PRIVACY-PRESERVING PLATFORMS FOR COMPUTATION ON HYBRID CLOUDS

ZHANG CHUNWANG

(B.Sc, Fudan University)

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2014

DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Zhang Chunwang

Zhang Chunwang

20 September 2014

Acknowledgements

First, I would like to express my sincere gratitude to my PhD advisor, Associate Professor Chang Ee-Chien, for his constant support, guidance and encouragement throughout my PhD study. He has been always patient and positive on me, brightening me many times when I encounter difficulties in my research and study. His rigorous attitude of scholarship, limitless passion on work as well as cordial and amiable style in life all have deeply influenced me. Without his advice and guidance, this thesis would not have become possible.

I would like to thank Associate Professor Roland H. C. Yap for his great ideas and extensive advice on the first work of the thesis. I would like to thank Associate Professor Ooi Wei Tsang for the numerous discussions and invaluable suggestions on the second work. I also wish to thank Associate Professor Liang Zhenkai for his help in my life and helpful suggestions on my whole PhD thesis work.

My stay in NUS would not have been so wonderful without my fellow students and friends. In particular, I would like to thank Dr. Xu Jia and Dr. Fang Chengfang for their countless helps and encouragement. It has been such a fruitful and pleasant experience working with them. I also wish to thank Dr. Dong Xinshu, Zhang Mingwei, Li Xiaolei, Dai Ting, Hu Hong, Jia Yaoqi, Zhu Xiaolu, Zhang Dongyan and many others for bringing so much joy and color to my life. In addition, I am also thankful to the friends in the SeSaMe centre for providing so many helps in all the matters related to video surveillance and sensing.

Lastly, my most heartfelt thanks go to my parents and my wife. I could not and would not have made it without their constant love and encouragement. They gave up a lot while offering everything I want. They are always there when I need them.

To my parents and my wife

Contents

1 Introduction **1**

2 Background **9**

2.1 Cloud Computing 9

2.1.1 Service Models 10

2.1.2 Cloud Advantages 12

2.2 Hybrid Clouds 13

2.2.1 Definition and Current Status 13

2.2.2 Scheduling on Hybrid Clouds 16

2.3 Secure Computing on the Cloud 17

2.3.1 Encrypted Domain Processing 17

2.3.2 Trusted Computing and Secure Hardware 19

2.3.3 Data Segregation Using Hybrid Clouds 19

3 Privacy-preserving MapReduce Computation on Hybrid Clouds **21**

3.1 Introduction 21

3.2 Overview 24

3.2.1 MapReduce 24

3.2.2 Overview of the Proposed Framework 28

3.3 Programming Model 30

3.3.1 Sensitivity Policy 31

3.4 Scheduling Modes 34

3.4.1 Two-Phase Crossing Mode (Partitionable Reduce) 35

3.4.2 Two-Phase Non-Crossing Mode 36
3.4.3 Hand-Off Mode (Unique Tag) 37
3.4.4 Mode Selection 38
3.5 Security Analysis 39
3.5.1 Motivating Examples 40
3.5.2 Scheduler-View and Public-View 41
3.5.3 Baseline - the Conservative Scheduler 42
3.5.4 Security Model 44
3.5.5 Leaky Implementation 45
3.5.6 Security of the Proposed Modes 47
3.5.7 Side-Channel Information 48
3.6 Implementation 48
3.6.1 Hadoop Overview 49
3.6.2 Input Data Tagging 52
3.6.3 Data Uploading and Replication 53
3.6.4 Map Task Management 53
3.6.5 Reduce Task Management 54
3.7 Evaluation 55
3.7.1 Experimental Setting 55
3.7.2 Experiments on Scheduling Modes 57
3.7.3 Experiments on Different Baselines 63
3.7.4 Experiments on Different Public Cloud Sizes 64
3.7.5 Experiments with Chained MapReduce 66
3.8 Extension – Routing Traffic through a Proxy 67
3.8.1 Main Idea 68
3.8.2 Implementation and Evaluation 69
3.9 Discussion 70
3.10 Summary 72

4 Privacy-preserving Video Surveillance Stream Processing on Hybrid Clouds 73

4.1 Introduction 73

4.2 Background on Video Surveillance 76

4.2.1 Video Surveillance Systems 76

4.2.2 Video Surveillance in the Cloud 78

4.2.3 Security and Privacy in Video Surveillance 79

4.3 Hybrid Cloud Video Surveillance Model 81

4.3.1 System Model 81

4.3.2 Stream Processing Model 82

4.3.3 Security Model 83

4.3.4 Cost Model 85

4.3.5 System Architecture 85

4.4 Problem Formulation 86

4.4.1 Optimization Problem 87

4.4.2 Extension of the Stream Processing Model 89

4.5 Proposed Approach 90

4.5.1 Transforming to Integer Programming 91

4.5.2 Minimal Configurations 92

4.5.3 Heuristic Selecting Method 94

4.6 Evaluation 96

4.6.1 Simulations 96

4.6.2 Proof-of-concept System Evaluation 101

4.7 Summary 104

5 Conclusions 105

Summary

In this thesis, we are interested in enabling efficient and cost-effective privacy-preserving computing on the cloud. Existing approaches on encrypted domain processing and trusted computing have been found limited, impractical or expensive. Instead, this thesis focuses on another approach of data segregation using hybrid cloud. With a hybrid cloud, one could properly segregate the data, pushing non-sensitive data to the public cloud while keeping sensitive data in the trusted private cloud. However, this computing model under hybrid cloud has not been well supported by many existing platforms. In particular, we look into two widely used platforms of MapReduce and video surveillance.

MapReduce is a popular framework for performing large-scale data analysis; however, MapReduce is designed for only one (logical) cloud and may leak sensitive data when working on a hybrid cloud. In view of this, we propose extending MapReduce by augmenting each key-value pair with a sensitivity tag. This tagging enables fine-grained dataflow control during execution to prevent information leakage. More importantly, the tagging provides increased flexibility by allowing sophisticated security polices and facilitating complex MapReduce computation. To address the performance issues introduced by the security constraint, we exploit useful properties of the MapReduce functions and present three scheduling modes which can rearrange the computation for increased efficiency while maintaining MapReduce correctness. A generic security framework is also provided for analyzing what information a scheduler can leak through execution on hybrid clouds. Experiments on Amazon EC2 show that our prototype on Hadoop is able to preserve data-privacy while effectively outsourcing computation and reducing inter-cloud network traffic.

We next consider processing of large-scale video surveillance streams on hybrid cloud. The challenge here shifts to problems of scheduling the processing tasks over the hybrid cloud so as to protect data privacy as well as to achieve certain efficiency. We first present a stream processing model that can take into account special properties of the hybrid

cloud in handling ad-hoc queries and dynamic clients. Based on this model, we formalize the scheduling challenge as an optimization problem to minimize the monetary cost to be incurred on the public cloud, subjected to several resource, security and Quality-of-Service (QoS) constraints. Our proposed scheduler exploits useful properties of the hybrid cloud for more efficient solutions and allows scaling to larger instances. Both the simulations and proof-of-concept system evaluation on Amazon demonstrate the effectiveness and efficiency of the proposed approach.

We conclude that privacy-preserving computation on the hybrid cloud can be made efficient, cost-effective and automatic. With the well-designed scheduling mechanisms, the overheads incurred by the security constraint could be significantly reduced.

