grid data transport services in addition to other data transport services such as reliable multicast and dynamic service deployment. This architecture employs QoS management at intermediate active routers, and in principal, is similar to the in-transit processing employed in our approach. Understanding how application requirements get mapped to such an architecture has not been fully addressed in existing works. In contrast, our proposed work
maps constraints associated with an application into capability of each component.

In contrast to the mentioned papers, in this dissertation we use Software Defined Networking to extract in-transit network resources and reduce the queuing time for tasks by executing the tasks before data reaches the destination. Unlike the mentioned papers which mostly simulate the in-transit workflow execution, we consider emulation and actual in-transit workflow deployment. We also map the workflow stages to available in-transit nodes and leverage in-transit nodes to monitor workflow execution. Moreover, we use a publish/subscribe method to move data between edge, in-transit, and core resources.

2.2 Wide-Area Analytics

The papers in wide area analytics have focused on effective execution of applications and data processing using resources connected to each other via wide area network links. Papers that are mentioned in this section have aimed to address different challenges and resolve various well-known problems that we are facing in wide-area analytics.

Tudoran et al. [85] targets streaming of events between data centers. Their system referred to as JetStream uses monitoring information of the infrastructure and inter-data center bandwidths. It reacts based on the mentioned monitoring information to improve big data transfers across data centers. JetStream also uses multiple intermediate nodes to enable multi-route data transfer to increase the event transfer rate between two geographically distributed data centers. Moreover, in this paper, the amount of event batching is determined at runtime to decrease latency of event transfers.

Rabkin et al. [48] have proposed another system which is also called Jetstream. Their proposed system addresses wide area stream of queries with a latency bound requirement. This system tries to overcome network bandwidth limitations in streaming engines. Query aggregation which combines multiple queries, and lossy adaptive data degradation are introduced to reduce the data size based on available bandwidth and latency. They test their system with different degradation policies and their results show that their proposed solution can reduce bandwidth usage compared to regular
data streaming execution.

Vulimiri et al. [86] have explained the concept of wide area big data (WABD) and its current main problem which is substantial cross data center network costs. To alleviate this issue in WABD, a Hadoop-based system called WANalytics has been proposed. This system automatically replicates the data and pushes computation toward the edge. The WANalytics has two main components: (i) Runtime layer which coordinates between the central and edge data centers and (ii) A workload analyzer which optimizes data movement and workflow execution and assigns/distributes the workload execution. The workload analyzer estimates the amount of data generated at each stage of the workflow and assigns each stage to edge or central data center based on the estimated information. Our proposed solution extends this model by monitoring the bandwidth between nodes using the delays in data transfer time.

Pixida [87] is a wide area analytics scheduler which aims to minimize the inter-datacenter traffic. When a job (data processing request) is submitted to this system, it gets the task-level graph of the job from users and provides data partition from distributed storage systems. Then Pixida converts the problem of task mapping to the data centers to a graph partitioning and min-k cut problem. By solving this problem, Pixida minimizes the amount of network traffic between data centers by assigning tasks to the best resources.

Geode [88] is an extension of WANalytics which targets the distributed databases across data centers. Geode tries to address wide area analytics of data structures as a SQL analysis while minimizing the bandwidth usage. Geode solves an integer linear program that minimizes the data transfer objective function in workflow mapping. For the standard benchmarks, Geode achieves up to a 360x bandwidth reduction at multi-TB scales compared to centralized solutions.

Iridium [89] explores the minimization of latency in wide area analytics. To achieve low latency query response time, Iridium uses a greedy heuristic optimization technique to find the best task and data placement on geographically distributed data centers. Iridium finds the bottleneck link between the sites and moves the data to that link before it is needed. Also, Iridium executes reduce tasks (in map-reduce ecosystems) on
the appropriate edge sites to reduce the effect of bottleneck links on the overall latency. They have tested Iridium on eight different sites of Amazon EC2 and their results show faster query processing and lower network usage.

SWAG [90] introduces a new job scheduling and reordering approach across data centers which reduces average job completion ratio. The scheduling is based on the coordination that happens between different data centers to finish parts of a job at different geographically distributed resources where the amount of computation at various sites are different. In this dissertation, we address this problem by applying workflow stages to the data as it moves through edge and in-transit nodes.

In this dissertation, we present a novel approach in wide area analytics by scheduling the workload across heterogeneous nodes to minimize network traffic, reduce waiting time at the destination site, and increase job acceptance ratio.

2.2.1 Multi-Cloud Data Processing

Cloud computing is a computing model that allows users and applications to have access to a shared pool of resources. Clouds offer on-demand resources on a pay-as-you-go basis and have a potential for scale-up, scale-down, and scale-out as needed. Hence, they are becoming an increasingly popular infrastructure for enabling large-scale data intensive scientific and business applications [91].

Increase in data size and complexity of data processing tasks, geographic distribution of data sources, and limited resources have led systems to move from centralized data processing systems toward multi-cloud data processing [92]. Through multi-cloud settings, the businesses can distribute their assets across multiple cloud-hosting vendors to achieve high availability [93], reliability [94], fast failure recovery [95], robustness, and fault-tolerance [96].

In this context, such systems that leverage resources from different vendors and data centers can achieve higher granularity in terms of price and resources types and have access to available heterogeneous resources at multiple locations to execute the workloads. In this model, clients/applications are responsible for leveraging and provisioning resources and scheduling their workloads across the multi-cloud environment [97].
2.2.2 Cloud Federation

When a set of cloud providers voluntarily (and through federation regulation) interconnect their infrastructures to allow sharing of resources among each other, a cloud federation is created [97–99]. The main benefit of cloud federation appears when resources are limited for one cloud provider. Such a limitation results in the rejection of the customer requests. Hence, cloud providers can overcome this limitation by outsourcing requests to other cloud providers and members of the federation. Cloud federation can handle spikes on demand and increase in the loads within one site by incorporating resources from other public/private clouds. Also, cloud providers can operate at low utilization and gain revenue by leasing their resources to other providers within the federation [100].

Cloud federation is different from the multi-cloud setting in sense that clients and applications do not need to be involved in load distribution and scheduling. In this dissertation, a cloud federation framework has been used to process the users’ requests and data processing jobs. For this purpose, we used CometCloud [101, 102] as our underlying cloud federation layer.

2.3 Network Abstraction Techniques

In this dissertation, we leverage novel networking techniques such as Software Defined Networks (SDN) and Network Function Virtualization (NFV) to gather information from in-transit network data centers and provision resources on unused resources of the network infrastructure to effectively process the workflows.

2.3.1 Software Defined Network

Software Defined Networking (SDN) [103, 104] is a notion that has brought extensive amount of attention in the community in recent years. This concept was introduced to resolve current limitations in the traditional network infrastructures. These limitations are: (i) lack of ability to change network topology and quickly fix problems within the
networks, and (ii) limited tools that cannot provide a global view of the network topology/nodes. SDN in particular is an approach devised to simplify network management through abstraction of lower-level functionality. Specifically, SDN separates control plane (where data is sent) from data plane (data forwarding functions). This enables the software-based control plane to be run on commodity servers and latest-generation of processors be leveraged, which are faster than embedded-class processors in most switches [5, 105]. This vertical separation reduces the complexities of such devices and makes them simple forwarding devices. The control plane is implemented in a logically centralized controller. Hence, the network (re)configuration and policy enforcement will be much faster and simpler. The users can write applications that are able to interact with the control unit and change the network topology for research and development purposes.

OpenFlow represents both control protocols specified by the Open Networking Foundation (ONF), enabling the network hardware to expose an API to application programs, and specifies a framework through which centralized control of flow forwarding rules can be orchestrated. OpenFlow is added as a feature to commercial Ethernet switches, routers, and wireless access points; thereby providing a standardized hook to allow researchers to run experiments, without requiring vendors to expose the internal workings of their network devices. OpenFlow is currently being implemented by major vendors, with OpenFlow-enabled switches now commercially available [106]. A key benefit of this approach is the ability to make use of spare capacity directly on network elements and couple this with capability within a data center. The coordinated use of these two types of resources provides opportunities for supporting data processing in (near) real time for batch processing and streaming applications. An SDN controller provides applications/users with the ability to potentially install forwarding rules inside a router, monitor flow status, and respond to particular events of interest influenced by the data being transferred [107]. There are several organizations worldwide including
Google\textsuperscript{1}, NDDI\textsuperscript{2}, and GENI\textsuperscript{3} running and testing OpenFlow networks.

In OpenFlow research, Onix\textsuperscript{[108]} is a control plane platform designed to enable scalable control applications. Onix provides advantages in terms of separating the task of network state distribution from applications, and provides them with a logical view of the network state. As in most of the OpenFlow based protocols, Onix provides a general API to control applications, while allowing them to make their own trade-offs among consistency, durability, and scalability. Tootoonchian and Ganjali\textsuperscript{[109]} develop “Hyperflow”, an event based engine for OpenFlow, that allows control applications to make decisions locally by passively synchronizing network-wide views of the individual controller instances. In the same OpenFlow research area, focus on supporting Consistent Updates\textsuperscript{[110]} tackles the problem of state management between the physical network and the network information base (NIB) to enforce consistent forwarding state at different levels (per-packet, per-flow).

SDN plays a key role in cloud federation as it has the potential of addressing issues of data transport service control and communication resource provisioning to meet varying QoS requirements from multiple coupled workflows sharing the same service medium. We envision a SDN model in which network providers offer resources co-located with the network infrastructure to perform complex in-transit analysis on-demand. Using this approach, we have a set of SDN controllers that can gather information about the network status and discover computational capabilities embedded in networks hubs. By integrating the ability of controlling the network with our autonomic mechanisms and federation models, we can obtain greater control over the infrastructure. For example, we can dynamically change the topology of the network and adapt data routes to optimize data migration between source and destination at each step of the workflow.

\textsuperscript{1}http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf
\textsuperscript{2}https://code.google.com/p/nddi/
\textsuperscript{3}http://groups.geni.net/geni/wiki/GENICloud
2.3.1.1 SDN Enabled Cloud Computing

SDNs provide a useful infrastructure to carry out in-transit analysis, where network control plane is decoupled and is directly programmable. This migration of control, which was formerly tightly bound in individual network devices, into accessible computing devices enables the underlying infrastructure to be abstracted for applications and network services, that treat the network as a logical or virtual entity. There are several monitoring tools for SDNs, e.g. [77, 111], which allow the development of more expressive traffic measurement applications by proposing a clean slate design of the packet processing pipeline. These tools focus on efficiently measuring traffic matrix using existing technology and aim to determine an optimal set of switches to be monitored for each flow. Another key benefit of using SDNs is the ability to support and dynamically make available multiple network topologies between source and destination. This is particularly useful within a data center, where multiple types of networks can co-exist.

This dissertation focuses on mechanisms that make use of SDN-based processing capability to improve cloud federation performance. Leveraging SDNs to improve the performance of applications that need to process large amounts of data has been discussed by other authors [112]. Processing large amounts of data using resources from multi-cloud infrastructures and cloud federations requires the use of public and private networking infrastructures. Hence, using SDN to control such resources is beneficial for workload deployment and monitoring. Wang et al. have analyzed the potential performance improvement of the big data application by integrating network control directly within the application [113].

SDN features enhance the performance of cloud computing in many aspects such as adapting to dynamically changing workloads, programmable control logic, providing global view of inter/intra data center network, network resource management and network virtualization [112]. In [114] Xiong et al. show that the integration of SDN and distributed data storage system improves the performance if big data analytic applications. They proposed to leverage SDN visibilities to achieve better performance
in distributed query processing. They proposed solution is implemented using real physical machine and OpenFlow enabled switches. Das et al. propose improving the application performance by selecting particular routes and making more effective use of dynamic links [115] within a network. The authors only provide an overview of their method without any concrete results. In [116], Wang et al. integrated network resources into data center orchestration and provided isolated virtual networks or improve service quality. Their framework, which is called LiveCloud, has been deployed on both public and private clouds. The performance and scalability of LiveCloud controllers are assessed by measuring event processing speed on various core/CPU and network resources. Similarly, Luo et al. [117] and Miyamoto et al. [118] propose to transform network bandwidth into manageable resources and provide guaranteed virtual networks. Furthermore, in-transit computation has been used to aid HPC computation in [119, 120]. In our work, the focus is on improving performance of federated cloud systems by using SDN-based approaches, and unlike mentioned papers, our target is not only considering networking resources, but also dealing with other computational and storage resources through the use of SDN capabilities.

2.3.2 Network Function Virtualization (NFV)

Network Function Virtualization (NFV) [11, 121] extends the as-a-service cloud model to offer on-demand networking functions using virtualization techniques. The context of NFV is based on the decoupling of physical networking assets and the function that these assets provide. The key reason for using virtual machines (VMs) is the possibility of elastically scaling functions by simply adding or removing VMs based on data workload characteristics. This approach promises, as the cloud, a reduction in capital expenses and fast delivery of new functionalities [5].

NFV addresses the issues associated with increasing capital and operational costs of service providers by providing flexible deployment, provisioning, management and joint optimization of network functions and resources [122]. Moreover, NFV brings the flexibility for the network function deployment by allowing sharing and reassigning of the infrastructure. Thus, the hardware and software can play various roles at different
times [121].

In [123], authors proposed techniques to dynamically deploy network functions using NFV and SDN. Through a set of experimental setups, they demonstrate that the software-controlled network is able to reconfigure the virtualized network functions to guarantee the agreed SLAs to the user. In a different work, Souza et al. [124] allocate VMs connected by virtual infrastructure. They proposed a relaxed mixed integer programming model to enforce QoS constraints and reduce latency in SDN/NFV-enabled cloud data centers.

Bomafiglia et al. [125] proposed a framework that uses NFV techniques to deploy network function at the edge and cloud data centers, and the networks that connects these functions together. They used OpenDaylight as their SDN controller and OpenStack to control the deployed resources. The orchestrator provides end-to-end service orchestration. This work is partially similar to what we have considered to deploy the workflows.

2.4 Quality of Service for Workflow Applications

Data processing and delivery with end-to-end QoS constraints has been explored in different papers. Karim et al. [126] maps the user’s QoS to the Software as a Service (SaaS) layer by developing a hierarchical QoS model and assigning QoS weights. In [127] authors took another approach to control the execution environment at runtime by finding a service composition that meets QoS and recomposing the services at runtime, if necessary. Bhat et al. [46] investigated in-transit data manipulation and proposed reactive strategies to achieve higher QoS, even in congested network conditions. Processing data within the deadline and budget constraints has been investigated in [128] that targets cost-time optimization techniques to schedule the workflows. This is complementary to our work. Yu et al. [129] proposed a genetic algorithm [130], meta-heuristic inspired by the process of natural selection, targeting heterogeneous and reservation based service-oriented environments to schedule the workflows under deadline and budget constraints. The scheduling result of their proposed approach significantly improves
the result compared to heuristic approaches. Maswood et al. [131] developed a mixed integer linear programming method to allocate resources by minimizing location dependent costs. Using numerical evaluations, authors showed that their approach reduces the provisioning cost and energy consumption.

On the other hand, our proposed framework combines both static and dynamic approaches to provide end-to-end QoS. The static approach is the workflow stage mapping to find the best resources that meet deadline and budget, while the dynamic approach uses on-demand feedback loops to check the execution of the streaming workflows at runtime. In our framework, we consider this idea to provide end-to-end QoS by reducing the resolution of the data. However, the decision about this trade-off is made using on-demand control loops created after execution of each stage on edge and in-transit nodes.
Chapter 3

A Model to Leverage the Combined Use of Computational Capabilities Available at the Network Edge and between Computing Resources

3.1 Introduction

Combining IoT and Cloud computing capability enables the creation of smart environments that can respond to real-time events, by (a) combining services offered by multiple stakeholders (i.e. those that are at the network edge with services provided within a data center) and, (b) providing scale to support a large number of users in a reliable and decentralized manner. They need to be able to operate in both wired and wireless network environments and deal with constraints such as access devices or data sources with limited power and unreliable connectivity. The Cloud application platforms need to be enhanced to support (a) the rapid deployment of services by providing domain specific programming tools and environments and (b) seamless execution of applications harnessing capabilities of multiple dynamic, and heterogeneous, resources to meet quality of service requirements of different users.\(^1\)

Additionally, network operators are increasingly becoming potential providers of general purpose computation infrastructure. They are minimizing the amount of network-specialized hardware hosted in their data centers and moving towards the use of commodity hardware. This strategy is supported in recent efforts in Software Defined Networking (SDN) and Network Functions Virtualization (NFV). SDN, in particular, is an approach devised to simplify network management through abstraction of lower-level functionality. Specifically, SDN separates control plane (where to send data) from

\(^1\)The results presented in this chapter have been published in IEEE Transactions on Services Computing [1].
data plane (data forwarding functions). This enables the software-based control plane to be run on commodity servers and to leverage the latest-generation of processors, which are faster than embedded-class processors in most switches [36]. On the other hand, NFV goes a step further and extends the as-a-service cloud model to offer networking functions on-demand using virtualization techniques. This approach promises, as the cloud, a reduction in capital expenses and a rapid deployment and delivery of new functionality [37].