List of Figures

2.1 The layered architecture of a cloud computing environment. 11

2.2 Illustration of a hybrid cloud. 13

2.3 RightScale 2014 State of the Cloud Report [5]. 15

3.1 Overview of tagged-MapReduce from the perspective of users and programmers. Shaded rectangles are files/tuples marked as sensitive, shaded ellipses are workers/scheduler run in the private cloud. Note that the output tuples carry sensitivity information which can be fed to the next job, thus multiple MapReduce computation can be naturally supported. 29

3.2 Example code corresponding to original map (left) and tagged-map (right) for the `WordCount` job. The difference is the code within the dashed box that computes and sets the tags of the output tuples. 32

3.3 Default scheduling mode: Single-Phase (SP). 34

3.4 Two-Phase Crossing (TPC) mode. 35

3.5 Two-Phase Non-Crossing (TPNC) mode. 36

3.6 Hand-Off (HO) mode. 38

3.7 Examples of the public-view corresponding to the scheduler-views illustrated in Figure 3.3–3.6. 43

3.8 Illustration of the baseline scheduler: (a) scheduler-view; (b) the corresponding public-view. 44

3.9 Two-Phase Crossing with Local-Reducer. 46

3.10 Overview of Hadoop distributed file system (HDFS). DataNodes store actual blocks from files while NameNode stores only the metadata. 50

3.11 High-level overview of Hadoop MapReduce workflow. 51

3.12 The word count example on Hadoop. Suppose we have two files, foo.txt and bar.txt. Two mappers (and reducers) were created to process them. Intermediate results with the same key were sent to the same reducer. . . . 52

3.13 Inter-cloud communication. 58

3.14 Job elapsed time. 60

3.15 Computation outsourcing ratio. 61

3.16 Monetary cost incurred on the public cloud. 62

3.17 Job elapsed time with different baselines. 64

3.18 Job elapsed time with different public cloud sizes (left: word count; right: sort). 65

3.19 Inter-cloud data traffic with different public cloud sizes (left: word count; right: sort). The 3 private nodes only setting does not incur inter-cloud communication, and hence is not shown in the figure. 65

3.20 Computation outsourcing ratio with different public cloud sizes (left: word count; right: sort). The 3 private nodes only setting does not involve computation outsourcing, and hence is not shown in the figure. 66

3.21 Routing inter-cloud traffic through a trusted proxy to prevent side-channel leakage on private worker identities. 68

3.22 Overheads of routing inter-cloud data traffic through a proxy. 69

4.1 Illustration of a video surveillance system [150]. Video cameras (and microphones) capture the events and activities of the environment. 77

4.2 Multiple cameras installed at a single site. 78

4.3 System model for hybrid cloud video surveillance. 81

4.4 Illustration of a stream processing task. 83

4.5 Architecture of the hybrid cloud video surveillance system. 86

4.6 Illustration of the case where there could be multiple ways to complete a task. 90

4.7 Configurations in the 2D load-cost graph. Those marked as dark red diamonds are the minimal configurations. 93

4.8 Illustration of the heuristic. 95

4.9 The task templates created for simulations. 97

4.10 Simulation result without security constraint (ProposedC, ProposedB and Greedy are indistinguishable in (c)). 99

4.11 Simulation result with security constraint (ProposedC, ProposedB and Greedy are indistinguishable in (c)). 100

4.12 Task template for proof-of-concept system evaluation. This task has two alternative ways to complete. 102

4.13 Experimental result of prototype evaluation. ProposedC and ProposedB produce the same result, hence they are rendered as one line (Proposed). . 103

List of Tables

3.1 Summary of the computing jobs and datasets 56

3.2 Mode assignment for individual MapReduce jobs 66

3.3 Experimental results on chained MapReduce 67

4.1 Study on the size of minimal configurations $\mathcal{MF}(T)$. 94

4.2 Effectiveness of the heuristic. 96

4.3 Time taken to solve the integer problem. 100

Chapter 1

Introduction

Cloud computing has drawn extensive attention from both academia and industry in the recent years. By combining a set of existing and new techniques such as virtualization and Service-Oriented Architectures (SOA), cloud computing provides scalable and on-demand resources as services over the Internet, at relatively low prices (e.g., $0.098 per hour for compute and $0.03 per GB for storage).[1] Successful examples of cloud services are Amazon AWS [24], Google App Engine [10] and Microsoft Windows Azure [12] etc. Organizations and individuals are increasingly realizing that, by simply tapping into the cloud, they can enjoy a wide range of benefits including reduced monetary cost, high scalability and availability, ubiquitous access etc. More and more data and applications are being moved to the cloud [143, 157].

However, cloud computing also faces a series of striking challenges which may impede its wider adoptions. Among these challenges, *data security and privacy* could be the most significant one. Users outsource their data to the cloud and lose physical control of the data. Yet cloud providers cannot be fully trusted due to various inside threats and outside attacks. For example, many data breach incidents have been reported over the years for various cloud service providers [29, 31, 37, 123]. NSA has been revealed to secretly tap into Google and Yahoo data centers and collect data "at will" [30]. Ristenpart

[1] Prices are taken from Amazon EC2 and S3 respectively, for the Asia Pacific (Singapore) region as in May 2014. The compute cost is measured for on-demand Linux/UNIX instances of `m3.medium` type; the storage cost is measured for a total space requirement of less than 1 TB per month.

et al. [141] also demonstrated that confidential information can be extracted through side-channel leakage across virtual machines (VMs) resided on the same physical machine. Indeed, data security and privacy has long been ranked as one of the top concerns in the cloud [5–7]. Oftentimes organizational data involve both sensitive and non-sensitive information, e.g., an organization's file system may contain general (non-sensitive) files mixed with confidential business data. Also, many datasets for analytical tasks such as network logs, email archives and healthcare records may involve data from public sources mixed with sensitive private data. Computation on such mixed-sensitivity data should not be carried out on the unsecured clouds without security measures to prevent data leakage.

There are multiple ways to achieve privacy-preserving computation in the cloud. In the simplest form, one could encrypt the data, e.g., using AES, before outsourcing them to the cloud. This simple solution, however, would give rise to technical challenges when computation has to be carried out on the data. Researchers have developed multiple cryptographic primitives to support encrypted domain processing. The main idea is to encrypt the data in a proper form such that the cloud can compute on the encrypted data without learning any plaintext information. Some traditional encryption schemes are partially homomorphic, supporting only limited operations such as addition for ElGamal [80] and multiplication for Paillier [131]. They allow outsourcing of specific applications such as modular exponentiation [92] and linear algebra [38], but are not suitable for general-purpose computation. In 2009 Gentry [84] presented the first construction of a fully homomorphic encryption (FHE) scheme which supports evaluating arbitrary functions on the encrypted data. Unfortunately, FHE scheme is still not practical [151] though a number of improvements and implementations have been made over the years [50,83,85,155]. Another line of research work utilizes trusted computing techniques [17] to establish a secure and isolated computing environment in the cloud which can handle sensitive data and operations [53, 65, 173]; however, such approaches still require trust on a certain amount of hardware such as CPUs which is under physical control of the cloud providers. Also, these secure hardwares are usually expensive and relatively slow. They do not qualify as

building blocks for a cost-efficient and scalable cloud computing infrastructure.

In this thesis, we are interested in enabling efficient and cost-effective privacy-preserving computation on the cloud. In view of the limitations and difficulties of the above existing solutions, we focus on another approach of data segregation using hybrid cloud. A *hybrid cloud* essentially combines a private and a public cloud. The private cloud could be an organization's existing internal data center, on which the organization have full control and can trust. The public cloud is typically one of the general commercial clouds mentioned before. A seamless integration of these two clouds offers increased scalability and cost-effectiveness. The private cloud can be used for typical workloads which fit within the local resources, but when additional resources are needed during peak computation, the public cloud is harnessed. This hybrid cloud model has gained wide adoptions and is still undergoing rapid development [1, 5].

While the hybrid cloud model was initially proposed to handle the issues of scalability and dynamic workload, it can also be used to address the security issues. With a hybrid cloud, one could segregate the computation on non-sensitive data from that on sensitive data, such that the former can be comfortably outsourced to the public cloud while the latter, possibly much smaller in size, can be easily handled on the private cloud. In this way, the computation can be carried out both securely and efficiently. Unfortunately, this computing model under hybrid clouds has not been well supported by many existing cloud platforms. As a result, users have to manually separate the data into two partitions, compute sensitive (or non-sensitive) partition on the private (or public) cloud, and then combine the partial outputs using their own code. This process is neither efficient nor transparent. We want to provide platforms that can automate this process and make privacy-preserving computation in the cloud efficient, cost-effective and automatic. In particular, we look into two widely used platforms of MapReduce and video surveillance.

MapReduce [74] is a popular framework for processing huge data sets in a cluster of commodity machines. Conceptually, MapReduce divides a huge problem into multiple smaller sub-problems (or map/reduce tasks) and provides a seamless distribution of

these tasks among nodes in the cluster in a way which is transparent to the programmers/users. Users only implement the two map and reduce functions, without caring about the complex issues of task-scheduling and data movement. Unfortunately, MapReduce is designed for only one single (logical) cloud and does not distinguish between data and machines with differing sensitivity. From the viewpoint of MapReduce, all data are identical in terms of sensitivity and all machines have the same level of trust. Hence, if used on a hybrid cloud, MapReduce cannot prevent sensitive data/information from being leaked to the untrusted public cloud.

To address this problem, we propose *tagged-MapReduce*, a conservative extension to the existing MapReduce framework. Tagged-MapReduce augments each key-value tuple in MapReduce with a *tag*, which is a small piece of meta data indicating the sensitivity of that tuple. Meanwhile, the map and reduce functions are also modified to work on tagged-tuples appropriately. The sensitivity tags enable the platform to do fine-grained dataflow control during execution to prevent information leakage: once a tuple is tagged as sensitive, it cannot leave the private cloud. More importantly, the tagging provides increased flexibility by: 1) allowing programmers to code sophisticated *security policies* in map/reduce programs to guide sensitivity transformation during execution, and 2) providing sensitivity information for data across multiple MapReduce computation, which is necessary for many real-world applications involving chained or iterative MapReduce. The flexibility in turn allows legacy MapReduce programs to be easily supported. To address potential performance issues introduced by the security constraint, we investigate useful properties of the map/reduce functions, namely partition-able reduce and unique tag, and propose several *scheduling modes* that can rearrange the computation for better performance while preserving data-privacy and maintaining MapReduce correctness. We further present a generic *security framework* that can capture and analyze what kind of information leakage a scheduler can make through execution on hybrid clouds. This security framework can be used to compare the information leakage of different schedulers and to determine whether a scheduler is secure or not. We have prototyped tagged-MapReduce

on Hadoop [18], a well-known open-source MapReduce implementation. Experiments on a small hybrid cloud we built on Amazon EC2 show that tagged-MapReduce can effectively preserve data privacy on hybrid clouds, outsource more computation to the public cloud and reduce both inter-cloud communication and monetary cost.

Next, we consider processing of large-scale video surveillance streams on hybrid clouds. Video surveillance is a widely used application which deals with large data and also has privacy issues. The challenge here lies on how to schedule the stream processing tasks on the hybrid cloud so as to protect video privacy which achieving certain efficiency. Such scheduling decisions cannot be manually made by system administrators due to high system dynamics as well as various factors in consideration. Thus, it is desired to have a platform that unifies the two clouds and schedules the tasks securely and effectively. We first give a stream processing model that is specifically designed for the hybrid cloud setting. This model takes into account special properties of the hybrid cloud and can handle ad-hoc queries and dynamic clients without rescheduling mostly. Based on this model, we then formalize the scheduling problem as an optimization problem to minimize the monetary cost incurred on the public cloud, with several constraints being satisfied, namely resource, security and Quality-of-Service (QoS). The optimization problem itself is NP-hard; however, our proposed scheduler can exploit specialized properties of hybrid clouds for more efficient solutions. Essentially, for each task of the input, we convert it to a set of configurations and search for the "minimal configurations", and then employ integer programming to select the desired configurations. To guarantee that the integer problems are sufficiently small, we further provide a heuristic to select only a few representatives. Experiments through both simulations and proof-of-concept system run on Amazon EC2 illustrate the effectiveness and efficiency of the proposed approach.

The above two work investigated into fundamental issues for two widely used programming paradigms. Through these two work, we demonstrated that privacy-preserving computation on hybrid clouds can be made efficient, cost-effective and also automatic. For future work, we plan to extend our ideas to other platforms such as Apache Spark [23]

as well as to combine with practical encryption schemes. In addition, we are also interested in providing routing anonymity for cloud computing so that the leakage from dataflow can be prevented. This would complement existing research work on cloud security.

Contributions

This thesis addresses the issues of data security and privacy in cloud computing. In view of the limitations of the existing solutions, this thesis focuses on a different approach of segregating computation in the emerging hybrid cloud setting. More specifically, the thesis studies how to partition and schedule computation with mixed-sensitivity data on hybrid cloud systems so as to preserve data privacy and achieve increased efficiency. The two platforms studied in the thesis, namely MapReduce and video surveillance, represent two popular programming paradigms for cloud computing. This thesis work provides one of the first platforms that automate and make effective the process of privacy-aware computation on hybrid clouds.

The work completed in the thesis made two major contributions.

- We proposed tagged-MapReduce (Chapter 3), the first generic and flexible framework to support privacy-aware computation on hybrid clouds, and gave a new programming model for MapReduce that supports tagging of sensitive data (Section 3.3). We then presented several scheduling modes (Section 3.4) that can assign the map and reduce tasks between the public and the private cloud for increased efficiency and reduced cost. We also proposed a general security framework to analyze and compare the information leakage by different schedulers (Section 3.5). A prototype has been implemented on top of Hadoop (Section 3.6), with experiments on a hybrid Amazon cloud to demonstrate its efficiency in terms of inter-cloud network traffic and task completion time (Section 3.7).

- We dealt with partitioning and scheduling of video processing operations in the domain of video surveillance (Chapter 4). First, we presented a well-designed stream processing model that is suitable for hybrid cloud video surveillance (Section 4.3).

Based on this model, we formulated the scheduling issue as an optimization problem to minimize the monetary cost, with several constraints being satisfied (Section 4.4), and gave an efficient solution using a simple observation and a heuristic (Section 4.5). We conducted experiments through both simulations and proof-of-concept system runs to demonstrate the efficiency and effectiveness of the proposed approach (Section 4.6).

Organization

Chapter 2 provides the background of cloud computing, together with a brief summary of existing work on cloud computing security. Chapter 3 details the design, implementation and evaluation of the proposed tagged-MapReduce extension. We continue in Chapter 4 by looking at the problem of processing large-scale video surveillance streams on hybrid clouds. Chapter 5 concludes the thesis, with several suggestions for the future directions.

Chapter 2

Background

This chapter provides a brief overview of cloud computing, with emphasis on the hybrid cloud model. We also summarize existing research work on privacy-preserving computing in the cloud.

2.1 Cloud Computing

A cloud might be thought of as a large pool of resources, unified through virtualization or job scheduling techniques, that can be managed to dynamically scale up to match the load, using a pay-as-you-use business model. The main idea behind cloud computing is not new: John McCarthy in the 1960s already envisioned that computing facilities will be provided as utilities to the general public [132]. It was until 2006 when Google's CEO Eric Schmidt used the word to describe their new business model that the term started to gain its real popularity. Yet for a long time, "cloud computing" is only used as a marketing term without any standards or formal definitions, causing ambiguities and confusions. There are a few attempts to standardize the notion [124, 163]. In this thesis, we adopt the definition by the U.S. National Institute of Standards and Technology (NIST) [124]:

> **NIST's definition of cloud computing**. *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of con-*

figurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The above definition captures well the essential characteristics of cloud computing which include:

- *On-demand self-service*. Users can request and manage resources without human interaction with the service providers, using, for example, a web portal or management interface. Provisioning and de-provisioning of resources happen automatically on the providers' side.
- *Broad network access*. Clouds are generally accessible via the Internet, and use the Internet as the service delivery medium. Thus, any device with Internet connectivity, e.g., a smartphone, a PDA or a laptop, is able to access the cloud services.
- *Shared resource pooling*. Computing resource such as CPUs, memories and storage are implemented as a homogeneous architecture that is shared among all users.
- *Rapid elasticity*. Resources can be allocated and released rapidly and elastically. This will allow the users to scale up the resources at any time to address peak workloads and usage, and then scale down by returning the resources to the pool when finished.
- *Metered service*. Computing in the cloud is offered as utility where users only pay for what they have used, like any other utility enterprises paid for such as electricity and gas.