Data centers managed and operated by network providers form a significant part of the current Internet infrastructure, as there is a large number of such data centers that are almost ubiquitous across the world. These data centers may not be as powerful as computational data centers, hosted by cloud providers or traditional high performance computing (HPC) providers. However, their ubiquity and the fact that we have to necessarily use them, when moving data over the Internet, makes them a useful source of pervasive computing at the edge of the network. Understanding how the availability of commodity servers within such “network data centers” can contribute towards data processing would enable an effective way to extend the boundaries of a cloud system – from a high end, often localized data center, to multiple distributed data centers that can process data while it is in transit from source to destination. This also provides the possibility of additional revenue models for network providers – who are able to convert underutilized network resources to offer in-transit computation.

In this chapter we propose a model to leverage the combined use of computational capabilities available at the network edge and within network data centers to support data transformation and analysis from source to destination. We demonstrate how this can lead to more efficient use of computational resources and extend the capability and capacity of the overall infrastructure. The contributions of this chapter are:

- An in-network computational model to leverage computational resources located at the edge, within the network and those at a traditional cloud data centre.

- An optimization strategy that allows us to prioritize data processing based on the expected value of the data to user. We describe how this subjective notion can
influence the location of where data processing takes place.

- An experimental and analytical validation of the proposed model using a video analysis use case.

The rest of the chapter is organized as follows. Section 3.2 presents our motivating use case. Section 3.3 presents our federation model. Section 3.4 formalizes our problem, and Section 3.5 proposes an scheduling optimization strategy. Section 3.6 describes the deployment used for our experiments. Section 3.7 presents our evaluation and results. Finally, Section 3.8 discuss the results obtained.

### 3.2 Video Analytics Use Case

We describe use cases centered on processing video sequences submitted from a single/multiple camera(s). These video sequences can be encoded using different formats, and need to be processed within a deadline. We therefore consider a semi-real time video sequence analysis – compared to “batch” analysis, where a video sequence is first archived and subsequently analysed in an off-line manner. Two aspects of video stream processing (consisting of a sequence of image frames submitted from a camera) can be considered: (i) the stream is viewed and stored for archiving – generally requiring the captured data to be transmitted over a network infrastructure for viewing/archiving purposes; (ii) the stream is processed (and annotated) using pre-defined filters (to support object detection & colour-based classification, template matching, etc). Operations associated with (i) are often seen as a precursor to those for (ii). Video analytics also involves a user identifying features/events of interest to be considered in the video sequence – such as detecting objects of interest, size/colour-based classification, potential area of interest (spatially), and an estimated duration associated with such events. Figure 3.1 presents the high level stream processing workflow. Video sequences can be encoded using a variety of different formats (Full HD, QCIF, CIF/4CIF/D1, H.264, etc). Each encoding (generally at 25 frames/sec) leads to different storage requirements and number of pixels per frame. The level of automation involved in analysing the video stream can also vary. From full automation, where a user defines an “analysis request”
and does not require further interaction with the system, to an interactive request, where a user is able to see partial results at each stage of analysis and able to interact and modify analysis parameters. In sections 3.2.1 and 3.2.2 we describe how (partial) video analysis can be carried out at the capture site and at in-transit nodes (located between the capture source and a Cloud-based data centre). The base line scenario (for comparison) is that all data is migrated from capture source to the datacenter for analysis. Figure 3.2 illustrates the baseline scenario.

Figure 3.1: High Level Stream Processing Workflow. ©2017 IEEE (reprinted with permission) Zamani et al. [1].

Figure 3.2: Video Analytics Scenario – adapted from [2]. ©2017 IEEE (reprinted with permission) Zamani et al. [1].
3.2.1 Scenario 1: Processing at Capture Node (Edge)

In this scenario, all data captured at the source is pre-processed at the source, prior to transmission across a network. Co-located hardware enables the captured data to be analysed before sending the processed results to the data centre. Processing at capture site can include: (i) data compression (at the camera) & buffering; (ii) data sampling – also as a means to support data size reduction; (iii) tagging of video frames prior to transmission. It may also be useful to combine data feeds from multiple cameras, to support correlation across multiple capture sites. Such aggregation and analysis would be useful at the first hop network component from the capture site.

3.2.2 Scenario 2: Processing at In-transit Nodes

In this scenario, data captured at the source is channeled through a number of intermediate “network data centres”, prior to arrival at a video processing data centre. Each intermediate network data centre processes the data enroute – depending on the computational capability available. Subsequently, the data is archived and processed at the data centre, expected to have a much greater computational capability. The capability made available at the network data centre can vary over time, influenced by the other data streams that are being channeled through it at any time.

To make more effective use of the entire computational infrastructure, we propose that rather than sending all unprocessed data to a centralized location for processing, it is more efficient to initiate data processing at the edge of the infrastructure. Such an infrastructure includes data capture and generation devices, and the network path to its destination – e.g. using IoT gateways or devices, SDN switches, network data center, and clouds. In this way, we can incrementally augment the relevant information contained in the data, potentially reducing its size, while the data is being moved from source to destination. Additionally, this approach enables computational resource sharing, which not only improves resource utilization and throughput, but also increases the resilience to failures. Such an approach can also reduce latency and processing times resulting from unpredictable data (generation) sizes.
3.3 Resource Federation Model for Video Analytics

We extend CometCloud federation model [132] to expose in-transit and edge capabilities to participant sites of the federation. Figure 3.3 shows our architecture. We include a service, called Controller, that is aware of the network topology, using SDN technology, and it also has information about the available computational capabilities of each network data center. Each data center has an SDN router that is managed by the Controller and a set of resources to process tasks. The Controller can be consulted by the sites to optimize workload scheduling using the strategies proposed in Section 3.5.

This federation model is built using the CometCloud framework [102]. CometCloud is an autonomic framework for enabling real-world applications on software-defined federated cyberinfrastructure, including hybrid infrastructures integrating public and private Clouds, data-centers and Grids. The CometCloud federation is created dynamically and collaboratively, where resources/sites can join or leave at any point, identify themselves (using security mechanisms such as public/private keys), negotiate terms of federation, discover available resources, and advertise their own resources and capabilities [133].

Our federation model is coordinated using CometSpaces [134] at two levels. Comet-
Spaces provide a tuple-space like abstraction for coordination and messaging in the federation model – internally it implements a publish/subscribe messaging layer and an information lookup system built on a content-based distributed hash-table (DHT) based on a structured peer-to-peer overlay. First, a single management space (*Comet-Cloud Federation Space*) spans across all resource sites creating and orchestrating the federation. This space is used to exchange any operational messages for discovering resources, announcing changes at a site, or routing users’ request to the appropriate site(s). Second, multiple shared execution spaces (*SE-Space*) are created on-demand during application workflow executions to satisfy computational or data needs. Execution spaces can be created within a single resource site, or can burst to others, such as public clouds or external HPC systems.

Computational resources of our federation support at least a CometCloud Master, which acts as an agent or broker between local resources and the rest of the federation. It is also responsible for accepting computational requests from users and edge devices that want to access the federation. CometCloud Masters interact with the rest of the federation through the federation management space in a publish/subscribe fashion. Each CometCloud Master publishes information about the status of its resources, the services they offer, or computational needs of its users. Additionally, a CometCloud Master creates subscriptions to be notified when there is some event of interest, such as a request for computation. A CometCloud Master evaluates each request and decides if it can process it within the given QoS requirements, in which case it temporarily reserves the resources and answers the request with various details defining the Service Level Agreement (SLA), such as completion time and cost. If the client agrees to the SLA, then the computation proceeds, otherwise resource reservation is eliminated. In order to process a request, a CometCloud Master might create a SE-Space where it inserts the tasks and deploy CometCloud Workers to actually compute the tasks. In Edge and In-transit resources there are memory limitations, hence the SE-Space is not deployed and tasks are consumed as a stream by the CometCloud Workers.

As illustrated in Figure 3.3, data captured at an edge resource (labelled E) is either
directly submitted to a first hop gateway/router \( R \), or pre-processed prior to transmission. Each \( E \) supports a CometCloud Master \( M \). The first hop router can also aggregate data streams from multiple edge resources. This data is subsequently forwarded across a chain of in-transit resources (labelled as \( R \)) to a data center (labelled Site \( i \)). Each in-transit resource, similar to a data center or edge device, must support a CometCloud Master – but with varying resource capability. At the data center, the Master also communicates with a number of Workers \( W \) which are responsible to process the tasks. In this way, our resource federation can be logically seen as a collection of CometCloud Master nodes, which interact with each other to achieve the optimization objectives outlined in Section 3.4.

### 3.4 Problem Formulation

Let us consider a set of surveillance cameras \( C : \{c_1, ..., c_n\} \), each of which generates a video stream that needs to be processed in a timely manner. A video stream can be partitioned into a sequence of \( m \) chunks, where each chunk contains a number of image frames. Processing each of these chunks is considered a computational job in our system – hence a given camera \( c_u \) is going to generate a set of jobs \( J_u : \{J_{u1}, ..., J_{um}\} \).

These jobs are introduced in the system periodically, as a sequence, i.e. if we consider 12 second chunks, then every 12 seconds a new job enters the system. Any given job \( J_{ux} \) is processed by a sequence of stages \( \{s_1, ..., s_z\} \), forming a workflow as described in Section 3.2, Figure 3.1. Each stage is composed of \( t \) tasks \( \{j_1, ..., j_t\} \). The number of tasks depend on the size of the video chunks (e.g. in the classification stage we can have a task per frame and there are 25 frames in a second of video). The location of a camera is defined as source \( s \). At the camera controller or aggregator, we need to decide which part is computed locally, in-transit, and/or at the cloud. We consider that clouds are located at multiple network hops from the data source, at the core of the infrastructure.

We consider that the video chunk and/or processed results need to be sent to a specific data center for storage and potentially additional offline processing with older data. This data center is defined as destination \( d \) in our system. The service level agreement (SLA) of a job \( J_{ux} \) includes: a deadline \( (\text{Deadline}(J_{ux})) \) by which results have to be
placed at the destination – this is typically determined by the size of the video chunk; and a budget ($Budget(J_{ux})$) that sets the maximum amount available to the user to spend on computing job $J_{ux}$.

Central to our approach is the concept of value associated with the processing of data – a subjective criterion identified by a user. We define the value of data (i.e. a video chunk in our example application) as the significance a user associates with processing of particular data items within a deadline (captured as a subjective probability), in preference to other data items. For instance, in a surveillance scenario, the value associated with processing a video sequence (to detect/classify objects) would be higher if there was a public event in progress. Therefore, in our approach we pay attention to the value parameter to prioritize the processing of video streams by, for example, allocating high value workload closer to the data source and allocating low value workload to cheaper resources or rejecting low value workload when there are insufficient resources available.

We define three types of computational resources forming our federated infrastructure, namely edge devices (local to the data capture site), network data centers (in-transit resources), and computational data centers (cloud resource providers or sites). Formally, we define these resources as a set $R$ with $q$ resources $\{r_1, ..., r_q\}$. We assume that SDN components are present in our infrastructure to ensure dynamic control over the network and provide QoS guarantees. The following symbols are used to characterise the problem:

- $P(r_i)$ is the average number of tasks that resource $r_i$ completes per unit of time.
- $E(J_{ux}, r_i)$ is the time job $J_{ux}$ spent computing at resource $r_i$.
- $BaseCost(r_i)$ is the cost per unit of time for using resource $r_i$ for computation.
- $T(r_i, r_k)$ is the time spent transferring data between resources $r_i$ and $r_k$.
- $BaseCostNet(r_i, r_k)$ is the cost of reserving a network channel per unit of time, between resources $r_i$ and $r_k$.
- $Value(J_{ux})$ is the value obtained from processing any given job $J_{ux}$. 
The overall time needed to process a job $J_{ux}$ is defined as:

$$\text{CompTime}(J_{ux}) = \sum_{i}^{q} E(J_{ux}, r_i) + \text{Transfer}(J_{ux}) \quad (3.1)$$

where $\text{Transfer}(J_{ux})$ is the sum of the time spent transferring data between resources ($r_i \in R$), where the first resource is located at the source of the data $s$ and the last one is the destination $d$.

$$\text{Transfer}(J_{ux}) = \sum_{i}^{q} \sum_{k \neq i, k} T(r_i, r_k) \quad (3.2)$$

The overall cost of computing job $J_{ux}$, $\text{Cost}(J_{ux})$, is defined as:

$$\text{Cost}(J_{ux}) = \text{CostExec} + \text{CostNet} \quad (3.3)$$

where the computational cost ($\text{CostExec}$) is defined as:

$$\text{CostExec} = \sum_{i}^{q} [CE(r_i) \cdot E(J_{ux}, r_i)] \quad (3.4)$$

$$CE(r_i) = \text{BaseCost}(r_i) \cdot (1 + \frac{1}{Ratio_i}) \quad (3.5)$$

$$Ratio_i = \frac{\text{Capacity}_{r_i}}{\sum_{j=1}^{q} \text{Capacity}_{r_j}} \quad (3.6)$$

The cost of a resource $r_i$ is defined by $CE(r_i)$ in Equation 3.5. This cost varies depending on the ratio between the capacity of $r_i$ ($\text{Capacity}_{r_i}$) and the total capacity of the set $R$ of resources ($\sum_{i}^{q} \text{Capacity}_{r_i}$). This ratio is represented by $Ratio_i$. The larger the capacity of a site, the lower the cost and the other way around.

The cost of transferring data associated with a job ($\text{CostNet}$) is defined as:

$$\text{CostNet} = \sum_{i}^{q} \sum_{k \neq i, k} [T(r_i, r_k) \cdot \text{BaseCostNet}(r_i, r_k)] \quad (3.7)$$

subject to $E(r_k) \neq 0$.

In this work, our objective, from the infrastructure's perspective, is twofold. On the one hand, we want to maximize the throughput of our infrastructure, defined as maximizing the overall number of jobs processed by the system – as described in Equation 3.8. On the other hand, we want to maximize the overall value obtained from the
processed data – as described in Equation 3.9.

\[
\begin{align*}
\max & \sum_{u} \sum_{x} \sum_{i} P(r_i) \times E(J_{ux}, r_i) \\
\max & \sum_{u} \sum_{x} Value(J_{ux})
\end{align*}
\] (3.8)

(3.9)

where the overall objective considers all cameras \( n \in C \) and all jobs \( m \in J_u \) generated by each camera \( c_u \) are inserted into the system. These objectives are subject to ensuring the QoS requirements of each processed job, which is detailed in Section 3.5, Equations 3.13 and 3.14.

It is important to clarify that we do not associate particular operations (job executions) with particular resources in our system, i.e. it is not necessary for all collected data to be pre-processed at edge devices prior to their transmission to network or the data center resources. Data pre-processing, for instance, could be carried out on any resource depending on their capability, capacity and cost. The proposed architecture would be most effective if similar types of operations could be executed across all the devices in the system (with varying QoS profiles, depending on device type). In the limiting case, it may become necessary to carry out all such analysis at the data center, with edge and network devices primarily enabling data capture and transmission. However, the aim of the optimization process is to push some of these operations to the edge of the network, whilst not violating some of the other constraints associated with application execution deadline or cost.

### 3.5 Scheduling Optimization Strategy

To achieve the previously described objectives, we add an additional stage to the workflow described in Figure 3.1. This stage is used to estimate the expected value of a video chunk, which represents the likelihood of finding relevant information within this chunk. This value is used to perform a systematic sampling that reduces the size of data without affecting its content. In practice, the expected value can be estimated using historical information combined with the current status of the recorded area.

The semantics of value can change for different uses of video analysis. We consider
two main aspects that influence this parameter: (i) static: these include characteristics such as the importance of a particular video source (e.g. position/geo-location of a particular camera), the video sequence captured during a particular event within a particular time window (e.g. a football event), etc. These characteristics are therefore known apriori, i.e. before the analysis is carried out; (ii) dynamic: these include characteristics that are derived after a part of the video sequence has been analysed – e.g. detection or classification associated with particular types of objects, with such object(s) not being known before analysis commences. Such dynamic aspects generally require an interactive analysis of a video sequence. In this work, we primarily focus on (i), although the workflow we propose could also be extended to (ii), but would require human/operator assessment during the workflow.