2.1.1 Service Models

The architecture of a cloud computing environment can be broadly divided into 4 layers: the hardware layer, the infrastructure layer, the platform layer and the application layer, as

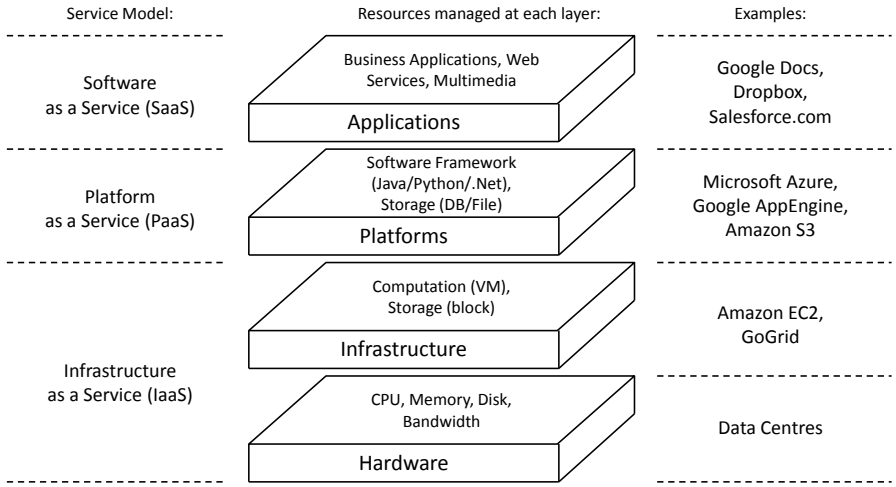


Figure 2.1: The layered architecture of a cloud computing environment.

shown in Figure 2.1. Corresponding to this classification, services offered by the clouds can be grouped into 3 categories: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

- *Infrastructure as a Service (IaaS)*. In this model infrastructure resources are provided, usually in terms of virtual machines (VMs), to cloud users. Users have access to and can manage the computing power, storage mediums and necessary network components. Users thus can run arbitrary operating systems and softwares that best meet their needs, with full control and management. An example of IaaS would be Amazon EC2 [24].

- *Platform as a Service (PaaS)*. In this model platform layer resources including operating system support and software development frameworks are provided, hence users can create, deploy and run custom applications targeting specific platforms, with full control of the applications and their configurations. Examples of PaaS would be Google Apps Engine [10] and Microsoft Azure Platform [12].

- *Software as a Service (SaaS)*. In this model on-demand software and applications are provided over the Internet, thus users can rent the software using pay-per-use, or subscription fee. Examples of SaaS would be Dropbox [19] and Salesforce.com Customer Relationship Management (CRM) software [22].

Note that it is entirely possible for a SaaS or PaaS provider to run its cloud on top of a IaaS provider. For example, Dropbox, an online file hosting and sharing service, stores customers' data on Amazon S3. There are also cases where SaaS/PaaS and IaaS are parts of the same organization, e.g., Google and Salesforce.com.

2.1.2 Cloud Advantages

Cloud has advantages in offering more scalable, fault-tolerant services with even higher performance and lower cost. More specifically, from customers' point of view, cloud computing offers a wide range of benefits including:

- *No up-front investment*. Cloud resources are provided in a pay-as-you-go pricing model. Users do not have to invest in any infrastructure or hardware (e.g., plants, computers, networks, etc.) in order to start their business. They can simply rent resources from the cloud based on their needs and only pay for the usage.

- *High scalability and elasticity*. The cloud provides a seemingly infinite set of resources which can be allocated and de-allocated on-demand. Users can easily expand their applications to large scales in order to handle rapid increase in service demand. They do not need to provision capabilities according peak workloads, and thus save a significant amount of monetary cost and increase the resource utilization rate.

- *Low operating and maintenance cost*. By deploying services and applications in the cloud, users can free themselves from complex tasks of operating and maintaining infrastructure and policies. Furthermore, they also shift business risks (e.g., hardware failures) to the cloud providers which usually have better expertise and are better equipped for managing these risks.

- *High accessibility*. Services hosted in the cloud are generally web-based. Thus, they are easily accessible through a variety of devices with Internet connectivity.

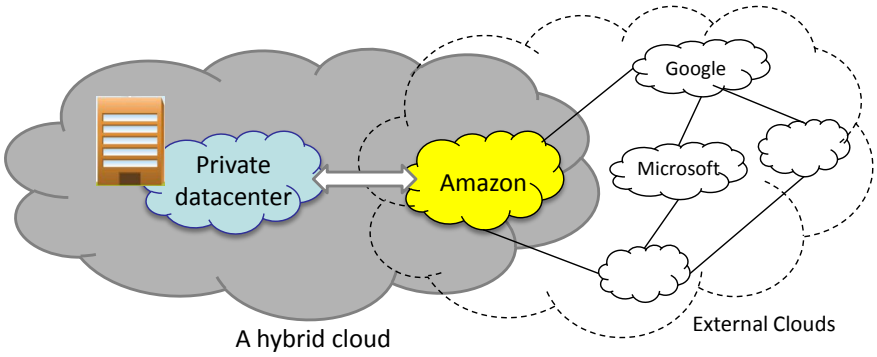


Figure 2.2: Illustration of a hybrid cloud.

These devices include not only desktop and laptop computers, but also smartphones and PDAs.

These features make cloud computing a compelling model for developing and deploying new services and applications, especially for small and medium business (SMBs) and individuals.

2.2 Hybrid Clouds

2.2.1 Definition and Current Status

Cloud computing comes mainly in three forms: public clouds, private clouds, and hybrid clouds. A *public cloud* makes resources, such as compute and applications, available to the general public. Public clouds provide the best economies of scale but lack fine-grained control over the data and applications. A *private cloud* is a data center owned by a single organization. The goal of a private cloud is not to provide services to external customers but instead to gain the benefits of cloud architecture without giving up the full control. Private clouds can be expensive with typically modest economies of scale, and are driven by concerns around security and compliance.

A *hybrid cloud* is an integrated cloud service utilizing both public and private clouds to perform distinct functions within the same organization. Applying the definition from the NIST, "a hybrid cloud is a combination of public and private clouds bound together

by either standardized or proprietary technology that enables data and application portability". Hybrid cloud models can be implemented as a combination of a private cloud inside an organization (i.e., the local in-house datacenter) or a private cloud hosted on third-party premises, together with one or more public cloud providers. A hybrid cloud is illustrated in Figure 2.2.

A hybrid cloud offers advantages of both the public and private clouds. On the one hand, a hybrid cloud can handle typical workload on the private cloud while offloading additional workload to the public cloud, thereby is more scalable and cost-effective than private clouds. On the other hand, it allows sensitive and business-critical data to be managed and processed locally, and hence is arguably more secure than public clouds. More specifically, a hybrid cloud can offer its users the following features:

- *Scalability*. While private clouds do offer a certain level of scalability depending on their configurations (whether they are hosted internally or externally for example), public cloud services will offer scalability with fewer boundaries because resource is pulled from the larger cloud infrastructure. By moving as many non-sensitive functions as possible to the public cloud, it allows an organization to benefit from public cloud scalability while reducing the demands on a private cloud.

- *Cost-efficiency*. Public clouds are likely to offer more significant economy of scale (such as centralized management) and so greater cost efficiency, than private clouds. Hybrid clouds therefore allow organizations to access these savings for as many business functions as possible while still keeping sensitive operations secure.

- *Security*. The private cloud of the hybrid cloud model not only provides the security where it is needed for sensitive operations but can also satisfy regulatory requirements for data handling and storage where it is applicable.

- *Flexibility*. The availability of both secure resource and scalable cost-effective public resource can provide organizations with more opportunities to explore different operational avenues.

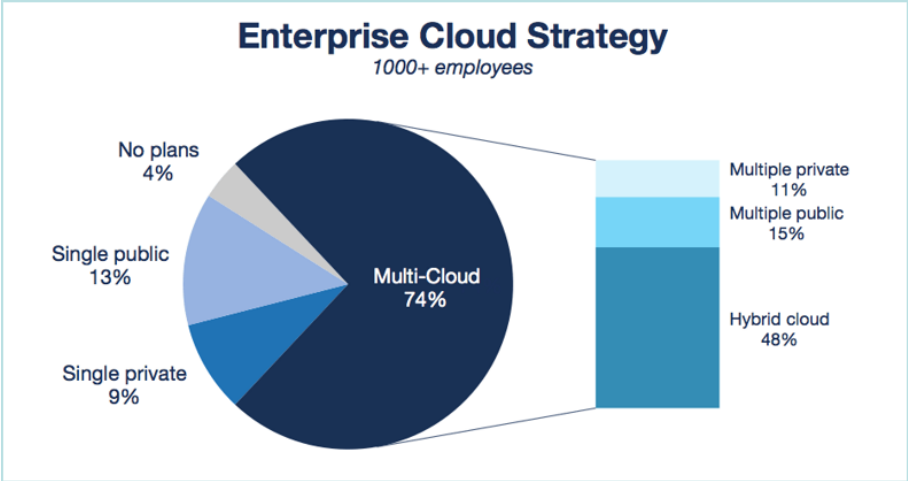


Figure 2.3: RightScale 2014 State of the Cloud Report [5].

A large number of recent surveys reveal the popularity and growing demand of hybrid clouds [1, 3–5, 15]. For example, the RightScale 2014 State of the Cloud Report shows that the hybrid cloud has accounted for around 50% of all the cloud adoptions in 2013 (as shown in Figure 2.3) and this number is still expected to be increasing. Rackspace's 2013 Cloud Survey [15] gives us more details about industry's attitude on hybrid clouds:

- 60% of respondents have moved or are considering moving certain applications or workloads either partially (41%) or completely (19%) off the public cloud because of its limitations or the potential benefits of other platforms, such as the hybrid cloud;
- 60% of IT decision-makers see hybrid cloud as the culmination of their cloud journey;
- Top reasons for using hybrid cloud instead of a public cloud only approach: better security (52%), more control (42%), and better performance or reliability (37%);
- Top benefits hybrid cloud users report: more control (59%), better security (54%), better reliability (48%), reduced costs (46%) and better performance (44%);
- Average reduction in overall cloud costs from using hybrid clouds: 17%.

2.2.2 Scheduling on Hybrid Clouds

The hybrid cloud possesses certain characteristics making it different from the pure public or private cloud. In particular, while servers within each single public or private cloud are often connected by a high-bandwidth, low-latency network (for example, Gigabit LANs), connections across the public and private servers in a hybrid cloud have to go through a wide area network (WAN) or the Internet, having relatively smaller bandwidth and higher delay. In addition, under current typical cloud pricing models, data traffic within each single cloud is free-of-charge whereas data traffic out from/in to the public cloud may incur high monetary cost. For example, Amazon does not charge for data transfer in the same Availability Zone within the Amazon AWS, but charges as high as $0.19 per GB for data transfer out from Amazon EC2 to the Internet.[2] Based on these observations, it is therefore desired to carefully schedule the computation so as to reduce the inter-cloud data traffic as well as the monetary cost. With the advances in hybrid clouds, this scheduling issue has drawn growing research interest. For example, Zhang et al. [174] propose a hybrid cloud computing model for Internet-based applications with highly dynamic workload, and augment this model with a workload factoring service. The core technology is a fast "hot" data prediction algorithm. Van et al. [162] propose a scheduling algorithm to minimize the cost in a multi-provider hybrid cloud setting with deadline-constrained and preemptible workloads that are characterized by memory, CPU and data transmission requirements. De et al. [73] and Mattess et al. [121] similarly evaluate the cost-benefits of different strategies for scheduling workloads between a local cluster and a public cloud. However, none of these works takes into account the data security and privacy requirement. In the following Section 2.3.3, more works considering both security and efficiency in hybrid cloud scheduling will be discussed.

[2]Prices are taken from the Asia Pacific (Singapore) region in May 2014, with a total amount of data transfer less than 10 TB per month.

2.3 Secure Computing on the Cloud

Security and privacy are often the first concern when organizations consider outsourcing their data to a public cloud [1, 3, 5, 67]. In public cloud environments, data is usually located outside an organization's network, so that users have access to resources but not to physical machines, network and other related equipment. Users have to rely on the cloud providers for ensuring data security and privacy, which may not be a good practice. On the one hand, public cloud services cannot be fully trusted due to potentially malicious insiders [68, 102]. On the other hand, public clouds may also suffer from outside attacks. For example, confidential information can be extracted through side-channel information leakage across VMs resided on the same physical machine [141]. Snowden recently revealed that NSA secretly tapped into Yahoo! and Google data centers to collect sensitive information [30]. Therefore, how to compute in public clouds without revealing sensitive information is a challenging problem in general. In this section, we summarize existing approaches and broadly divide them into three categories: encrypted domain processing, trusted platforms and data segregation using hybrid clouds.

2.3.1 Encrypted Domain Processing

One simple approach is to employ client-side encryption before pushing data to the cloud. However, traditional encryption techniques such as AES do not allow computation to be carried out on the encrypted data. Cloud users have to download the data, decrypt it and then process it which is extremely inefficient. In response, multiple cryptographic techniques have been proposed to support encrypted domain processing.

Homomorphic encryption allows one to compute on encrypted data without getting the underlying plaintext information. Early homomorphic encryption schemes are restricted to specific operations such as multiplications for RSA [142], additions for Paillier [131], or additions and up to one multiplication [46]. They only support outsourcing of specific computations, e.g., modular exponentiations [92], linear algebra [38], sequential

comparison [39] and DNA searching [42], to untrusted servers. Gentry in 2009 introduced the notion of Fully Homomorphic Encryption [84] which supports arbitrary computations on the encrypted data. Gennaro et al. [82] then present an idea to securely outsource general computations using fully homomorphic encryption in such a way that both input/output privacy and correctness/soundness of computation are guaranteed. Following their work, Chung et al. [66] propose an improved version in which the original inefficient "offline stage" is significantly simplified. Unfortunately, fully homomorphic encryption schemes are currently not efficient enough for practical usages [83, 151], though various improvements and implementations have been made over the years [49, 50, 85, 155].

There are also works focusing on encrypted domain searching specifically. The notion of searchable encryption was first studied by D. Song et al. [158] in the symmetric setting, and then improved and revised by Chang et al. [59] and Curtmola et al. [71]. Wang et al. [166] later give a secure keyword ranked search scheme which utilizes keyword frequency to rank searching results instead of returning undifferentiated results. Boneh et al. [44] give the first searchable encryption construction in the public key setting. These schemes only support searching over a single keyword. To enrich the search functionality, conjunctive keyword search [41, 45, 88] over encrypted data are then proposed. Predicate encryption schemes [105, 154], as a more general search approach, are published recently which support both conjunctive and disjunctive keyword search. To improve searching experience, Cao et al. [54] then propose the first multi-keyword ranked search in which searching results are ordered by "coordinate matching", as an improvement to their early work [166] which only considers single keyword. Kamara et al. [99] introduce a cryptographic cloud storage service that, by combining techniques of searchable encryption, attribute-based encryption and proof of storage, enables the cloud to search on the encrypted data without leaking its information while allowing users to verify the integrity of the data at any time.