Figure 3.4 shows the workflow considered in this work, with the extra stage to assess value. We define high value (HV) and low value (LV) video chunks as follows:

\[
Value(J_{ux}) \begin{cases} 
[0.5, 1] & \rightarrow \text{High Value (HV)} \\
[0, 0.5) & \rightarrow \text{Low Value (LV)}
\end{cases}
\]  

(3.10)

The value of a video chunk, \( Value(J_{ux}) \), is used to decide how much data of this chunk we keep. Specifically, we perform a systematic sampling that reduces the size of the data. This sampling interval \( k \) is calculated by \( \frac{N}{y} \), where \( N \) is the total number of frames and \( y \) is the sample size. The sample size \( y \) is calculated as follows:

\[
Value(J_{ux}) \begin{cases} 
[0.5, 1] & \rightarrow \quad y = \lfloor Value(J_{ux}) \times N \rfloor \\
[0, 0.5) & \rightarrow \quad y = \lfloor 0.5 \times N \rfloor
\end{cases}
\]  

(3.11)

Moreover, the value of a video chunk is also used to decide how a job should be scheduled. Currently, we consider two strategies: (i) minimizing the computational
time required to process a job; and (ii) minimizing the cost of computing the job.

\[
\begin{align*}
Value(J_{ux}) &= \begin{cases} 
[0.5, 1] \rightarrow (i) \min (CompTime(J_{ux})) \\
[0, 0.5) \rightarrow (ii) \min (Cost(J_{ux})) 
\end{cases} 
\end{align*}
\] (3.12)

both scheduling strategies are subject to performing computation within the given deadline (3.13), and keeping costs within the given budget (3.14).

\[
\begin{align*}
CompTime(J_{ux}) &\leq Deadline(J_{ux}) \\
Cost(J_{ux}) &\leq Budget(J_{ux})
\end{align*}
\] (3.13, 3.14)

In fact, our admission control strategy enforces Equations 3.13 and 3.14 by only accepting those jobs that can be completed while satisfying these constraints.

### 3.6 Configuration of Testbed

We configured a testbed using our previously proposed multi-layer computational model (Section 3.3). In this model, we have three different kinds of computing resources: (i) Edge resources, close to data source; (ii) In-transit resources, close to data in movement; and (iii) Core data centers or sites, located deep into the infrastructure and far from data sources. We have used AWS EC2 cloud platform to emulate an actual scenario where resources are virtual machines and the network is controlled using Mininet \(^2\) and Linux traffic control. Specifically, we used a total of 11 VM instances that emulated different geographically distributed sites, as described in Figure 3.5. Two VMs represented camera aggregators, located at the edge of the infrastructure, named Source1 and Source2. Another VM represented the datacenter where results are ultimately stored, named destination. The other eight VMs were in-transit resources: Mid1 through Mid8 – located between sources and destination, see Figure 3.5. Specifically, four in-transit resources (Mid1, Mid2, Mid5 and Mid6) were located along the path from Source1 to the Destination, and the other four (Mid3, Mid4, Mid7 and Mid8) were located along the path from Source2 to the Destination.

---

\(^2\)mininet: http://mininet.org
Our deployment emulated an actual geographically distributed infrastructure by configuring network bandwidths connecting the sites and the performance of the computational resources according to experimentally obtained information (between Rutgers and AWS East region). Table 3.1 summarizes the characteristics of the resources at each level of our infrastructure. We considered each worker had the performance of an Amazon EC2 c4.xlarge instance, with a base price of $0.21. The hardware characteristics of this instance match the one used to characterize the workload [2]. The price was calculated using the cost model defined in Equation 3.5. Equation 3.5 assumes that computational resources are limited and adjust the price based on the total capacity of each type of resource, the fewer resources, the more expensive is to reserve them and vice versa.

All instances were deployed with Mininet and the network between them was configured to emulate a SDN environment among these 11 mininet instances. Each VM
Table 3.1: Computational Resource Properties. ©2017 IEEE (reprinted with permission) Zamani et al. [1].

<table>
<thead>
<tr>
<th>Resource</th>
<th># Workers per Site</th>
<th># Sites</th>
<th>Price ($/Hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
<td>1</td>
<td>2</td>
<td>4.83</td>
</tr>
<tr>
<td>In-transit</td>
<td>2</td>
<td>8</td>
<td>2.52</td>
</tr>
<tr>
<td>Core site</td>
<td>4</td>
<td>1</td>
<td>1.365</td>
</tr>
</tbody>
</table>

had one mininet host and one mininet switch. Switches were connected to each other using Generic Routing Encapsulation (GRE) tunneling[135]. Bandwidth allocation for data links was implemented in the hosts using a token bucket filter. Routing tables and connections were controlled by a POX SDN controller (POX is a python based SDN controller). We had an additional VM designated as the controller of the network. The controller managed network connections using two types of connections: (i) UDP was used for gathering information; and (ii) TCP was used for regular communication and establishing data paths. TCP rules for each switch were installed in a proactive manner. That is, every time a switch connected to the controller (i.e. when switch starts), the controller would install rules (as described below).

We implemented our in-transit optimization approach, described in Section 3.5, as follows. We defined our protocol for communication between the controller and hosts using UDP packets. We established that switches would forward all UDP packets to the controller unless a specific destination was included in the packets. We used this rule to enable communication between client and controller. Specifically, when the source site (i.e. client) wanted to communicate with the controller, it would send a UDP packet without destination field instantiated. This packet would be automatically forwarded to the Controller. Upon receipt of this packet, the controller would send UDP packets to all in-transit resources (hosts) asking for their status and capabilities information. Since those UDP packets would have a specific destination, switches knew where to send them (i.e. in-transit resources). Then, the controller would gather all replies, analyze their information, and create a plan. This plan was returned to the client and included
a data transfer path as well as information about the allocated in-transit computation.

For example, in the case of the video processing environment, each camera aggregator collects video from the cameras and decides where the workload is computed according to the established policies and resource availability. Using the network configuration defined above (in this section), camera aggregators can transparently contact SDN controllers (without knowing their location or address) to obtain a view of the infrastructure and take operational decisions. This is achieved by simply sending a UDP message without destination. Using this approach, the use of SDN does not involve complex changes in the client and data producers. As mentioned before, if SDN controller is implemented as a logically centralized physically distributed manner, this model would be highly scalable.

3.7 Evaluation

In this work, we considered that each camera aggregator (Source) had three cameras capturing and sending video to them. Specifically, at each camera aggregator we had one camera capturing video with QCIF quality, another with CIF quality, and another one with 4CIF quality. We considered that the video feeds were sent in chunks of 48 seconds, hence each camera generated a new video processing job every 48 seconds. A second of video had 25 frames, where all of these video frames were independent of each other, from an object detection perspective. Table 3.2 summarizes the application characteristics in terms of execution and data size for different quality of video – the application was characterized by Anjum et al. in [2]. We used the execution time and data size to assign a deadline to each type of video to ensure timely delivery of results. We performed two sets of experiments using different deadlines to observe the behavior of the system and to understand the capacity of the infrastructure when processing our use case application. Additionally, we validated our model against results obtained from experiments and analytically studied the effect of changing various parameters in the system. In order to evaluate the model analytically, we implemented the mathematical model and the optimization strategy proposed in Sections 3.4 and 3.5 into a custom made Python simulation.
Table 3.2: Video Stream Analysis Time and Characteristics obtained from [2]. ©2017 IEEE (reprinted with permission) Zamani et al. [1].

<table>
<thead>
<tr>
<th>Format</th>
<th>Decode</th>
<th>Analysis</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCIF</td>
<td>0.4</td>
<td>4 s</td>
<td>80 MB</td>
</tr>
<tr>
<td>CIF</td>
<td>2 s</td>
<td>12 s</td>
<td>320 MB</td>
</tr>
<tr>
<td>4CIF</td>
<td>4 s</td>
<td>40 s</td>
<td>1200 MB</td>
</tr>
</tbody>
</table>

Each experiment lasted around 30 minutes. During this time, each camera sent a total of 39 video processing jobs. As each camera was generating data with different encoding formats, the computational jobs were also heterogeneous in both computational and data transfer needs. Once a job was generated, the camera aggregator had to decide where to execute the job, if possible. The camera aggregator also estimated the expected value of each job. This value can be estimated using historical information or current status of the recorded areas. In our experiments, we used a random distribution to assign an expected value to each job. Figure 3.6 shows the value distribution for each type of job.

Figure 3.6: Value Distribution for Each Job Type. ©2017 IEEE (reprinted with permission) Zamani et al. [1].

We evaluated three different scenarios:

**Cloud (C):** This scenario considered a traditional approach where all data was transferred to a large central data center for processing. Thus, every time a job was generated the camera aggregator asked the data center whether it was possible to complete the job to meet the pre-established SLA guarantees. If the job was accepted, then its data
was transferred to the data center for processing. The scheduling policy of this scenario was to minimize cost while meeting the deadline.

**In-Transit plus Cloud (I+C):** In this scenario we added a layer of computational resources to help the central data center process the workload. In this case, the camera aggregator asked the SDN controller to optimize the route from source to destination and determine the most efficient job execution (e.g., part of the job, i.e. tasks within the job, can be processed at one or more in-transit sites and part at the destination data center). The scheduling policy of this scenario was to minimize the cost of low value jobs by sending them to the core cloud data center, and to minimize the completion time of high value jobs by using in-transit resource(s) whenever possible. As before, jobs that could not be completed within the deadline were rejected.

**Edge plus In-Transit plus Cloud (E+I+C):** In this scenario, we added one more layer of processing (the layers that have been existed before but have not been leveraged yet) to enable computation at the edge of the infrastructure (i.e. the camera aggregators). Specifically, we implemented the optimization strategy proposed in Section 3.5 to perform (some limited) computation at the edge. This computation involved performing a systematic sampling to reduce the size of the job (i.e. dropping frames). Sampling is a popular approximate computing technique for image processing, and it has shown to be very effective in accelerating computation while keeping the error of the solution within acceptable margins [47]. Next, jobs were scheduled across in-transit and cloud resources minimizing the cost for low value jobs and minimizing the completion time for high value jobs.

Due to capacity constraints, we considered that edge devices were not able to queue jobs and therefore had to push jobs to the next hop resource (an in-transit resource). On the other hand, we considered that cloud and in-transit resources had more capacity and therefore were able to queue jobs for processing.

The remaining subsections describe our experimental results for each of these scenarios.
3.7.1 Experiment 1 – Deadline based on completion time

In the first set of experiments we used a deadline for each type of job that is 50% higher than its minimum completion time (execution plus data transfer) in the cloud scenario. Thus, the deadline for QCIF is 12 seconds, for CIF is 45 seconds, and for 4CIF is 156 seconds. These experiments were executed in a deployment of our CometCloud framework in Amazon EC2, as described in Section 3.6. Figure 3.7 illustrates the
Figure 3.7a compares the job acceptance ratio, represented as the percentage of jobs accepted for processing compared to the total number of jobs submitted. We can observe that the traditional approach of sending all data to a central data center located at the core of the infrastructure (labeled as C) was only able to process a small number of the required jobs. However, by adding an additional layer of processing to use resources located along the data path from source to destination (in-transit resources), the infrastructure was able to significantly increase the number of accepted jobs – especially smaller jobs (results labeled as I+C). This was not only due to the additional computational resources, but also to the fact that data was being processed earlier and its size was reduced. This contributes to reducing large waiting times, which allowed the acceptance of small jobs (QCIF and CIF) that in the first scenario had to be rejected due to potential violation of their deadlines. The last experiment introduced edge resources (E+I+C) and the possibility of doing some computation in-situ (where data was being generated). In this case, we observe that the infrastructure was able to further increase the number of completed jobs – only rejecting around 20% of the QCIF jobs.

As part of our optimization strategy, we wanted to increase the number of high value jobs processed by the infrastructure, as they were expected to provide us with more relevant information. Thus, Figure 3.7b compares results showing the value associated with accepted and rejected jobs. We observe that by simply using a different scheduling policy depending on the value parameter, we were able to prioritize high value jobs. In the I+C (In-transit plus Cloud), we accepted all high value jobs and only rejected 30% of the low value jobs. Additionally, in the E+I+C (Edge plus In-transit plus Cloud) we were able to accept all high value jobs, rejecting less than 17% of the low value jobs. In general, we can conclude that adding additional layers of computation closer to the data, improves the performance of the infrastructure and minimizes the network bottlenecks.

Figure 3.7c compares the completion time of all jobs in the system, calculated as the time since a job was inserted until it was processed. Figure 3.7c show how different
scenarios influence the average completion time of jobs. It is worth noting the impact of our scheduling approach, depending on the value of the data, in I+C scenario. In this scenario we did not have any filtering of the video frames. However, the value of a job was used to decide the way such a job was scheduled, which in practice prioritized high value jobs. As a consequence, the average completion time of high value jobs was up to a 7% lower than the average completion time of low value jobs. Alternatively, in the E+I+C, we observe that the average completion time of high value jobs was 60% higher than low value jobs. The main reason was that during the sampling phase, high value jobs had more data compare to low value jobs and therefore the execution took longer. However, we observe that the completion time was significantly reduced when compared with the C and I+C scenarios (up to a 50%). In general, we observe that by adding edge and in-transit resources, we were able to reduce the completion time of all jobs compared with the approach of using only cloud resources (the C scenario) at the core of the infrastructure.

The completion time of a job was composed of three components, namely waiting time, execution time, and data transfer time. The waiting time or queue time, is defined as the time that a job spends waiting to be executed. Since our infrastructure is a multi-queue system, a single job may have to wait in more than one queue. Thus, the waiting time of a job was calculated as the sum of its waiting time in every queue it was scheduled. Figure 3.7f collects the waiting time (queue time) of the jobs. As we can observe, in the I+C and E+I+C scenarios, the average waiting time of the high value jobs was up to an 77% lower than the waiting time of low value ones. In the C scenario (cloud), only the large jobs 4CIF, which had a large enough deadline, were able to wait in the queue, while small jobs (QCIF and CIF) were penalized and rejected due to having a very short deadline in comparison with the large jobs.

Next we analyzed the impact of the in-transit and edge resources on the execution time of the video processing jobs. Figure 3.7e demonstrates how our strategy of performing sampling and preprocessing of jobs at the edge helped to reduce the amount of execution needed for video processing jobs. Similarly, we observe in Figure 3.7h that the average amount of time spent transferring data between source and the place of
Figure 3.8: Summary of experimental results. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. Deadlines are: QCIF = 48s, CIF = 48s, 4CIF = 120s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10.

computation was, in average, between 18% and 64% lower when using edge computation. As we can observe the network had a strong influence on the completion time of jobs, being at times larger than the execution time.

Finally, we also analyzed the impact of the scenarios on the total cost of the jobs. Table 3.1 collects the price of each resource per unit of time. In our model, the fewer resources a site have, the higher its price. In this way, performing computation at the
edge is much more expensive than performing the same computation in a core cloud data center (composed by a large number of resources). Figure 3.7d collects the results. We observe that, in the I+C and E+I+C scenarios, low value jobs (scheduled aiming at minimizing cost) were typically computed at the destination (core cloud data center). However, high value jobs (scheduled aiming at minimizing completion time) chose to compute using edge and in-transit resources when they were available, which increased the price of computation. We also observed that in the E+I+C scenario, the average cost of the jobs was lower than in the I+C scenario. These savings were caused by the use of a sampling technique at the edge, which reduced the amount of data to be processed.

3.7.2 Experiment 2 – Deadline based on video size

In this experiment, we increased the deadline of the jobs of type QCIF and CIF to 48 seconds, which matches the size of the video to be processed, and in practice means near-real time processing of video feeds. Whereas in Experiment 1, we mapped deadline to minimum completion time, in this experiment we relate deadline to the size of the video to be processed. At the same time, we set the deadline of 4CIF job type to 120 seconds to reduce the significant waiting time observed earlier. Large waiting times for this type of jobs prevented us from accepting more small jobs (QCIF and CIF) as their deadline could not afford the wait. This experiment was executed in a deployment of our CometCloud framework in Amazon EC2, as described in Section 3.6 Figure 3.8 compares the results.