2.3.2 Trusted Computing and Secure Hardware

Another line of works try to address the cloud security and privacy issues by establishing trusted execution environments where cloud clients can verify the integrity of the software and hardware platforms. The use of trusted computing-based remote attestation in the cloud scenario was recently discussed [65]. Trusted Virtual Domains [53, 90] are one approach that combines trusted computing, secure hypervisors, and policy enforcement of information flow within and between domains of virtual machines. However, those approaches require trust in a non-negligible amount of hardware (e.g., CPU, Trusted Platform Module (TPM) [17]) which are under physical control of the cloud provider. A virtualized TPM [134] that is executed in software could be enhanced with additional functionality (see, e.g., [146]). However, such software running on the CPU has access to unencrypted data at some point, hence, if the cloud provider is malicious, confidentiality and verifiability cannot be guaranteed by using trusted computing.

Secure co-processors [78, 156] are tamper-proof active programmable devices that are attached to an untrusted computer in order to perform security-critical operations or to allow establishing a trusted channel through untrusted networks and hardware devices to a trusted software program running inside the secure coprocessor. This can be used to protect sensitive computation from insider attacks at the cloud provider [97]. However, as secure hardware is usually expensive, relatively slow, and provides only a limited amount of secure memory and storage, it does not qualify as building blocks for a cost-efficient, high-performant, and scalable cloud computing infrastructure.

2.3.3 Data Segregation Using Hybrid Clouds

In view of the difficulties of the above cryptographic approaches or trusted platforms, the academic and research community show growing interest in the data segregation approach over hybrid clouds. Ideally, with a hybrid cloud, an organization can keep sensitive and private data in the private cloud which is under full control of the organization

while pushing non-sensitive data to the elastic public cloud, addressing the security issues by preventing sensitive information flowing into the public cloud. But this data segregation model is not well supported by many of today's data-intensive computing frameworks. Ko et al. present the HybrEx model [108] which discusses various ways to partition data and MapReduce computation over a hybrid cloud. Four execution models are presented accordingly, that is, map hybrid, horizontal partitioning, vertical partitioning and hybrid. However, they only give an outline without further details or implementations. Sedic [175] gives a practical implementation of the map hybrid model on top of Hadoop [18]. However, Sedic has limitations in terms of flexibility. The reduce can only happen in the private cloud while not utilizing the public cloud resources. Also, Sedic does not naturally support complex MapReduce computation involving chained or iterative MapReduce which is important to many real-world applications. Bugiel et al. [52] propose using the private cloud to encrypt the data and verify the intensive computation performed in the untrusted public cloud.

On the issue of secure query processing, Relational Cloud [70] uses a graph-based partitioning algorithm to achieve near-linear elastic scale-out, and an adjustable encryption scheme that encrypts each value in a "onion". Query operations can be performed by decrypting the value only to an appropriate layer, achieving both privacy and efficiency. Oktay et al. [130] formulate the database partitioning over hybrid clouds as an optimization problem with a set of performance, cost and disclosure constraints, and give an efficient greedy algorithm. They simply measure the data disclosure risk as how much percent of the sensitive data can be stored on the public cloud. In contrast, Aggarwal et al. [34] investigate how to achieve information-theoretically secure partitioning by decomposing database relation schemas across two non-colluding cloud providers (not necessary to be a public and a private cloud). Other works include distributing human genomic computation to hybrid clouds so as to protect sensitive DNA information [62, 168].

Chapter 3

Privacy-preserving MapReduce Computation on Hybrid Clouds

3.1 Introduction

In this chapter, we consider the MapReduce framework and present our extension which supports privacy-preserving MapReduce computation on hybrid clouds.

As mentioned in the Introduction, a simple solution for secure computing on hybrid clouds is to separate the data into sensitive and non-sensitive parts, outsource non-sensitive data to the public cloud while keeping sensitive data being handled on the private cloud. In this way, the data can be processed both securely while being elastic. However, this data segregation model under hybrid clouds is not well supported by MapReduce [74], today's most popular data-intensive computing framework. MapReduce is designed for only one (logical) cloud and does not distinguish between data and servers with differing sensitivities. Hence, cloud users have to manually split the data, compute them on each cloud separately and then combine the results using an additional code. This is highly inefficient and also contrary to the transparency provided by MapReduce. Our objective is an automatic and general framework to facilitate secure MapReduce computation on hybrid clouds.

Sedic [175] addresses this problem to some extent by pre-labeling the input data which is then replicated to both the public and private clouds, but with sensitive portions in the public cloud "sanitized". During computation, map tasks operate in both clouds and send all intermediate results to the private cloud for reducing to prevent data leakage from the intermediate results. However, the sanitization approach taken by Sedic has limitations in terms of flexibility - it does not fit well with complex MapReduce jobs such as chained or iterative MapReduce, which is important to many data analytical tasks and realistic applications [140]. In addition, the sanitization approach may still reveal relative locations and length of sensitive data, which could lead to crucial information leakage in certain applications [118]. A more generic, flexible and secure framework is desired.

In response, we propose a conservative extension to MapReduce that deals automatically with mixed-sensitivity data in hybrid clouds and supports a new MapReduce programming model where data sensitivity can be manipulated during computation, e.g., security-aware programs can be used to downgrade the sensitivity of data in execution. We propose tagged-MapReduce that (conceptually) augments each key-value pair in MapReduce with a sensitivity tag, extending the map and reduce functions appropriately. The tagging helps to achieve the following goals: 1) it enables fine-grained dataflow control during execution to prevent leakage and supports scheduling of map and reduce tasks in the two clouds; 2) it allows programmers to code sophisticated security policies to guide sensitivity transformation during execution and supports sensitivity downgrading which is useful in sensitivity-aware applications; 3) it provides sensitivity information for data across multiple MapReduce jobs which is necessary for complex MapReduce computations with chained jobs. The flexibility also allows legacy MapReduce programs to be supported by simply having a default tagging policy. Sedic programs can be expressed as a special class of tagged-MapReduce programs; however, Sedic cannot express all tagged-MapReduce programs.

The concerns of preventing data leakage mean that there is a security constraint on where computations can be run and where data can be sent in a hybrid cloud computing

job. We provide scheduling strategies for reduce tasks so that some reducers can execute in the public cloud. The scheduling strategies exploit useful properties of common map and reduce functions to rearrange the computation for more effective load-balancing and inter-cloud network usage while maintaining MapReduce correctness. For example, if a reduce operation is "partitionable", tagged-MapReduce will automatically carry out partial reduce computation on the public cloud (with non-sensitive data), which lessens not only the private cloud's workload but also the total amount of inter-cloud data traffic. Our prototype implementation allows the properties to be easily coded into the tagged-MapReduce programs, from which the scheduler decides automatically which scheduling strategy is to be employed.

Nevertheless, special care must be taken in designing such scheduling strategies as different strategies lead to different actual dataflows during execution, which in turn leads to different amounts and types of information being exposed to the public cloud. In particular, a scheduler that aggressively rearranges the computation to the public cloud, while improving efficiency and maintaining MapReduce correctness, may leak more information than a "conservative" scheduler that carries out all reduce computation in the private cloud. Such leakage is beyond the programmers' anticipation and could be unacceptable in some scenarios. To analyze the scheduling strategies, we propose the first security model that captures how dataflow can leak information during execution. This model is suitable for analyzing what additional leakage a scheduler might make through execution on a hybrid cloud over a reference "baseline" scheduler whose information exposure is deemed to be acceptable. Using this model, we are able to show that some potentially more effective scheduling strategies indeed leak more information than the baseline whereas ours do not.

We implement a prototype of tagged-MapReduce on Hadoop, with experiments to evaluate the practicability of the system and the effectiveness of the proposed scheduling strategies. The experiments are run on a small-sized hybrid cloud built on Amazon EC2, using both single and chained MapReduce jobs. The results show that the tagged-

MapReduce prototype which implements the security constraints for preventing data leakage is able to automatically and efficiently outsource computation to the public cloud and reduce inter-cloud data traffic. The system is practical with only small overhead compared to the baseline Hadoop which ignores the data confidentiality and security constraints.