Figures 3.8a and 3.8b compares the results of admission control mechanism used in the infrastructure. We observe that by increasing the deadline of the smaller jobs, and specially reducing the deadline of the large jobs (4CIF), the Cloud scenario (labeled C) was able to increase the number of accepted jobs, resulting in a 45% acceptance ratio. On the other hand, the I+C scenario (In-transit plus Cloud) rejected 27% of low value jobs, while E+I+C scenario (Edge plus In-transit plus Cloud) only rejected 1.5% of the low value jobs. In both cases, all high value jobs were accepted.

Figure 3.8c compares the completion time for this experiment. We can observe that,
despite increasing the deadline to the QCIF and CIF job types, their average completion
time was similar to the one observed in Figure 3.7c. In fact, we observe in Figure 3.8f
that the extra deadline was mainly used by the system to accommodate more jobs by
increase waiting times. Moreover, we observe that the waiting time of the 4CIF jobs
was reduced to enforce the new deadline. Therefore, we can conclude that the system
is able to adaptively change job allocation to not only enforce the required QoS, but
also to maximize the amount of jobs processed in the system.

![Graphs showing acceptance ratio, number of jobs accepted, and completion time for QCIF, CIF, 4CIF, LV, and HV scenarios.](a) Real  (b) Model

Figure 3.9: Summary of model validation results for 12-42-150 deadlines. ©2017 IEEE
(reprinted with permission) Zamani et al. [1]. HV and LV correspond to high value and
low value videos processing jobs, respectively – as defined in Equation 3.10. Each
column represents a set of experiments, where Real means experimentally obtained and
Model means analytically obtained.

Since the execution time and data transfer times were similar to those in the previous
Figure 3.10: Summary of model validation results for 48-48-120 deadlines. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10. Each column represents a set of experiments, where Real means experimentally obtained and Model means analytically obtained.

3.7.3 Experiment 3. Validation of the Model

In this Section, we validate our mathematical model by comparing the experimentally obtained results in Sections 3.7.1 and 3.7.2 against those obtained analytically using the model proposed in Section 3.4 and 3.5. This model has been implemented using a custom made Python simulator. The simulator gets the network topology, available
resources, jobs and amount of computations, and simulates the model. Figures 3.9 and 3.10 collects the results of each experiment organized by columns.

Columns (a) and (b) of Figure 3.9 compare the results of the real experiment and the model for the experiment described in Section 3.7.1, respectively. Top and middle rows show the acceptance ratio per type of job and per value. We observe that there was no difference regarding to the number and type of jobs accepted. The bottom row shows the completion time of the jobs. In this case, we observe small differences between the model and the real experiment. The biggest differences were found in the smallest type of jobs (QCIF), where we had up to a 46% difference in the average completion time in the scenario including edge computing (E+I+C) – from 7.8 seconds in the real experiment to 4.2 seconds in the model. However, the other type of jobs showed less than 10% of difference. These differences, more noticeable in small jobs, were due to unaccounted overheads of the real infrastructure. The real experiment was executed in Amazon EC2, where the network performance was not guaranteed. Moreover, the orchestration and decision making operations could have also affected the completion times – e.g., interacting with the controller, deciding how to schedule the workload. These decision making overheads were hard to estimate to include in the model.

Columns (a) and (b) of Figure 3.10 compare the results of the real experiment and the model for the experiment described in Section 3.7.2, respectively. In this case, we observe in Figure 3.10a and 3.10b middle row a small difference in the number of accepted jobs. In particular in the real experiment two low value jobs were rejected in the E+I+C scenario, while the model considered that those jobs should have been accepted. Regarding the completion time, we observe also some differences in the type QCIF and scenarios named I+C and E+I+C. In this case the difference is up to a 33% for the QCIF jobs, and less than 10% difference for the rest of the cases.

We observed that the results obtained with the model were within a small error margin of the experimental results. Therefore, we consider to be proven that the model can reliably represent our experimental environment. Next, we continue the evaluation of our approach using our model, which allows us to easily introduce variations to the execution environment (i.e. infrastructure).
Figure 3.11: Summary of experimental results – Modifying Bandwidth. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10.

3.7.4 Experiment 4. Analytical Evaluation

In this Section, we performed an analytical evaluation of our use case using our model. We changed different parameters of the infrastructure to observe their effect in the
workload. We used the deadlines set in our first experiment (Section 3.7.1), as they had more room for improvement. The deadlines were QCIF = 12s, CIF = 42s, 4CIF = 150s. These experiments were analytically simulated using our mathematical model described in Section 3.4 and 3.5, and validated in Section 3.7.3.

Previously, we observed that the network had a strong influence on the completion time of our jobs. Hence, we first analyzed how the number of accepted jobs and their completion times were affected by changes in the network bandwidth. Figure 3.11 collects the results – each column represents a different experiment. Column (a) of Figure 3.11 was our baseline scenario with a bandwidth of 20MB, which shows the results of the real experiments performed in Section 3.7.1. Columns (b), (c), and (d) of Figure 3.11 show results for an increasing bandwidth of network links, that is, 30MB, 40MB, and 60MB, respectively.

We can clearly observe how the bandwidth positively affected the acceptance rate and the completion time. We can observe in the bottom row of Figure 3.11, that in all cases the completion time of jobs was significantly reduced, between 20% to 50%, when increasing the network bandwidth. This was the expected behaviour as our use case was highly data-intensive. We can also observe in the top row of Figure 3.11 that by reducing the completion time, the acceptance ratio increased. The scenario that most benefited from this increase was the Cloud deployment (labeled as C), as it is the one that required a larger amount of data to be transferred from source to the core of the infrastructure. Although we observed that from 20MB to 30MB we had a strong increase in the number of accepted jobs, between a 40% and 64%, this tendency slowed down in subsequent increases in bandwidth. It required increasing the network bandwidth by 200% to achieve 100% of job acceptance in the Cloud scenario. On the contrary, the other approaches also benefited from bandwidth increases and quickly reached 100% completion ratio for high value jobs. In the edge computing scenario (labeled as E+I+C), the system was able to accept all jobs, including low value ones, starting with 30MB network links. We can conclude that the proposed approach showed a lower dependency on the performance of the network (although the network capacity will influence the number of jobs submitted to the in-transit and cloud resources),
primarily due to better use of the resources closer to the edge of the infrastructure, which allowed a lower use of the network links close to the core.

### 3.7.4.1 Number of Workers

In this section, we evaluated the effect that increasing the number of workers at different layers of the infrastructure has on the acceptance ratio of jobs. We also used different bandwidths to understand the relationship between the bandwidth and the number of workers. Table 3.3 collects the number of accepted jobs for each experiment. We use the first column named BW (bandwidth) to group experiments based on the network bandwidth used (i.e., 20, 30, and 40 MB). Within each network bandwidth group, we have experiments for the different scenarios considered in this chapter, namely cloud (C), in-transit plus cloud (I+C), edge plus in-transit plus cloud (E+I+C). The four most right columns identify the number of workers used by each experiment. We have the Baseline column, which used the number of workers described in Table 3.1. Next we have the $2x\ C$ column, which doubled the number of workers in the cloud site; the $2x\ I$ column, which doubled the number of workers at each In-transit site; and the $2x\ C\&I$ column, which doubled the number of workers in both, the cloud and the in-transit sites.

Looking at the Table 3.3 from left to right, we can observe that changing the number of workers did not affect the acceptance ratio for any of the performed experiments. The main reason is that there was a small number of jobs waiting idle to be computed, as described in Figure 3.7f. Thus, the only changes observed in Table 3.3 were due to changes in the network link bandwidth – the effect of the bandwidth in the acceptance of jobs was studied in the previous section. Therefore, we can conclude that for data intensive applications, increasing the number of workers may not affect the number of accepted jobs.

### 3.7.4.2 Performance

Lastly, we studied the effect that the performance of the workers has on the acceptance ratio. We also used different bandwidths to make sure that the bandwidth did
Table 3.3: Number of accepted jobs – Modifying Number of Workers. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 2x C means we doubled the number of workers in the cloud site; 2x I means we doubled the number of workers in the in-transit sites; and 2x C&I means we doubled the number of workers in both, the cloud and the in-transit sites. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10.

<table>
<thead>
<tr>
<th>BW</th>
<th>Scenario</th>
<th>Value</th>
<th>Baseline</th>
<th>2x C</th>
<th>2x I</th>
<th>2x C&amp;I</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 MB</td>
<td>C</td>
<td>LV</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>I+C</td>
<td>LV</td>
<td>82</td>
<td>82</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>E+I+C</td>
<td>LV</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td>30 MB</td>
<td>C</td>
<td>LV</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>I+C</td>
<td>LV</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>E+I+C</td>
<td>LV</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td>40 MB</td>
<td>C</td>
<td>LV</td>
<td>83</td>
<td>83</td>
<td>83</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>I+C</td>
<td>LV</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>E+I+C</td>
<td>LV</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
</tbody>
</table>
not impact the results. Table 3.4 collects the number of accepted jobs for each experiment. We use the first column named BW (bandwidth) to group experiments based on the network bandwidth used (i.e. 20, 30, and 40 MB). Within each network bandwidth group, we have experiments for the different scenarios considered in this chapter, namely cloud (C), in-transit plus cloud (I+C), edge plus in-transit plus cloud (E+I+C). The four most right columns identify the number of workers used by each experiment.

We have the Baseline column, where the performance of workers was as described in Table 3.1; in the 2x C column, we doubled the performance of the workers located at the cloud site; in the 2x I column, we doubled the performance of the workers located at each In-transit site; and in the 2x C&I column, we doubled the performance of the workers located at both Cloud site and In-transit sites.

Looking at the Table 3.4 from left to right, we observe limited changes when modifying the performance of the workers, we marked those cases where the number of accepted jobs was different from the baseline. Table 3.4 shows that the performance of the workers had a limited effect in the number of accepted jobs for the Cloud scenario (labeled as C). For this scenario we observed that when the bandwidth was 20MB, the system was able to accept one additional low value (LV) job, although it rejected a high value (HV) job – in this scenario the scheduler did not differentiate between LV and HV jobs; for a bandwidth of 30MB, we observed an improvement of 8% for the HV jobs when increasing the performance; and for a bandwidth of 40MB, we observed no improvement at all. On the other hand, for the I+C (In-transit plus Cloud) and E+I+C (Edge plus In-transit plus Cloud), we observed no improvement in any of the cases. On the contrary, we observed that when increasing the performance of the Cloud workers, columns 2x C and 2x C&I, the number of accepted low value (LV) jobs decreased. The reason for this was that the scheduler, that used to prioritize allocating high value jobs among In-Transit resources to minimize completion time, decided to move some of the high value workload towards the Cloud due to its increased performance. This decision affected low value jobs as our scheduling policy tried to minimize the cost of low value jobs by allocating them in the Cloud. In general, we can conclude that, when the bandwidth limits the workload we can accept, increasing the performance of the resources
Table 3.4: Number of accepted jobs – Modifying Performance of Workers. ©2017 IEEE (reprinted with permission) Zamani et al. [1]. 2x C means we doubled the performance of workers in the cloud site; 2x I means we doubled the performance of workers in the in-transit sites; and 2x C&I means we doubled the performance of workers in both, the cloud and the in-transit sites. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 3.10.

<table>
<thead>
<tr>
<th>BW</th>
<th>Scenario</th>
<th>Value</th>
<th>Baseline</th>
<th>2x C</th>
<th>2x I</th>
<th>2x C&amp;I</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 MB</td>
<td>C</td>
<td>LV</td>
<td>26</td>
<td>27</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>42</td>
<td>41</td>
<td>42</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>I+C</td>
<td>LV</td>
<td>82</td>
<td>49</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>E+I+C</td>
<td>LV</td>
<td>102</td>
<td>91</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td>30 MB</td>
<td>C</td>
<td>LV</td>
<td>75</td>
<td>71</td>
<td>75</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>65</td>
<td>71</td>
<td>65</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>I+C</td>
<td>LV</td>
<td>102</td>
<td>82</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>E+I+C</td>
<td>LV</td>
<td>118</td>
<td>106</td>
<td>118</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td>40 MB</td>
<td>C</td>
<td>LV</td>
<td>83</td>
<td>83</td>
<td>83</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>I+C</td>
<td>LV</td>
<td>102</td>
<td>83</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>E+I+C</td>
<td>LV</td>
<td>118</td>
<td>103</td>
<td>118</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HV</td>
<td>116</td>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
</tbody>
</table>
in the federation have a limited effect in the ability of the system for increasing the number of accepted jobs.

3.8 Discussion

The results presented in this chapter show the limitations that traditional approaches, consisting on transferring all data from its source (data capture site) to the core of the infrastructure, face when applied to data-intensive applications with time constraints (such as video surveillance, smart Grid, and other IoT scenarios). In our experiments, we show how the network links can quickly become a bottleneck that slow down workload processing. This can lead to a number of jobs being rejected, as processing times will not be able to meet required QoS constraints (e.g., deadline).

We observed how our approach was able to overcome the limitations of a traditional approach by leveraging computational resources at the edge of the infrastructure (camera aggregators) and within the network data centers, leveraged and enabled by SDN technology. The results show that by using our in-network computational model and the proposed scheduling strategy, the system was able to accept up to a 70% more workload. In our approach we used edge computing to perform a systematic sampling of the data, which in practice reduced computational requirements without affecting the key information derived from its content. Other applications could use edge computing to filter out invalid or out-of-range parameters, or perform similar operations that can help in using limited resources more effectively and increase the obtained value of the data. Nevertheless, the results show that even in cases where edge computing was not possible, leveraging SDN and NFV technologies to perform in-transit computation within the network data centers had a significant impact on the size of workload processed. In particular, we observed up to a 60% improvement in the job acceptance ratio.

We also discuss additional scenarios using our mathematical model to modify different parameters of the infrastructure that could affect the size of workload that the system was able to process. These experiments confirmed that the network was the
single main factor limiting the amount of workload that the system was able to compute given certain deadline constraints. We observed that increasing the number of workers had no effect on the number of jobs accepted and increasing the performance of the workers had a very limited effect, less than 10%. Horizontal and vertical scaling of machines within a data center is certainly significant for data intensive applications, however, distributed data-intensive applications can benefit significantly from geographically distributed in-transit (computational) resources, as proposed in this work.

3.9 Relevant Publications

This chapter contains portions adapted from the following published papers with permissions from the copyright holder.


Chapter 4

Edge and In-transit Enabled Approximate Computing

4.1 Introduction

There is a significant class of data intensive applications (especially those involving data generated from IoT devices) that have associated analysis deadlines with direct relationships to a quality of results metric. The incoming workloads for these applications can include a number of sources, such as various types of sensors (i.e., potentially inexact inputs), and the associated data analysis algorithms are often stochastic in nature (e.g., iterative algorithms). For example, in the building energy optimization scenario being considered in this work, simulations are iterative and have different numerical configuration parameters that directly influence the duration of simulation, as well as the quality of results. Therefore, it becomes important to develop a suitable computational infrastructure wherein both hardware and software are continually optimized, in-line with application specific objectives.¹

One of the main topics that we study in this chapter is approximate computing. Approximate computing provides a useful mechanism for supporting the requirements identified above, and has attracted significant interest from both academia and industry. It enables techniques for creating robust and resilient applications by proposing the introduction of intervals of acceptable errors into the execution to address both computing and application uncertainties. Approximate computing views regions in the application’s execution in terms of the degree of error tolerance, and uses this tolerance to trade off between storage, result accuracy, and efficient usage of computing resources.

¹The results presented in this chapter have been published in the Proceedings of IEEE 5th International Conference on Future Internet of Things and Cloud [3].
(i.e., energy, storage size, etc.). Thus, approximate computing provides a balance between the level of accuracy required by the user and that provided by the computing system, to achieve a spectrum of optimizations. Such techniques are applicable to a wide range of applications/frameworks, for example, data analytics, scientific computing, multimedia and signal processing, machine learning and MapReduce [47, 136].

In Chapter 3, we developed an analytical model to improve the use of computational resources at the network edge and within network data centers to support data transformation and analysis from source to destination. The objective of Chapter 3 was to combine efficient use of resources within a cloud data center and those at the network edge, while providing an extra source of revenue for those who operate and manage such resources. In this chapter, we extend this approach by investigating how network edge resources can be used to support a data analysis workflow deployed leveraging approximation techniques. We seek to find ways to optimize workflow execution over resources that are located at the network edge and to determine which approximation techniques are most beneficial given resource constraints at the edge. A key assumption in this chapter is that the core data center are far away from data source and have a greater capability (capacity, function) compared to those at the network edge.

Here, we investigate two uses of approximate computing: (i) The ability to reduce computation using approximate computing techniques. (ii) The ability to execute parts of the workload or increase the accuracy of approximation techniques using resources at in-transit nodes or at the edge of the network.