3.2 Overview

3.2.1 MapReduce

MapReduce is a framework for performing distributed computation across huge datasets over a large cluster of commodity machines. The MapReduce framework was originally developed at Google [74], but has recently seen wide adoptions and has become the *de facto* standard for large-scale data analysis. Publicly available statistics indicate that MapReduce is used to process more than 20 petabytes of information per day at Google alone [129]. Over 70 companies use MapReduce including Yahoo!, Facebook, Adobe, and IBM [2]. In addition, many universities (including CMU, Cornell etc.) are providing MapReduce clusters for research [2].

MapReduce Basics

In the MapReduce framework, the basic unit of information is a $\langle key, value \rangle$ pair where each key and each value are binary strings. The input to any MapReduce algorithm is a set of $\langle key, value \rangle$ pairs. Users provide two functions: a *map* function and a *reduce* function. A map function takes as input a single $\langle key, value \rangle$ pair, and produces as output any number of new $\langle key, value \rangle$ pairs. It is crucial that the map operation is stateless – that is, it operates on one pair at a time. This allows for easy parallelization as different inputs for the map can be processed by different machines.

A reduce function takes all of the values associated with a single key k, aggregates them and outputs a possibly smaller multiset of $\langle key, value \rangle$ pairs with the same key k. Typically just zero or one output pair is produced per reduce invocation. This highlights

one of the sequential aspects of MapReduce computation: all of the maps need to finish before the reduce stage can begin.

Between map and reduce, there is a *shuffling* stage whereby the underlying system that implements MapReduce sends all of the values that are associated with an individual key to the same machine. This occurs automatically, and is seamless to the programmer. More specifically, programmers only need to specify the two map and reduce functions, while the MapReduce framework handles the complicated tasks of scheduling and data movement during execution, providing high scalability and fault-tolerance. Thus, it is easier for programmers, even with no experience in parallel/distributed systems, to write programs working on large clusters.

Formal Definition

We now give a more formal definition of the MapReduce programming model. As mentioned above, the fundamental unit of data in MapReduce computation is the $\langle key, value \rangle$ pair, where keys and values are binary strings.

DEFINITION 3.1 *A* mapper μ *takes as input an ordered* $\langle k, v \rangle$ *pair with* r *as the auxiliary bits for randomness,*[3] *outputs a finite multiset of new pairs* $\{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle, \ldots, \langle k_m, v_m \rangle\}$ *for some* m*, i.e.,*

$$\mu(\langle k, v \rangle) \rightarrow \{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle, \ldots, \langle k_m, v_m \rangle\}$$

DEFINITION 3.2 *A* reducer ρ *takes as input a multiset of pairs* $\{\langle k, v_1 \rangle, \langle k, v_2 \rangle, \ldots, \langle k, v_n \rangle\}$ *of the same key* k *with* r *as the random bits, outputs a new multiset of pairs* $\{\langle k, w_1 \rangle, \langle k, w_2 \rangle, \ldots, \langle k, w_{n'} \rangle\}$ *for some* n *and* n'*, i.e.,*

$$\rho(\{\langle k, v_1 \rangle, \ldots, \langle k, v_n \rangle\}) \rightarrow \{\langle k, w_1 \rangle, \ldots, \langle k, w_{n'} \rangle\}$$

[3] As the map function can be probabilistic, the string r provides the randomness. r can be removed if μ is deterministic.

One simple consequence of the above two definitions is that mappers can manipulate keys arbitrarily, but reducers cannot change the keys.[4]

Next we describe how the system executes MapReduce computations. A MapReduce program consists of a sequence $\langle \mu_1, \rho_1, \mu_2, \rho_2, \ldots, \mu_R, \rho_R \rangle$ of mappers and reducers. The input is a multiset of $\langle key, value \rangle$ pairs denoted by U_0. To execute the program on input U_0:

For $r = 1, 2, \ldots, R$, do:

- EXECUTE MAP: Feed each $\langle k, v \rangle$ in U_{r-1} to mapper μ_r, and run it. The mapper will generate a sequence of tuples, $\{\langle k_1, v_1 \rangle, \langle k_2, v_2 \rangle, \ldots, \langle k_m, v_m \rangle\}$ for some m. Let U'_r be the multiset of intermediate $\langle key, value \rangle$ pairs output by μ_r, that is, $U'_r = \cup_{\langle k,v \rangle \in U_{r-1}} \mu_r(\langle k, v \rangle)$.

- SHUFFLE: For each k, let $V_{k,r}$ be the multiset of values v_i such that $\langle k, v_i \rangle \in U'_r$. The underlying MapReduce implementation constructs the multiset $V_{k,r}$ from U'_r.

- EXECUTE REDUCE: For each k, feed k and some arbitrary permutation of $V_{k,r}$ to a separate instance of reducer ρ_r, and run it. The reducer will generate a sequence of tuples $\{\langle k, v'_1 \rangle, \langle k, v'_2 \rangle, \ldots, \langle k, v'_{n'} \rangle\}$ for some n'. Let U_r be the multiset of pairs generated by ρ_r, that is, $U_r = \cup_k \rho_r(\langle k, V_{k,r} \rangle)$.

The computation halts after the last reducer, ρ_R, halts.

As stated before, the main benefit of this programming model is the ease of parallelization. Since each mapper μ_r only operates on one tuple at a time, the system can have many instances of μ_r operating on different tuples in U_{r-1} in parallel. After the map step, the system partitions the set of intermediate tuples output by various instances of μ_r based on their key. That is, part i of the partition has all $\langle key, value \rangle$ pairs that have key k_i. Since reducer ρ_r only operates on one part of this partition, the system can have many instances of ρ_r running on different parts in parallel.

[4]In this thesis, we adopt the definition given by Karloff et al. [104] whereby the key in the reduce output must be identical to the key in its input. However, in the original MapReduce paper by Dean et al. [74] they do not have such restriction and simply ignore the keys in the reduce output. In actual MapReduce implementations such as Hadoop, reducers can also output keys different from those in the input.

Applications in Security Domains

By virtue of its simplicity, scalability, and fault-tolerance, MapReduce is becoming ubiquitous, gaining significant momentum from both industry and academia. However, MapReduce has inherent limitations on its performance and efficiency. A large number of variants and improvements have been proposed over the years, including high-level languages (e.g., SQL) support like Microsoft SCOPE [56] and Apache Hive [161], loop programs support like HaLoop [51] and Twister [79], I/O optimizations [75, 138], improved scheduling algorithms [61, 91, 119], automatic performance tuning [96] and etc. Most of these works are performance-centric.

Depending on the applications and infrastructure, there could also be security and privacy requirements which cannot be met by the current MapReduce framework. Hence, several studies have endeavored to augment MapReduce with certain security features. HybrEx [108] and Sedic [175] are two examples that extend MapReduce to support privacy-aware computation on hybrid clouds. Airavat [144] integrates decentralized information flow control (DIFC) [110] and differential privacy [77] into MapReduce to provide rigorous privacy and security control in the computation for individual data. Mohan et al. then present GUPT [126], an improvement to Airavat, that can automatically learn and distribute the differentially private parameters. In addition, Xiao et al. propose Accountable MapReduce [171] which allows detecting of malicious nodes which generate inaccurate results through an A-test on every node in the system. Huang et al. study the similar problem of detecting cheating services under the MapReduce environment but based on techniques of watermark injection and random sampling [93].

There are other works employing the MapReduce framework for more efficient cryptographic operations. Mayberry et al. propose PIRMAP [122], a Private Information Retrieval (PIR) protocol that allow for optimal parallel computation during the "Map" phase of MapReduce, and homomorphic aggregation in the "Reduce" phase. Blass et al. present PRISM [43] that transforms the problem of word search into a set of parallel instances of PIR on small datasets, which can be efficiently computed in MapReduce.

Kamara et al. show how to construct parallel homomorphic encryption (PHE) schemes that can support various MapReduce operations on encrypted datasets, including element testing and keyword search [100]. Li et al. formulate the problem of outsourcing ABE to cloud service providers to relieve local computation burden, and propose a construction based on MapReduce [115]. Francois et al. show an interesting work of using MapReduce to detect botnets [81].