Our application workflow performs real-time energy optimization within a building, and makes use of multiple approximation techniques as part of the simulation(s). In this work, we determine which of these techniques can be executed in-transit or at the edge of the network in the presence of multiple constraints such as execution time and quality of results. The rest of the chapter is organized as follows. In Section 4.2 we present related work in approximate computing. In Section 4.3 we explain the application use-case as identified in building optimization, followed by the associated approximation techniques presented in Section 4.4. The methodology is presented in Section 4.5. We present our evaluation in Section 4.6 and discuss about the outcome.
of our proposal in Section 4.7.

4.2 Approximation techniques

In this section, we summarize several approximate computing techniques that can be applied at the network edge.

- Memory access skipping, task dropping/skipping, Memoization: Samadi et al. [137] propose a pattern-based approximation technique to reduce number of memory accesses by skipping tasks in a loop, apply memoization to optimize map and scatter/gather patterns and cache result of computationally expensive function calls, to reduce computational overhead. Samadi et al. [138] propose SAGE, a self-tuning approximation for graphics engines which uses data packing to reduce access to memory. Goiri et al. [139] present an ApproxHadoop module which can apply approximations to MapReduce frameworks using data sampling and task dropping.

- Using multiple inexact program versions/lossy compression: Vassiliadis et al. [140] propose a programming model and runtime system to improve energy efficiency of the programs. The programming model enables developers to identify the impact of different sections of their program on the final output.

- Neural network-based accelerators: McAffee and Olukotun [141] developed EMEURO which is a neural-network emulation and acceleration platform. With small approximation error rate, EMEURO can achieve considerable speedup in various applications within the image processing domain. Amant et al. [142] developed a general-purpose code acceleration and end-to-end solution to utilize analog circuits in order to accelerate approximate applications and neural network training phase.

- Approximating neural networks: Venkataramani et al. [143] have explored the use of approximate computing to design a new energy efficient hardware implementation for large scale neuromorphic systems(AxNN). Zhang et al. [144] propose
an approximate computing framework for ANN. It is based on approximating neurons which are less critical.

- **Precision Scaling**: Anam et al. [145] explore the trade-off in precision for energy and throughput in a generic matrix multiplication and one dimensional convolution. Yeh et al. [146] apply dynamic precision tuning in floating point computation. This techniques can increase the performance and decrease energy consumption in physics-based animation.

- **Loop Perforation**: Baek and Chilimbi [147] develop a framework for supporting energy-conscious programming using controlled approximation formed of 2 phases – “calibration” phase, which involved building a model for Quality of Service (QoS) loss, and an “operational” phase which involves directly applying the approximation decisions.

### 4.3 Energyplus Use Case

An instrumented built environment, which can consist of single/multiple buildings (homes, office buildings, sports facilities, etc), provides a useful scenario to validate the use of edge-supported approximation. Depending on the number of sensors within a single building, the frequency at which data is captured from such sensors and the particular data analysis objective (e.g. reduce energy consumption, improve efficiency of HVAC (heating, ventilation and air condition) function, improve comfort levels based on occupancy, etc), the computational capability requirements can vary significantly. In some instances such data is often analyzed off-line (in batch mode) to enable improvements in building design or to support long term facilities management. In other instances (evidenced by recent use of such instrumented environments), real time analysis needs to be carried out (over intervals of 15 to 30 minutes generally) to enable better energy efficiency and use of such infrastructure. When multiple such buildings are considered (e.g. within a business park, University campus or a housing association), the overall computational requirement can increase considerably [132].

To provide practical real time decision making in building energy management based
on real time monitored data, it is necessary to develop a ‘behaviour’ of a building energy system by using various simulation tools. During the process, domain experts are often involved in order to identify the main use cases and scenarios with associated input parameters and feasible outputs. In the modelling process, a number of components have to be assessed and calibrated iteratively, and the developed building energy simulation model is then executed (as the calculation engine) within a generic optimization program. In this work we seek to identify approximation techniques that can complement or replace the execution of multiple EnergyPlus\textsuperscript{2} instances, a software that requires significant computational resources to run, with different input parameter ranges [148].

Various types of sensors are used to monitor energy efficiency levels within a building, such as: (i) solid-state meters for accurate usage levels, (ii) environmental sensors for measuring temperature, relative humidity (RH), carbon monoxide (CO), and carbon dioxide (CO\textsubscript{2}) levels, (iii) temperature measurements using both mechanical (e.g., thermally expanding metallic coils) and electrical means (e.g., thermistors, metallic resistance temperature detectors (RTD), thermocouples, digital P-n junctions, infrared thermocouples) to provide sufficient accuracy. When dealing with large buildings such as sports facilities, the accuracy of these sensors is often questionable, largely because of the significant drift that occurs after initial calibration. In some buildings, there are specific requirements for sensors when monitoring CO\textsubscript{2} concentration, air flow, humidity, etc and these sensors are more expensive to use and deploy. We use sensor data from the SportE\textsuperscript{2} project pilot called FIDIA\textsuperscript{3}, a public sports building facility, located in Rome, Italy. SportE\textsuperscript{2} is a research project co-financed by the European Commission FP7 programme under the domain of Information Communication Technologies and Energy Efficient Buildings. This project focuses on developing energy efficient products and services dedicated to needs and unique characteristics of sporting facilities.

\textsuperscript{2}http://apps1.eere.energy.gov/buildings/energyplus/
\textsuperscript{3}http://www.asfidia.it
4.4 EnergyPlus approximation techniques

We identify the following approximation techniques applicable to EnergyPlus:

1. EnergyPlus loop reduction: This technique involves reducing the number of EnergyPlus instances by reducing the number loop iterations used within this simulation. As EnergyPlus execution needs to be carried out over a particular time frame, we can reduce the number of iterations/loop counter used, leading to a reduction in time over which EnergyPlus executions are carried out. This reduces the overall execution time while keeping the quality of results within a pre-defined error interval. The error rate is: \( err = \frac{1}{times} \), where \( times \geq 1 \) is the number of times/loops to repeat the simulation.

2. Use of Artificial Neural Network: Such a method involves the use of a learning algorithm for replacing the EnergyPlus simulation altogether. A neural network is trained based on historical (input/output) data obtained from previous executions of EnergyPlus simulations. This data is then used to train a neural network as a function approximator for the behaviour of an EnergyPlus simulation. The corresponding error rate is based on the size of historical data and on the efficiency of the neural learning algorithm being used. We set the error rate to 0.05, so
that additional EnergyPlus simulations can be triggered if the error exceeds this threshold.

3. Parameter value skipping: Based on a set of parameters that the simulation requires, this method reduces the number of the parameter values which are used as input to the EnergyPlus simulation. The corresponding error rate of this method is based on the skipping interval. The associated error rate is \( \frac{k}{100} \), where \( k \) is the number of parameter values skipped.

- Execution time without approximation techniques: \( total.time = n*m*time \), where \( time \) is the time of one EnergyPlus simulation
- Execution time with approximation techniques \( total.approx.time = (n*m*total.time) - (k*total.time) \), \( n \) represents total number of parameters values, \( m \) is the number of parameters, and \( k \) represents the number of parameter values skipped;
- \( error rate = \frac{k}{total.param} \), where \( total.param \) is the total number of parameter values.

4. Parameter interval reduction: From the interval associated with a parameter we reduce the interval limits so the simulation would use only values from a predefined average value/centrality associated with a parameter interval. The error of this method depends on the remaining number of parameter values to use as input in the simulation. The error rate is: \( =\frac{n+k}{100} \), where \( n \) is the number of the total parameters and \( k \) is the number of intervals being used.

- Execution time without approximation techniques: \( total.time = n*m*time \);
- Execution time with approximation techniques \( total.approx.time = (n*m*total.time) - ((n-k)*total.time) \), \( n \) represents total number of parameters values, \( m \) is the number of parameters, and \( k \) represents the number of parameter intervals reduced;
- \( error rate = \frac{n+k}{total.params+total.intervals} \), where \( total.params \) is total number of parameters and \( total.intervals \) is total number of intervals.
4.5 Methodology and Approximate In-transit Computational Model

To use network infrastructure more effectively, we propose hosting a data processing service at the network edge (on a gateway node connected to IoT devices) or within a network (making use of programmable network-based approaches, e.g. OpenFlow and other Software Defined Networks (SDN) approaches) to offer idle/available computational capabilities at the edge/in-transit data centers. Furthermore, this service is able to provision computational resources and allocate workload into such resources. Since the computational capabilities of the network data centers and edge clouds are limited, the main purpose of the resources at the edge and in-transit is to carry out small parts of the workload and increase the accuracy of the approximation techniques at the edge of the network by utilizing unused capacity within network data centers.

In general, two types of resources are considered in this work. The main source of computation is computational data centers (sites or resource providers), which collectively form the cloud federation established using CometCloud [102]. The secondary resources are network data centers, that are located at the edge of the network. Lets assume a client needs to compute a job $J$, composed of $k$ tasks, which is generated at the client’s location defined as source $s$. Whenever the client decides to outsource the job to be executed at a remote site defined as destination $d$, the data would be exposed to possible edge and in-transit resources. A set of $q$ network data centers $R : \{r_1, ..., rq\}$ has been considered as potential resources at the edge of the network. Hence, it is essential to identify the workload placement, the best route from source to destination, and possible edge/in-transit resources that can contribute to the execution of the job. The service level agreement (SLA) of a job $J$ includes: deadline ($Deadline(J)$) by which results have to be returned to the client and a budget ($Budget(J)$) that sets the maximum amount available to spend on computing job $J$.

To ensure the control over the network, all of the sites and network data centers are equipped with SDN enabled routers. We consider that there is some waiting time $W(J)$ before a job $J$ can be executed at destination site $d$. During this time, the job is idle and it occupies storage space at the destination site. Hence, we would like to identify
and configure a data path that leverages edge/in-transit computation to take advantage of $W(J)$ for a job. The following variables have been considered in our formulations:

- $P(r_i)$ is the average number of tasks that resource $r_i$ completes per unit of time.
- $E(r_i)$ is the amount of time spent computing in resource $r_i$.
- $CE(r_i)$ is the cost per unit of time of using resource $r_i$ for computation.
- $T(r_i, r_k)$ is the amount of time spent transferring data between resources $r_i$ and $r_k$.
- $CT(r_i, r_k)$ is the cost of reserving a network channel per unit of time.
- $W(J)$ is the waiting time before job $J$ can start its computation at destination resource.

The objective of our problem is to maximize the number of tasks completed at edge/in-transit resources:

$$\max \sum_i P(r_i) \cdot E(r_i)$$

subject to being ready to be computed at destination resource $d$ at the scheduled time (4.1) and making sure that the all of tasks within a job is executed completely(4.2), within the given deadline(4.3) and budget(4.4):

$$\sum_i E(r_i) + \text{Transfer}(J) \leq W(J), \quad (4.1)$$

$$\sum_i [P(r_i) \cdot E(r_i)] + P(d) \cdot E(d) = k, \quad (4.2)$$

$$\sum_i E(r_i) + \text{Transfer}(J) + E(d) \leq \text{Deadline}, \quad (4.3)$$

$$\text{Cost}(J) \leq \text{Budget}, \quad (4.4)$$

where $\text{Transfer}(J)$ is the overall transfer time of a job, defined as the sum of the time spent transferring data from source $(s)$ to first network data center $(r_i)$, the sum of the time spent transferring data between network data centers $\in R$, and the time spent
transferring data between the last network data center \((r_k)\) and destination \((d)\):

\[
Transfer(J) = T(s, r_i) + \sum_{i}^{q} \sum_{k \neq i,k} T(r_i, r_k) + T(r_k, d).
\]

\(Cost(J)\) is the overall cost of computing job \(J\), defined as:

\[
Cost(J) = CostExecMid + CostExecDest + CostNet,
\]

where the cost of computing in-transit \((CostExecMid)\) and computing at the destination resource \(d\) \((CostExecDest)\) are defined as:

\[
CostExecMid = \sum_{i} [CE(r_i) * E(r_i)],
\]

\[
CostExecDest = CE(d) * E(d),
\]

and the cost of transferring data associated with a job \((CostNet)\) is defined as:

\[
CostNet = T(s, r_i) * CT(s, r_i) + \sum_{i}^{q} \sum_{k \neq i,k} [T(r_i, r_k) * CT(r_i, r_k)] + T(r_k, d) * CT(r_k, d)
\]

subject to \(E(r_k) \neq 0\). Note that the time and cost of returning results to the client is negligible as only a few parameters are sent.

When resource providers cannot execute a job with the required SLA, the client can consider the use of approximation techniques. We consider that for an application \(app\), we have a set of applicable approximation techniques \(T = \{t_1, t_2, ..., t_m\}\), where each \(app\) has an associated required quality of solution and can be approximated with a subset of techniques \(Q_k = \{t_1, t_2, ..., t_k\}\), \(Q_k \subset T\), \(m \geq k\), provided these techniques can satisfy a minimum accuracy threshold. Our two main objectives are to determine:

1. set of approximation techniques \(T\) that are applicable to an application \(app\)
2. \(t_j \in T\) that produces results within accuracy threshold, and techniques that can be executed on edge/in-transit resource(s)(i.e. create \(Q_K\))

Given a set of suitable approximation techniques, \(Q_k = \{t_1, t_2, ..., t_k\}\), each \(t_j\) has a corresponding error rate. Our strategy is to pick the approximation technique that has
highest accuracy. If two approximation techniques have the same accuracy level, we pick the one with less computational resource requirement. If we pick an approximation technique, the edge/in-transit resource will use the waiting time \((W(J))\) to increase the accuracy of the chosen technique. For example, if a loop reduction approximation techniques is chosen, we try to increase the number of loops with resources at the edge of the network or in-transit nodes to increase the accuracy of the chosen approximation technique.

4.6 Evaluation

In this section, we present the overall setup for our experimental infrastructure and several scenarios that we deployed to validate our hypothesis.

4.6.1 Configuration of Testbed

Our federation has been deployed on CloudLab infrastructure platform. 8 VM instances are emulating geographically distributed environment to develop an evaluation testbed. Figure 4.2 shows the overview of the implemented infrastructure. 3VMs served as our resource providers’ data centers: Site1, Site2 and Site3. Furthermore, we dedicated 5 more VMs as in-transit and edge resources(Edge Clouds) which are located at the edge of the network (i.e. between main sites). To emulate geographic distribution of the resources, we use Hierarchy Token Bucket (HTB) to configure various network bandwidth parameters. This network configuration is inspired in data obtained from previous experiments [148, 149]. Based on the computational capabilities at the edge/in-transit and resource provider sites, we considered three different infrastructure scenarios. The details of each scenario has been shown in Table 4.1. To model the mentioned scenarios, we used the characteristics of Amazon EC2 VM instances in our model. Summary of resource characteristics have been shown in Table 4.2.

To deploy the instances and create the network between them, we use Mininet. In

\(^4\)CloudLab: https://www.cloudlab.us

\(^5\)mininet: http://mininet.org
Figure 4.2: Infrastructure Setup. Solid and dashed lines indicate high and low bandwidth links, respectively. ©2017 IEEE (reprinted with permission) Zamani et al. [3].

Table 4.1: Infrastructure Scenarios. ©2017 IEEE (reprinted with permission) Zamani et al. [3].

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Edge Resources</th>
<th>Site Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>c4.2xlarge</td>
<td>c4.2xlarge</td>
</tr>
<tr>
<td>Higher</td>
<td>c4.2xlarge</td>
<td>c4.4xlarge</td>
</tr>
<tr>
<td>Highest</td>
<td>c4.2xlarge</td>
<td>c4.8xlarge</td>
</tr>
</tbody>
</table>

Table 4.2: Resource Properties. ©2017 IEEE (reprinted with permission) Zamani et al. [3].

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>vCPU</th>
<th>ECU</th>
<th>Memory</th>
<th>Price ($/Hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>31</td>
<td>15</td>
<td>0.464</td>
</tr>
<tr>
<td>c4.4xlarge</td>
<td>16</td>
<td>62</td>
<td>30</td>
<td>0.928</td>
</tr>
<tr>
<td>c4.8xlarge</td>
<td>36</td>
<td>132</td>
<td>60</td>
<td>1.856</td>
</tr>
</tbody>
</table>

each VM, one Mininet host is connected to a virtual switch deployed by Open vSwich\(^6\). These virtual switches in different instance are connected to each other using Generic

---

\(^6\)Open vSwich: http://openvswitch.org
Routing Encapsulation (GRE) tunneling [135]. Aside from the computational resources, a POX SDN controller \(^7\) has been deployed to control the network infrastructure. In this work, the SDN controller has 2 main jobs: (i) installing the forwarding rules on switches. (ii) dedicating ports and connections to switches in order to receive necessary information from sites and optimizing the job execution. The controller establishes a dedicated connection between hosts and the controller. Here, we use UDP packets for communication between hosts and controller. Hence, the controller configures the switches to forward the UDP packet to the controller, unless those packets are generated from the controller. Specifically, when the source site, i.e. client, wants to talk to the controller, it should create a UDP message and send it to a specific address. The first switch receiving that message forwards the packet to the controller. Next, the controller sends UDP packets to all edge and in-transit sites asking for their status and computational capabilities. Since these UDP messages are from the controller, switches know where to send (i.e. in-transit resources) and deliver them flawlessly. Then, based on the feedback from the edge and in-transit resources, the controller selects the data transfer and execution pattern. This decision is sent to the client.

4.6.2 Experiments

For the use case, we considered that there are multiple smart buildings requesting to evaluate and optimize their energy consumption. Considering the FIDIA pilot \(^8\), three types of jobs have been considered in this work. Table 4.3 collects the characteristics of different job types. The budget has been chosen high enough for jobs to eliminate imposed cost limitation on the system. As a result, the deadline is the only limiting factor in our experiments. We have considered 4 strategies:

- **Traditional Approach(T):** In this strategy, only resource providers’ sites can perform computation.

---

\(^7\)POX: https://openflow.stanford.edu/display/ONL/POX+Wiki

\(^8\)FIDIA: http://www.asfidia.it
• **Edge Approach (E):** Using this strategy, the edge data center and in-transit resources can contribute to the execution of the EnergyPlus jobs.

• **Approximation Approach (A):** For this strategy, we considered that only resource providers' sites can execute jobs. However, if the system cannot meet the SLA requirement, using different approximation techniques will be considered.

• **Edge plus Approximation Approach (E-A):** In this approach, the edge and in-transit resources can contribute to the computations and also increase the accuracy of approximation strategy.

For all of the scenarios mentioned in Table 4.1, we conducted the experiments based on the different strategies mentioned above. In each experiment, 326 jobs were inserted from Site1 to a federated marketplace, generated using a Poisson distribution. Once a job was inserted in the federation, different sites (i.e. Site1, Site2, and Site3) offered their services using a blind auction mechanism. For all of the jobs, if approximation techniques were available (E and E-A approaches), we established the minimum accuracy requirement of 95%. Consequently, if there was not enough resources to execute the job within the deadline, the system executed the job using one of the available approximation techniques. The approximation techniques selection has been discussed in Section 4.5. Moreover, for the ANN approximation method, we considered a training period that prepares the ANN method for desired accuracy. We considered 100 complete EnergyPlus executions for ANN training phase. In other words, if we could accept 100 EnergyPlus jobs (excluding other approximation methods), ANN model would be ready and system could use ANN method afterwards.

Table 4.4 shows the required completion time for all types of EnergyPlus jobs in different infrastructures. The execution time for the different approximation techniques has been explained in Section 4.4 and is based on the parameters selected for approximation techniques, e.g. number of loops, amount of parameter value skipped, etc.

In order to compare different scenarios and evaluate the impact of edge clouds and approximation techniques, job acceptance ratio is demonstrated in Figure 4.3. The acceptance ratio is the ability of the resources to execute jobs completely within the
Table 4.3: Job Information. ©2017 IEEE (reprinted with permission) Zamani et al. [3].

<table>
<thead>
<tr>
<th>JobType</th>
<th>Data Size(MB)</th>
<th>Budget</th>
<th>Deadline(s)</th>
<th>Tasks^†</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobType1</td>
<td>10</td>
<td>20</td>
<td>120</td>
<td>10</td>
</tr>
<tr>
<td>JobType2</td>
<td>20</td>
<td>30</td>
<td>150</td>
<td>20</td>
</tr>
<tr>
<td>JobType3</td>
<td>30</td>
<td>40</td>
<td>180</td>
<td>30</td>
</tr>
</tbody>
</table>

^† – A job is composed of a set of tasks

Table 4.4: Time to completion of EnergyPlus job types. ©2017 IEEE (reprinted with permission) Zamani et al. [3].

<table>
<thead>
<tr>
<th>JobType</th>
<th>c4.2xlarge</th>
<th>c4.4xlarge</th>
<th>c4.8xlarge</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobType1</td>
<td>80 s</td>
<td>40 s</td>
<td>20 s</td>
</tr>
<tr>
<td>JobType2</td>
<td>100 s</td>
<td>50 s</td>
<td>25 s</td>
</tr>
<tr>
<td>JobType3</td>
<td>120 s</td>
<td>60 s</td>
<td>30 s</td>
</tr>
</tbody>
</table>

deadline. Various approximation techniques help to reduce the potential resource requirement and overcome the limitations imposed by lack of resources. Looking at Figure 4.3, addition of edge resources and approximation techniques increase the job acceptance ratio. In the base scenario, ANN method has not been used due to the small number of accepted jobs. Therefore, ANN method training has not reached the desired accuracy threshold. Moreover, in the base scenario, since the computing power of the site resources is limited, addition of the edge and approximation techniques slightly increase the acceptance ratio(2% to 5%). In the higher scenario, the best result has been reached in E-A approach, where sufficient E+ (EnergyPlus) simulations are executed to pass the threshold of accuracy of ANN, so ANN has a great impact on the job acceptance ratio. The edge strategy(E) has 22% more acceptance ratio compared to approximation(A) strategy, which shows the importance of the edge clouds in the case where the site computing resources are limited. In the highest scenario, E-A strategy has reached 100% acceptance ratio from which ANN covers around 2% of the jobs.
For the approximation(A) strategy, in all scenarios, if ANN is not available, parameter interval reduction has the most contribution among all approximation techniques followed by parameter value skipping and loop reduction techniques.

Figure 4.3: Job Acceptance Ratio. ©2017 IEEE (reprinted with permission) Zamani et al. [3]. E+: EnergyPlus, ANN: Artificial Neural Network, Loop: Loop Reduction, Interval: Parameter Interval Reduction, Parameter: Parameter Value Skipping

Figure 4.4 compares the average accuracy achieved in different scenario and strategies. As we mentioned earlier, the minimum required accuracy in case of approximation is 95%. Since approximation is not available for traditional(T) and edge(E) strategies, the accuracy for those cases are 100%. However, for approximation(A) and edge-approximation techniques(E-A), the maximum accuracy among available techniques are selected. Moreover, addition of the edge to approximation strategy produces more accurate results. This behavior is due to the fact that the system uses edge clouds to execute part of the job (tasks within the job) or possibly increases the approximation accuracy. Considering both Figures 4.3 and 4.4, we conclude that approximation techniques have large impact on the acceptance ratio with slightly less accuracy. Addition
of the edge resources results in more jobs to be executed with higher accuracy.

![Average Accuracy Graph](image)

Figure 4.4: Average Accuracy. ©2017 IEEE (reprinted with permission) Zamani et al. [3].

Figure 4.5 collects the information regarding the job waiting time (queue time). Having edge clouds can cause less waiting time because edge resources use waiting time to increase the accuracy of the approximation techniques or execute part of the jobs.

The average cost of the jobs has been shown in Figure 4.6. In general, approximation techniques (A and E-A strategies) result in cheaper job completion due to the reduction in job execution time. Specially, ANN techniques is significantly cost beneficial due to small execution time needed for ANN. However, we should note that approximation techniques are not as accurate as regular E+ execution.

4.7 Discussion

In this chapter, we explore the advantages of approximation techniques by presenting a real use-case scenario from the energy optimization domain. We demonstrate that long running computation jobs can be approximated using various techniques to reduce computation time without compromising the quality of results. This work identifies that approximation techniques, which have lower computational requirements, can be used directly closer to the data generation source to reduce latency of analysis, making
Figure 4.5: Idle time Overheads per job. ©2017 IEEE (reprinted with permission) Zamani et al. [3]. High standard deviation existed because various approximation techniques have been selected and they have wide range of execution times.

Figure 4.6: Average Cost. ©2017 IEEE (reprinted with permission) Zamani et al. [3].

more efficient use of available resources, improve acceptance ratio of tasks (i.e. enable a greater number of tasks to be completed within a given deadline), and provide a source of revenue for both owners of edge resources and network operators. Integrating approximation techniques with more conventional simulation can provide useful ways to improve utilization of our emerging computational infrastructure.
We have determined four different approximation techniques for EnergyPlus and investigated how these approximation techniques can be deployed at the edge the network and assessing their associated impact. Our results show that some approximation techniques cannot reach a desired accuracy threshold although completion time is improved. However when site resources are limited, a combination of the edge and approximation techniques help to increase the acceptance ratio.

4.8 Relevant Publications

This chapter contains portions adapted from the following published papers with permissions from the copyright holder.


Chapter 5
Workflow Scheduling and Runtime Management of Data Quality Using Edge and In-Transit Resources

5.1 Introduction

In previous chapters (Chapter 3 and Chapters 4), we have focused on partial and approximate data processing at the edge and in-transit nodes. The focus of the mentioned chapters was mostly on IoT applications. In this chapter, we take another step toward streaming applications and focus on providing QoS data intensive streaming application by deploying workflows/functions on the nodes located between data source(s) and data sink(s). In this chapter, we target large scale observatory systems which are designed to provide the scientific community with open access to data generated from geographically distributed instruments and sensors. As the number of sensors and their accuracy (e.g., image resolution) increases over time, the volume, variety and velocity of generated data grows exponentially. In such ecosystem, processing large volume of data requires large amount of resources, which are typically not co-located with the data sources. Hence, Processing is usually carried out in external and remote locations within well-provisioned data-centers in public/private clouds or academics institutions. Advanced Cyber-Infrastructures (ACIs) such as XSEDE (e.g., Jetstream) and cloud (e.g., AWS) resources play an important role to address limited and scarce local resources by providing on-demand resources for the users. In order to use remote ACIs, the data should be outsourced to remote resources for further processing. \(^1\)

In this context, users and applications require to receive processed/transformed data with several particular constraints such as deadline, budget and quality. We refer

\(^1\)The results presented in this chapter have been published in the Proceedings of 2018 30th International Symposium on Computer Architecture and High Performance Computing [4].
to these constraints as Quality of Service (QoS). However, guaranteeing on-time data delivery within specific constraints imposed by the users in environments composed of heterogeneous resources such as network links, virtual machines and bare metal servers requires sophisticated service/resource coordination. Moreover, in environments where data is continuously generated and processed via complex workflows, providing the guaranteed QoS and data quality for a long period of time while avoiding QoS degradation requires a comprehensive monitoring.

In this chapter, we propose a framework that integrates and utilizes heterogeneous distributed resources to process data while it moves towards the users, and manages data resolution in order to satisfy QoS requested by the users. We prove that edge and in-transit resources can be leveraged to process the data and adjust the data quality/resolution while it is moving between geo-distributed nodes. It considers users' constraints (deadline, budget and data resolution) and status of the resources in order to deploy workflows and coordinate data streams. A key feature of our approach is the ability to manage the data resolution being delivered to the users at runtime based on the comprehensive monitoring of the streams.

The main contribution of this work is to deploy the workflow stages over geo-distributed resources located between data source and destination, and leverage on-demand feedback loops to ensure end-to-end QoS for the users. Our work involves design and deployment of a subscription-based data streaming framework, consists of Kafka clusters [66], that incorporates edge and in-transit resources for workflow deployment and dynamically adjusts the quality of data at runtime. We also propose a computational model to deploy stream oriented workflows on heterogeneous resources. This framework targets large scale observatory applications, along with infrastructures with similar characteristics such as IoT applications and cyber-physical scientific experiments.

The remainder of this chapter is organized as follows. Section 5.2 motivates our work by discussing current data delivery limitations in scientific observatories. The problem of allocating workload using a comprehensive mathematical model is formulated in Section 5.3. The proposed framework and its implementation are explained
in Section 5.4. Experimental results are presented in Section 5.5. Finally, Section 5.7 concludes the paper and outlines future work.

5.2 Data Delivery Limitations in Scientific Observatories

Large-scale scientific facilities are essential part of the science and engineering enterprise. The generated data products from sensors and instruments within these facilities are made to be accessible for users around the world. Each second, massive amount of data is generated from distributed devices that needs to be processed in a timely manner. As the size of the data grows, processing and storing these data becomes challenging, costly and time consuming. These challenges cause limitations which have negative impacts on scientific discoveries. Although there has been a tremendous effort by the community to use public/private cloud and ACI services [150, 151], there is still a huge gap between ACI and scientific facilities which cause the users to be part of deployment and delivery cycle.

In such environments, there is a need for more effective data delivery mechanisms that can better integrate large facilities with cyberinfrastructure services, dynamically and automatically provide execution environments and leverage multiple resources from different entities to provide QoS for the users. Furthermore, the quality of data flowing toward the users needs to be reduced or adjusted (if applicable) at runtime in order to satisfy more requests from users and overcome network bandwidth limitations.

As the resources near the sensors and devices are limited, data is usually processed at centralized data centers. However, with current trend in big data applications and network limitations, this model is no longer sustainable. Hence, a new model that can integrate the edge resources (closer from the data source) and the in-transit nodes (between edge and core resources) can increase the efficiency of workflow execution and data processing by filtering unwanted data and reduce network traffic.

Our proposed framework aims at executing stream-oriented workflows considering user requirements and constraints using geo-distributed resources. The framework makes decisions related to data movement between different components, and quality
of the processed data being delivered to the users. These decisions are static (to map workflow stages to the resources) and dynamic (based on current state of the resources, timing and data resolution). Current observatories and IoT applications require such framework to effectively deliver and process data by considering the heterogeneous nature and properties (computing power and cost) of the resources and the constraints expressed by users. Hence, a system that monitors the progress of the workflow to meet users’ demands is necessary. Finally, such framework can integrate ACIs into the processing cycles and fill the gap between ACIs and large scale observatories.

5.3 Problem Definition and Model

In this section, a mathematical model has been proposed to enable the mapping of the workflow stages to available heterogeneous geographically distributed resources considering deadline and budget constraints. The objective is to minimize overall wide area network traffic caused by each stream. The inputs of the model are the workflow description and constraints (deadline and budget) that are imposed by the users, and the status of resources. The output is the mapping between workflow stages and resources. Each stage, except source and sink, gets the data from its previous stage, performs several operations on the data and provides the data to the next stage.

Processing each of the consumers’ requests is considered a computational job in our system. Any given job $J$ is represented by a sequence of stages $S : \{S_0, S_1, ..., S_Z, S_{Z+1}\}$, forming a workflow pipeline. Data production stage considered as $S_0$ and data consumption stage considered as $S_{Z+1}$. Figure 5.1 describes an overview of the pipeline workflow model. The processed data needs to be sent to the consumer site for storage, visualization and potentially additional offline processing with historical data.

The constraints of a job $J$ includes: a deadline ($\text{Deadline}(J)$) by which results have to be placed at the destination, typically determined by users; and a budget ($\text{Budget}(J)$) describing the maximum amount available to the user to spend on computing job $J$.

There are a set of $q$ geographically distributed computing resources (nodes) $R : \{r_1, ..., r_q\}$ in charge of data processing and applying part of the workflow. $r_0$ and
$r_{q+1}$ represent producer and consumer hops, respectively. Consequently the available set of hops is defined as: $H : \{r_0, r_1, ..., r_q, r_{q+1}\}$. The following variables are used to characterize the problem:

- $P(r_i)$: The average number of tasks that resource $r_j$ executes per unit of time.
- $E(J, r_i)$: The time job $J$ spent computing at resource $r_i$.
- $ES(J, S_i, r_j)$: The time job $J$ spent computing stage $S_i$ at resource $r_j$.
- $Task\_num(J, S_i)$: The number of tasks that is associated with stage $S_i$.
- $CompCost(r_i)$: The cost per unit of time for using resource $r_i$ for computation.
- $T(J, r_i, r_k)$: The time spent transferring data between resources $r_i$ and $r_k$ for job $J$.
- $CostNet(r_i, r_k)$: The cost of using the network channel per unit of data size, between resources $r_i$ and $r_k$.
- $Bandwidth(r_i, r_j)$: The available network bandwidth, between resources $r_i$ and $r_j$.
- $Dist(r_i, r_j)$: The geographic distance between $r_i$ and $r_j$.

Additional variables $L_{i,j}(J)$ are used to determine the mapping of stage $S_i$ to node $r_j$.

$$L_{i,j}(J) : \begin{cases} 1 & \text{if stage } S_i \text{ is mapped to resource } r_j \\ 0 & \text{if stage } S_i \text{ is not mapped to resource } r_j \end{cases}$$
The production and consumption stages are mapped to the producer and the consumer sites, respectively.

Each stage is mapped to exactly one hop. Equation 5.1 shows that stage $i$ of the workflow should be executed on exactly one node and the stages are not preemptive.

$$\sum_{j=0}^{q+1} L_{i,j}(J) = 1 \tag{5.1}$$

As depicted in Figure 5.1, the size of data generated at stage $i$ is assumed to be $D_i$. Other than $D_0$ which is the data generated from devices, data resolution effects the size of the data generated by each stage. For the mapping function, we considered the minimum acceptable resolution to determine the size of data at each stage. The overall time needed to process a job $J$ is defined as:

$$\text{CompTime}(J) = \sum_{j=0}^{q+1} E(J, r_j) + \text{Transfer}(J)$$

the $\text{Transfer}(J)$ and $E(J, r_j)$ are measured as follows:

$$\text{Transfer}(J) = \sum_{i=0}^{z} T(J, S_i)$$

$$E(J, r_j) = \sum_{i=0}^{Z+1} ES(J, S_i, r_j) \ast L_{i,j}(J)$$

$$ES(J, S_i, r_j) = \text{Task\_num}(J, S_i) / P(\tau_j)$$

Basically, the total transfer time of job $J$ is equal to sum of the time spent transferring data for each stage (between current stage and next stage) excluding consumption stage. We consider that the time it takes for the producer to generate raw data and consumer to consume the processed data are negligible.

Data transfer time between stage $i$ and stage $i + 1$ can be measured as follows:

$$T(J, S_i) = \sum_{j=0}^{q+1} \sum_{k=0}^{q+1} L_{i,j} \ast L_{i+1,k} \ast D_i / \text{Bandwidth}(r_j, r_k) \tag{5.2}$$

The cost of computing job $J$, $\text{Cost}(J)$, is defined as:

$$\text{Cost}(J) = \text{CostExec} + \text{CostNet}$$

where the computational cost ($\text{CostExec}$) is defined as:

$$\text{CostExec} = \sum_{i=0}^{q+1} \left[ \text{CompCost}(r_i) \ast E(J, r_i) \right]$$
The cost of transferring data associated with a job ($CostNet$) is defined as:

$$CostNet = \sum_{j=0}^{q+1} \sum_{k>j}^{q+1} [\text{Datasize}(J, r_j, r_k) \times CostNet(r_j, r_k)]$$

Where the data size between two resources $r_j$ and $r_k$ is measured as follows:

$$\text{Datasize}(J, r_j, r_k) = \sum_{i=0}^{z} L_{i,j} \times L_{i+1,k} \times D_i$$

Equation 5.3 is derived by considering execution of stage $S_i$ on $r_j$ and stage $S_{i+1}$ on $r_k$.

These general formulations are subject to ensuring the QoS requirements of each processed job:

$$\text{CompTime}(J) \leq \text{Deadline}(J)$$

$$\text{Cost}(J) \leq \text{Budget}(J)$$

Aside from satisfying users’ constrains, due to the fact that the network bandwidth plays an important role in streaming engines, the objective of this model is to minimize overall wide area network traffic between different components for each stream. Geographic distance between different components and data size have been considered for data movement minimization. Our overall objective is to minimize the function below:

$$\sum_{i=0}^{q+1} \sum_{j=0}^{q+1} \text{Datasize}(J, r_i, r_j) \times \text{Dist}(r_i, r_j)$$

The proposed model is solved using a Linear Programming Optimizer, PuLP [152], to determine $L_{i,j}(J)$. It has been mathematically proven in [153] that the mapping of linear workflows on the heterogeneous resources is NP-complete and finding an optimal solution can take a prohibitive time, if the number of resources increases dramatically; however, an approximate solution can be reached by considering a subset of the nodes for the optimization process. To show the amount of time that is needed to schedule the workflows, we run our optimization approach on Amazon t2.micro instance for various number of workflow stages and resources. As depicted in Figure 5.2, by increasing number of stages and resources, the time that is needed to find optimal solution increases. Other strategies such as random mapping, minimization of cost or execution time can be considered and deployed [1, 154].
Figure 5.2: The time to reach an optimal solution for different number of workflow stages and resources.

5.4 Stream-Oriented Data Processing Framework

In this section, a framework has been proposed which targets the execution of stream-oriented pipeline workflows/applications. These applications are modeled and executed as sequential functions and modules [153] which traditionally have been called linear workflows. The proposed framework is built on top of the geo-distributed ACIs and deploys the workflow stages on available resources at different locations based on the origin and destination of data. Moreover, it monitors the execution and progress of the workflows at runtime.

5.4.1 Overall Architecture

An overview of the architecture of the framework is illustrated in Figure 5.3.

*Execution/Delivery space* consists of geo-distributed resources, and data sources such as sensors and instruments that are part of the observatories. Resources join the *Execution/Delivery space* by executing light-weighted *agents*, in charge of giving resources access to the federation layer, managing local resources and sending status reports. The main components of *Execution/Delivery Space* are:

**CDN servers**: CDN servers are responsible for moving the data toward the clients.

The proposed streaming engine relies on *Apache Kafka*, a distributed streaming platform that stores streams of records in categories called *topics* [66].
Producer and Consumer nodes: Producer nodes are responsible for getting data from large scale observatories and pushing it to one of the available CDN servers. Consumer nodes are the end-users requesting processed data with specific constraints.

Compute nodes: Compute nodes process data streams and apply workflow stages/functions on data while it moves toward destination.

The Federation layer enables the coordination of resources and allows them to join and leave the federation as needed. The Broker provides an interface for the programmers and end-users to interact with the framework. It translates high-level instructions from the users to low-level instructions used by the Runtime layer. Constraints, priorities and workflow description are provided by the users through the Broker. The focus of this chapter is on the Runtime layer, which provides the following capabilities:

1. Scheduling: This function maps the stages of the workflows to the heterogeneous resources considering QoS, location and available bandwidths.

2. Deployment: After scheduling stage, the deployment layer installs the routes between the nodes and starts the execution by setting the nodes ready for requested stream.
3. Monitoring: Our framework deploys control loops to check status and progress of the workflows/streams. The monitoring service defines the execution plan and rules for the nodes and asks them to notify the monitoring service if the progress of the workflow does not follow the execution plan.

4. Stream Database: This module stores an up-to-date information regarding the status of the streams currently being delivered to users. The main benefit of this module is to help scheduler in reducing the amount of streams/data going toward the users and clustering the requests which is explained below.

5.4.2 Combining Requests

In scientific and IoT ecosystems, there is a significant amount of temporal locality among the client requests\cite{155}. Also, in large scale observatories there are many cases that users/application are interested in particular data sources or requesting to apply same workflow on the same data [155, 156]. For example, users might want to receive the processed data if specific event occurs in one of the data sources, or if applications running on distributed nodes are requesting information generated from the same data sources in (near) real-time. In [155], Shannigrahi et al. explored that current request patterns in large climate data distribution. They have shown that the requests are indeed aggregatable and data aggregation can reduce the load on the servers.

In this context, we propose a solution that leverages shared resources wherever and whenever possible to optimize resources usage, and process more requests by sharing resources across the requests. If the data source and the workflow that is requested by users are similar, leveraging this approach causes multiple streams to be clustered together which removes the redundant executions and data transfers. The main benefit of this approach is in reducing the bandwidth and computing resource usage per request by sharing resources between multiple requests. Combining/Clustering of the streams is done based on the user’s geographic location, requested data and workflow, the current stream being delivered, and users’ preferences.
5.4.3 Implementation

5.4.3.1 Subscription Based Data Movement

A publish/subscribe messaging system is considered as the main technique for data movement between the components. When the manager receives a request from the consumer, it maps the workflow stages to the nodes and finds an appropriate path. Then, the deployment layer installs the necessary subscriptions on the nodes that are going to be involved in each particular data stream. After the data path is set, the stream of data moves toward its destination. Figure 5.4 shows how subscription-based data movement is performed within the execution space. Specifically, the deployment layer provides each node with topics and IP addresses for publish/subscribe method. Using this approach, data moves towards the clients without any requirement on handling the data transfer procedure for every data chunk. If part of the execution is assigned to any of the compute nodes, it subscribes to the associated data topic and CDN server, applies the function on the stream and publishes the partially/completely processed data back to the CDN server (as depicted in Figure 5.4).

Figure 5.4: Subscription-Based Data Movement Using Distributed Kafka Clusters. ©2018 IEEE (reprinted with permission) Zamani et al. [4].
5.4.3.2 Resource Join Procedure

Each component (CDN, producer, consumer and compute) is able to access the execution space by knowing the IP address of one of the bootstrap nodes and sending join/leave requests to that IP. Upon receiving a join request, the service coordinator provides the IP address of CDN servers present in the system. Each component runs iperf [157] to the provided IP addresses in order to measure the available bandwidth between itself and remote IP addresses. This bandwidth information is reported to the manager that stores them in a database and virtually creates a network of nodes. This information is periodically measured and reported to the manager. Based on the network connectivities, the manager assigns a CDN server to providers, consumers and compute nodes. A production solution could be built on top of perfSONAR [158], which is a multi-domain network monitoring and measurement framework.

5.4.3.3 Monitoring and Approximation

The monitoring is implemented through the series of rules/thresholds, actions and reactions which are established by manager. Rules/thresholds, actions and reactions create feedback loops between resource and monitoring systems which causes monitoring service to be able to control infrastructure and workflow progress. Rules/thresholds are constraints that are installed on the compute nodes. They are locally checked by each node before and after execution of each stage. Each of the Rules/thresholds is associated with an Action that is also installed on the compute nodes and indicates an action that each node should take if one of the thresholds is met or rules are violated. For each stream, a thread is created at each of the corresponding compute nodes that is responsible to get/publish the data, provide the data for compute process and check these Rules/thresholds and take the actions, if needed.

We consider data approximation, i.e. adjusting the data resolution, in two different ways. (i) In scheduling procedure, the minimum resolution that is needed for each request is considered for scheduling. (ii) Approximation is tied to monitoring services such that the rules and threshold can tell monitoring services if resolution of data
is sufficient. Then, based on this information, at runtime, the framework is able to change the resolution of the generated data at each node by increasing or decreasing it, if needed. The increase/decrease in data resolution is amended for the next data chunk by instructing previous nodes to publish data with higher/lower resolution. The decrease in resolution is also applied for current data chunk by publishing lower resolution data to next node.

In this paper, the *action* at each node is to inform monitoring service. The *reaction* is the decision of the monitoring service which is data resolution adjustment for upstream nodes. Considering the deadline and estimated data transfer and execution time, manager estimates the arrival time of data at each node. Based on the difference between actual arrival time and estimated arrival time of data (called *diff*), five different categories have been considered in this paper. These thresholds and reactions are listed in Table 5.1. The runtime strategy for data resolution is to start the delivery of the streams with the minimum required resolution and adjust the resolution at runtime. It is worth noting that we used fixed timing and conditions. However, strategies where timing and conditions are different for various requests and they are changing at runtime can be considered and deployed. In this paper, in order to be sure that data is delivered without delay, resolution reduction rate has been considered higher than induction rate (as mentioned in Table 4.1).

---

**Table 5.1: Conditions and Monitoring Reactions. ©2018 IEEE (reprinted with permission) Zamani et al. [4].**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Timing(s)</th>
<th>Monitoring Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>very early</td>
<td>$2 &lt; \text{diff}$</td>
<td>Increase resolution By 5%</td>
</tr>
<tr>
<td>early</td>
<td>$1 &lt; \text{diff} \leq 2$</td>
<td>Increase resolution By 2%</td>
</tr>
<tr>
<td>on-time</td>
<td>$0 \leq \text{diff} \leq 1$</td>
<td>N/A</td>
</tr>
<tr>
<td>late</td>
<td>$-1 \leq \text{diff} &lt; 0$</td>
<td>Decrease resolution by 5%</td>
</tr>
<tr>
<td>critically late</td>
<td>$\text{diff} &lt; -1$</td>
<td>Decrease resolution by 10%</td>
</tr>
</tbody>
</table>
5.4.4 Combining the requests at runtime

Once a request is submitted to the system by users or applications, it is time to evaluate such request and schedule the workflow on the geo-distributed resources. However, scheduler first contact the stream database and ask for any possible way that this request can be satisfied using the existing data streams that are currently being served. If the requested resolution, data source and workflow can be satisfied by other request(s)/streams, these information will be sent to the deployment module. Then deployment module adds additional steps that should be taken to combine these requests. Next, the processed data will be delivered to the user/application and the request will be satisfied. If satisfying new request couldn’t be accomplished by combing requests, then scheduler tries to schedule the workflow from scratch using the model explained in Section 5.3.

In addition, since our proposed framework is based on a publish/subscribe model, clustering the streams is fast and feasible without the need adjust existing streams. It is fast and feasible because once it is decided that new request can be satisfied using existing data streams, we just need to add few additional subscription between the new data path and previous existing path. For example, in Figure 5.5, after it is determined by the runtime layer and manager to cluster new requests with an existing one, the client/user program is instructed to subscribe to corresponding topic and data that is already existed in CDN node to start receiving processed data.

The main advantage of our proposed approach is shown in Figure 5.5. In the regular streaming approaches, separate data streams with the same content are being sent to different users. However, in this work, we send one stream for the same data source and workflow to specific geographic region and add subscriptions at proper locations to save computing resources and more importantly network bandwidth resources. It is clear that when request clustering is available, redundant data transfers are eliminated.
5.5 Evaluation

5.5.1 Workflow

In this work, our target is to use images captured by OOI high quality underwater cameras as our image stream inputs. These images are processed in order to identify the different types of fish appearing in them. The image processing and object detection workflow used in this paper is derived from Dalal et al. [159]. Traditionally this method which is called *sliding window and image pyramids* is mainly used for object detection algorithms in image data sets. A stage of the workflow has been shown in Figure 5.6. Three consecutive sliding window stages have been considered for this paper.
5.5.2 Experimental Setup and Scenario

The experimental setup is comprised of three types of resources: Edge, In-transit and Core resources with 10, 20 and 60 compute nodes (each node requires 1 CPU and 1 GB RAM), respectively. Edge resources are the closest resources to the producer and the producer-edge bandwidth is higher than producer-in-transit and producer-core. Evaluation has been performed on CloudLab\(^2\), a distributed testbed for the computer science research community. It provides on-demand servers and virtual machines over distributed sites in the United States. Hierarchy Token Bucket (HTB) [160] has been used to deploy links with different bandwidths. Figure 5.7 presents a schematic view of our infrastructure.

Figure 5.7: Infrastructure Consisting of Producer, Edge, In-transit and Core Resources. ©2018 IEEE (reprinted with permission) Zamani et al. [4].

The cameras are generating 10MB size images every 10 seconds. The quality of service is deadline, budget and data resolution. If the system is able to process and deliver data within requested constraints, it accepts user requests and starts the delivery of the processed images.

---

\(^2\)CloudLab: https://www.cloudlab.us
In total, 194 users join the system, each user requests an independent image stream for a random period of time between 50 to 400 seconds. Users join the system following a Poisson distribution with a mean of 15 minutes and variance of 7 minutes. We considered a period of 30 minutes for each experiment. In all of the scenarios, for each request, we used the model described in Section 5.3 to map workflow stages to the available resources. Also, we considered a deadline of 25 seconds and assigned $1 budget for each image in the stream. The Amazon EC2 prices for bandwidth and the t2.micro template size for compute nodes has been considered in this work. The main focus of this paper is deadline and resolution constraints. If the system accepts to deliver and satisfy minimum QoS requested by the users, the minimum bandwidth that can satisfy such request will be allocated for that user and others streams cannot use that amount of bandwidth until the stream stops. This is enforced by controlling the amount of data that each node is allowed to produce per second for specific stream.

5.5.3 Results

In this section, several scenarios with various parameters for deadline and resolution have been considered to indicate the effectiveness of our framework in various conditions by comparing number of streams being delivered, resource utilization and handling change in execution conditions. Our baseline is a current state of the art solution which streams all the data to one central well-provisioned data center for processing, i.e., all the data goes to the core resources and the workflow stages implemented at central core data center using three workers for each stream. Figure 5.8 shows the utilization of the resources and number of streams being delivered to the user at any given time throughout the experiment for baseline scenario. Figure 5.8, demonstrates that although there are free resources available at the core, they remain unused due to the fact that bandwidth resources are limited and being used for previous requests and there is not enough bandwidth available for new requests. The utilization of the infrastructure at best is 55% and only 11 concurrent streams are guaranteed for delivery.

3https://aws.amazon.com/ec2/pricing/
with requested QoS.

![Figure 5.8: Utilization And Number of Streams without Approximation, Edge and In-Transit Resources. ©2018 IEEE (reprinted with permission) Zamani et al. [4].](image)

Using the same setup, with availability of edge and in-transit resources, the number of streams being delivered over time has been measured. As shown in Figure 5.9, if data resolution requested for the streams goes down, the concurrent number of streams that can be processed and delivered increases. This shows the effectiveness of our model in taking advantage of edge and in-transit resources to reduce the data size going toward core and end users.

In order to compare the baseline scenario with scenarios where edge, in-transit and approximation are available, we compare the acceptance ratio of different scenarios. Acceptance ratio is the percentage of the accepted requests to total number of requests. Figure 5.10 proves that using heterogeneous resources near data sources and using them to filter unwanted data increases the number of accepted requests and potentially users’ satisfaction.

The resource utilization for the edge, in-transit and core resource for three different data resolutions have been shown in Figure 5.11. With 80% data resolution, utilization of core resource can reach 48% which is slightly less than our baseline due to edge and in-transit participation. However, for data resolution of 60% and 50%, maximum utilization of 63% and 71% have been reached, respectively, which are higher than the
Figure 5.9: Number of Streams Being Delivered to Users Over Time for Different Minimum Acceptable Data Resolutions. ©2018 IEEE (reprinted with permission) Zamani et al. [4].

Figure 5.10: Comparing Acceptance Ratio in Baseline and Edge/In-Transit Enabled Scenarios for Different Qualities. ©2018 IEEE (reprinted with permission) Zamani et al. [4].

baseline scenario (55%). By comparing these figures, we conclude that by leveraging edge and in-transit resources, data is filtered before it reaches bottleneck links and more data can be injected to the core resource which results in more utilization at the core.
Figure 5.12 presented the maximum utilization that can be achieved for each resolution. As shown in Figure 5.12, as users’ requested utilization decreases, the utilization of our core infrastructure increases.

![Graphs showing utilization for different resolutions](image)

Figure 5.11: Utilization of Resources for Different Resolutions ©2018 IEEE (reprinted with permission) Zamani et al. [4].

Next, we show how system dynamically adjusts the streams resolution when the
Figure 5.12: Maximum utilization of the core resources for various scenarios and resolutions.

execution/delivery environment is changed at runtime. We used 12 concurrent streams for three different resolutions (100, 80 and 60), 4 streams each. We use the same infrastructure depicted in Figure 5.7. After 220 seconds, the network bandwidth between In-transit and Core nodes cuts down to 200Mbit/s. Figures 5.13 shows the number of streams and the average resolution of the streams being delivered to the user for various request types. For high resolution requests (e.g. 100%) the streams are stopped and nothing will be delivered to corresponding users as 100% strict condition does not allow system to adjust the resolution and the system cannot do anything to recover delivery of the streams. It is also shown that the number of streams for 100% resolution decreases after the incident and reaches zero (note: resolution zero means no streams for such resolution requirement is being delivered). For 80% minimum resolution requirement, our system shows more resistance by first reducing the resolution and then stopping the streams. However, for 60% resolution, the system is able to continue flawlessly by reducing the quality of the streams for a while. It is interesting to note that the resolution of 60% requests increases once other streams (100% and 80%) being dropped and more network bandwidth gets available for remaining streams (60%). Hence, the
system dynamically adjusts the resolution at runtime and overcomes the changes in execution space by reducing the resolution of the streams, if users are willing to sacrifice resolution.

Figure 5.13: Effect of Sudden Change in Bandwidth on Number of Streams and Average Resolution of the streams. ©2018 IEEE (reprinted with permission) Zamani et al. [4].

Finally, we studied the effect of different deadlines on the data resolution and number of accepted streams. We used similar setup as previous experiment and considered 80% as minimum acceptable resolution. The deadlines are 20, 25 and 30 seconds uniformly distributed across the requests. It is clear that the requests with a longer deadline
require less bandwidth as they have more time available for data transfer. Hence, in general, more streams with a higher deadline are accepted (as shows in Figure 5.14a) and data is being delivered with higher resolution for longer deadline requests (as shown in Figure 5.14b).

Figure 5.14: Number of Users and Average Resolution at Runtime for Various Deadlines and Resolution of 80%. ©2018 IEEE (reprinted with permission) Zamani et al. [4].
5.6 Evaluation

In this section, we run another set of experiments in order to show the effectiveness of our proposed solution in request aggregation/combination. In this section, data delivery and processing without request aggregation is considered as our baseline scenario. We consider two metrics: (i) acceptance ratio and (ii) the number of requests being served throughout the experiment. Then, we measure these two metrics for the baseline scenario and request aggregation-enabled approach and demonstrate the advantage of our approach.

The experimental setup of this experiment is similar to the previous experiments, shown in Figure 5.7. The duration of the experiment is 30 minutes where 194 users join the system with Poisson distribution with a mean of 15 minutes and variance of 7 minutes. Users are staying for random period of time and requesting the camera images to be processed using the sliding window image pyramid (shown in Figure 5.6), and delivered to them with 100% image resolution. The deadline is set to be 25 seconds. The only different in this experiment is number of available data sources (cameras). We have increased the number of available data sources/cameras. We have considered 10, 20, 30, 50 and 100 data sources for different scenarios. Users pick one of the data sources randomly and request data to be processed and delivered to them. All of the users are located at the proximity of each other. Also, we assume that the users are located at one geographic region.

To prove the effectiveness of our request aggregation approach, we measured the acceptance ratio for the mentioned scenarios. In general, when the number of available data sources decreases, the possibility to choose a common data source for different users increases. Hence, our framework is able to cluster and accept more requests if the number of data sources is bounded. Increase in the available data sources also decreases the acceptance ratio as users are able to choose from wide range of data sources and there will be less chance for request clustering/aggregation. Figure 5.15 shows that with 10 data sources, the framework is able to accept 100% of the requests. However, this number has been reduced to 64% with 20 data sources. Acceptance ratio for 100
data sources is 41% which is slightly higher than the baseline scenario which is 35%.

![Graph showing acceptance ratio across different scenarios.](image)

Figure 5.15: Acceptance Ratio for baseline and data merging scenarios. Acceptance ratio decreases when number of data products increases.

Figure 5.16 demonstrates the number of requests being served throughout the experiment. As depicted in Figure 5.16, with fewer number of data sources, more requests can be satisfied due to our request clustering approach. As we combine the request, we reduce the bandwidth and resource usage per stream. In fact, with the same infrastructure, by sharing the resources across the requests, more users will be able to receive their processed data within requested QoS.

### 5.7 Discussion

Large scale observatories rely on the efficient processing and delivery of data generated from geographically distributed instruments and sensors. This paper introduces a subscription based data streaming framework and a runtime management system that provides QoS for the users and effectively utilizes heterogeneous geo-distributed resources to maintain application’s quality of service. The proposed framework fills the gap between large scale observatories and ACIs by automatically executing user-defined
pipeline workflows on available geo-distributed resources. It also adjusts the data quality by taking advantage of the resources at the edge and in-transit and filters unwanted data going through the core resources. The evaluation showed that our system can increase main resources utilization by more than 10% using unwanted data filtering at edge/in-transit nodes for low resolution requests.

### 5.8 Relevant Publications

This chapter contains portions adapted from the following published papers with permissions from the copyright holder.


Chapter 6
Conclusion

6.1 Summary

Data intensive applications and workflows are designed by scientists and engineers to process data and extract insight from large amounts of data that is generated from distributed sensors and data sources. Moreover, by entering the big data era, emerging data intensive applications are playing an increasingly important role in science and engineering. Execution of these workflows and applications has been relied on distributed public and private clouds and clusters and advanced cyberinfrastructure. Traditional models designed to process large amounts of data from distributed data producers are sub-optimal and inefficient because they fail to address bandwidth limitations and high latencies caused by data movement between distributed nodes.

In this regard, users and applications require receiving processed and transformed data with several particular constraints such as deadline, budget, and quality, which are referred to as Quality of Service (QoS). However, guaranteeing on-time data delivery within specific constraints imposed by the users in environments composed of heterogeneous resources such as network links, virtual machines, and bare metal servers requires sophisticated service/resource coordination. Furthermore, in environments where data is continuously generated and processed via complex workflows, providing the guaranteed QoS and data quality for a long period of time while avoiding QoS degradation requires comprehensive monitoring.

In this dissertation, we took concrete steps to handle big data that should be processed in a timely manner. These steps are summarized below.
6.1.1 Leveraging SDN to exploit and provision in-transit resources

In chapter 3, we studied the effectiveness of SDN on leveraging in-transit resources. We also proved the impact of edge and in-transit data processing in cloud federation ecosystem. Moreover, we showed that edge and in-transit data processing can increase the job acceptance ratio drastically by processing the data while it moves between nodes and filtering unwanted data to save network bandwidth. Also, we proposed a new in-network computational model that leverages resources distributed across the network, including edge devices and network data centers. We described how integrating SDN capability into our federated infrastructure can enable the use of resources located at the network data centers to perform in-transit computation of data that is being transferred.

Moreover, we proposed a strategy that leverages edge devices to prioritize workload processing depending on the estimated value of the data. By this means, we were able to increase the amount of data processed and consequently, increase the overall value of the obtained results. We used a video surveillance application as a use case and tested several scenarios to show the feasibility and benefits of our proposed computational model by making use of edge and in-transit data analysis.

Furthermore, we showed that edge and in-transit resources are able to increase job acceptance ratio and data quality in cloud federation environment that is composed of distributed resources. More importantly, this work enables in-transit data processing and network resource provisioning by leveraging SDN capabilities [1, 5, 132, 154].

6.1.2 Using edge and in-transit resources to increase quality of solution

In chapter 4 we took another step towards incorporating QoS into workflow deployment. Hence we added approximation capabilities to our framework. We provided a computational model to process part of the data on edge and in-transit nodes and automatically use an approximation technique to satisfy end-to-end quality of service [3, 24, 154, 161].
This framework has been tested through a smart building application called EnergyPlus. Our experiments showed that EnergyPlus optimization requires large amounts of computational resources, which affected the number of jobs that the system was able to accept as well as the associated job completion ratio (i.e. the number of tasks within a job which could be completed by a deadline). In order to (i) reduce potential resource requirements and (ii) improve use of in-transit resources, we proposed to automatically use different approximation techniques such as loop reduction, value skipping, interval reduction, and trained artificial neural network (ANN) models alongside EnergyPlus. The approximation-based model complements the execution of EnergyPlus and provides an approximation of the EnergyPlus output. Using edge/in-transit approximate deployment, we were able to accept more requests and increase the accuracy of the final results.

6.1.3 Scheduling of the stream-oriented workflows on edge and in-transit nodes

In chapter 5, we proposed a subscription-based data streaming framework that moves the data effectively between geo-distributed nodes, leverages such resources, applies workflow stages on the streaming data, and controls the data quality using on-demand control loops. Moreover, we explored scheduling scientific workflows on edge and in-transit resources. Specifically, we investigated the impact of in-transit staging, caching, approximation, and compression/decompression services on the workflow execution. Furthermore, we proposed methods to optimize data movement in wide area networks and provide on-time delivery. Finally, we studied the runtime management of data quality using edge and in-transit nodes for wide area streaming applications and proposed an effective mechanism to control data quality and apply workflow stages/functions on the stream of data to improve the end-to-end QoS required by workflows [4, 162]. The effectiveness of the framework has been proven using OOI underwater high quality images.
6.2 Prospectives

In many data intensive applications, the requirement for providing tighter coupling between the workflow components is inherent. For instance, many applications require sharing of large data volumes with varying hard (on-time data delivery) and soft (with some degree of freedom) QoS requirements. Whereas previous work in workflow enactment over distributed infrastructures have often emphasized the need to provide loose coupling between services that make up the workflow, in some applications where on time processing constraints need to be observed, it is necessary to also manage connectivity between such services. Enabling loose coupling between services whilst at the same time having the ability, when necessary, to control data transfer between these services, provides a useful capability that is made possible with SDNs. In particular, for those applications where the performance of the services/processes are depend on time, the interval between the generation of data at a producer and subsequent consumption of this data can have significant impact on the execution of the workflow. The implications of such time-sensitive applications are complex as the process of data delivery can have significant impact both on the producer and the consumer. In addition, applications may also require different data representations between the source and destination. Hence, the data has to be transformed in a timely manner before it can be consumed.

Furthermore, Scientific workflows that process data from large scale observatories are typically composed of multiple steps such as data calibration, data transformation, computational modeling, analytics, visualization, and result collection. Also, data dissemination from the observatories involves the end-to-end delivery of processed data from the data source(s) to one or more data consumer(s) using these workflows across a wide area environment. This is challenging for several reasons. First, transferring massive amounts of data over limited wide area network resources within prescribed time constraints can be difficult. Second, workflows involve non-trivial processing that require significant resources, which may not be co-located with the data producer or consumer. Finally, variability in resource availability and performance requires runtime
adaptation as well as the use of approximation techniques to meet time and quality constraints. Therefore, a novel cyberinfrastructure ecosystem that integrates non-trivial resources and services along the data path can be used to address the challenges in executing data processing workflows, potentially accelerate scientific discoveries and satisfy more requests submitted by users around the world.

6.3 Future Work

The framework that is proposed in this dissertation provides new opportunities for scientists and big data application developers to move data effectively between distributed nodes. This framework is specifically useful for upcoming data processing and cyberinfrastructure models that rely on data transfer between nodes, making in-transit data processing a viable solutions.

The research presented in this dissertation can be expanded in several areas:

- Dynamic bandwidth allocations: Our SDN model can be extended to incorporate dynamic bandwidth allocation that can help us further improve the use of the infrastructure while obtaining the maximum value out of the data. This method can be used to give priority to specific data streams that have higher priority and tighter QoS requirements such as deadline. Based on the current status of the resources, SDN is able to change the bandwidth assigned to each data stream at runtime, which helps in adjusting the network status to meet end-to-end QoS requirements for various applications/users.

- Multi-path data processing: Existing systems such as our proposed framework can be extended by methods to divide the data and distribute each batch to different paths and expose the data to more in-transit nodes for faster and robust data processing. In this case, data processing can be faster since there are more resources to process the data and network resource contention is reduced due to the reduction in the data transfer in each path. This method is also more robust to failures in networking infrastructure. More specifically, when a path encounters a data delivery problem, other links can still deliver the rest of the data to the
applications/users.

- Multiple data sources or sinks: If a workflow requires data from multiple data sources or sensors, there is a need to optimize data execution for multiple data streams. This means that data can be merged at optimal places to avoid transferring all data to a central unit. Hence, data can be merged at the earliest or nearest nodes to save bandwidth. Moreover, in cases where multiple users and applications request data from the same data source(s), it would be beneficial to combine the requests and reduce the number of streams flowing towards specific geographic locations.

### 6.3.1 Scalability

While this dissertation provides a framework for deployment of the stream-oriented workflows on the nodes located between source and destination, more work is required to make such system highly scalable. In order to increase scalability of our framework, the proposed centralized services need to be deployed in a distributed manner.

In this dissertation, a centralized SDN controller has been used to control networking infrastructure and provision in-transit resources. As mentioned before, SDN controller can be deployed in logically centralized and physically distributed manner [39]. Another important aspect that can be explored in future work is to deploy SDN controllers in distributed fashion where each SDN controller is responsible for specific domain or geographic area and they work collaboratively together in a logically centralized form. In this case the system will be highly scalable.

Moreover, in Chapter 5, we used a centralized manager to schedule the workflows and monitor the execution of the workflows. Manager and monitoring system can be also deployed in a distributed fashion. For instance, same as SDN controller, a separate monitoring node that is responsible for the specific domain can be utilized and deployed. Each compute node should report to its designated monitoring node. As a result the messages that needs to be passed between each node and its assigned monitoring node will be delivered with less latency and the system would be scalable.
6.4 Relevant Publications

This dissertation contains portions adapted from the following published papers with permissions from the copyright holder.


References


[67] Ivo Santos, Marcel Tilly, Badrish Chandramouli, and Jonathan Goldstein. Dial:


[133] J. Diaz-Montes, Y. Xie, I. Rodero, et al. Federated computing for the masses -


[141] Lawrence McAfee and Kunle Olukotun. Emeuro: A framework for generating