3.2.2 Overview of the Proposed Framework

MapReduce is not designed for processing a mix of sensitive and non-sensitive data in the hybrid cloud, as data can flow freely between all nodes in the two clouds, increasing the risk of information leakage. To prevent such leakage, we propose to extend MapReduce by explicitly tagging each key-value pair as either sensitive or non-sensitive. The tags serve as auxiliary information for the system to move data during execution, ensuring that sensitive tuples never leave the private cloud. We propose *tagged-MapReduce*, as shown in Figure 3.1, which involves: (1) a *scheduler* in the private cloud that schedules map and reduce tasks to workers and controls the flow of intermediate data with respect to their security tags, and (2) multiple *workers* across the public and private clouds that carry out the assigned tasks.

Tagged-MapReduce programs are similar to MapReduce programs, the difference being that a *programmer* can program in the map and reduce routines various policies that guide how the sensitivity should be changed during execution. For instance, in the classic word-count example that reads in text files and counts how often each word occurs, one can code in the map routine that: a tuple, i.e. $\langle word, 1\rangle$ is output as sensitive iff $word$ is from a sensitive input file and not in the set of "stop words". The logic for deciding the sensitivity of the output tuples is broadly called the *sensitivity policy* (in Section 3.3). Figure 3.2 illustrates how sensitivity policies can be programmed using the word-count example with the aforementioned security policy.

To perform a computation over tagged-MapReduce, the input data have to be tagged

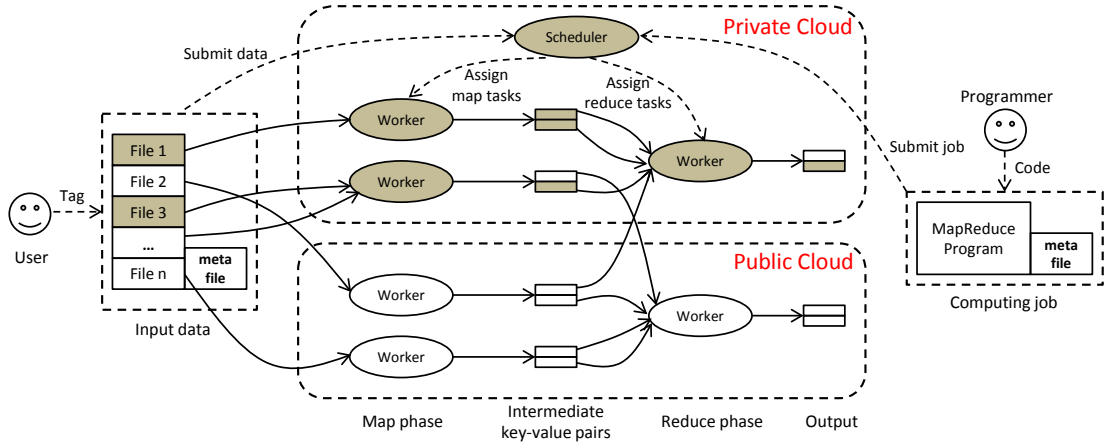


Figure 3.1: Overview of tagged-MapReduce from the perspective of users and programmers. Shaded rectangles are files/tuples marked as sensitive, shaded ellipses are workers/-scheduler run in the private cloud. Note that the output tuples carry sensitivity information which can be fed to the next job, thus multiple MapReduce computation can be naturally supported.

first by indicating the sensitive portions. As manual tagging or individual tuple-level tagging can be tedious, for simplicity and usability, our prototype implementation considers *file-level tagging*,[5] i.e., the input data consists of multiple files and each file contains either all sensitive data or all non-sensitive data. The sensitivity of input files is specified in a meta file which is then uploaded to the framework together with the input data. The underlying distributed file system then starts to replicate the data in a privacy-aware way, ensuring that sensitive files are only stored in the private cloud.

In addition, the programmer can also specify certain *properties* that the map or reduce function meets. The two properties we particularly looked into are *partition-able reduce* and *unique tag* (in Section 3.4). Roughly speaking, a reduce function is partition-able if it can be carried out in a "divide-and-conquer" manner; a map function meets the unique tag property if each key in its output is tagged as either sensitive or non-sensitive, but never both. These properties can be directly coded into users' tagged-MapReduce programs using an additional API provided by us. Such information helps the system to decide how to schedule the tasks using an appropriate *scheduling mode* (in Section 3.4). We have

[5]File-level tagging is simple and yet does not lose generality as more sophisticated tagging, e.g., tuple-level tagging, can be simply done by having an initial tagged-MapReduce job with all input files being tagged sensitive and output tuples with the desired sensitivities.

provided four scheduling modes that allow the reduce computation to be redistributed across the hybrid cloud for improved efficiency and reduced performance overhead.

3.3 Programming Model

Original MapReduce has map and reduce functions operating on key-value tuples. Our tagged-MapReduce framework extends the programming model of MapReduce to support tags with the corresponding functions *tagged-map* and *tagged-reduce* operating on *tagged-tuples*. Specifically, we extend the key-value pair $\langle k, v\rangle$ in MapReduce where k and v are binary strings with tagged-tuples of the form $\langle k, v; t\rangle$, where t is a symbol *sensitive* or *non-sensitive*,[6] and k and v are as in MapReduce.

A *tagged-map* $\hat{\mu}$ extends a given map μ in the original MapReduce framework. If μ on input $\langle k, v\rangle$ with a random string r as the auxiliary bits for randomness, outputs a finite multiset

$$\{\langle k_1, v_1\rangle, \ldots, \langle k_m, v_m\rangle\}$$

for some m, then the corresponding tagged-map $\hat{\mu}$ is a function that on input $\langle k, v; t\rangle$ and with r as the auxiliary data, outputs

$$\{\langle k_1, v_1; t_1\rangle, \ldots, \langle k_m, v_m; t_m\rangle\}$$

for some tags $t, t_1, \ldots, t_m$.

Similarly, a *tagged-reduce* $\hat{\rho}$ extends a given reduce ρ. If ρ on input a multiset $\{\langle k, v_1\rangle, \langle k, v_2\rangle, \ldots, \langle k, v_n\rangle\}$ with random string r as the auxiliary bits, outputs a multiset of pairs

$$\{\langle k, w_1\rangle, \ldots, \langle k, w_{n'}\rangle\}$$

for some n and n', then the tagged-reduce $\hat{\rho}$ is a function that on input $\{\langle k, v_1; t_1\rangle, \langle k, v_2; t_2\rangle,$

[6]For simplicity, we use binary tags of sensitive and non-sensitive but a larger attribute set is possible.

$\ldots, \langle k, v_n; t_n \rangle\}$ with r as the auxiliary data, outputs the multiset

$$\{\langle k, w_1; t'_1 \rangle, \ldots, \langle k, w_{n'}; t'_{n'} \rangle\}$$

for some tags $t_1, t_2, \ldots, t_n, t'_1, t'_2, \ldots, t'_{n'}$.

The tags in tagged-tuples are just auxiliary data which do not affect the keys and values. Two different tagged-reduces $\hat{\rho}_1$ and $\hat{\rho}_2$ that extend the same reduce ρ but with different algorithms for deciding the output tags, will output the same distribution of keys and values.[7] Hence, our extension is conservative since the program will not be changed if all data are non-sensitive and it does not affect the output distribution. The role of the tags is to feed information to the scheduler which decides where computation is to be carried out. This segregation of roles provides clarity in coding programs to process sensitive data and in analyzing algorithms. Moreover, the tags also carry sensitivity information for data across multiple MapReduce jobs (see Section 3.7), and thus complex MapReduce computation with chained or iterative MapReduce can be supported naturally.

Figure 3.2 gives Hadoop Java code in our prototype of tagged-map for the extended `WordCount` job working on a set of sensitive and non-sensitive files (right). Compared to the original map (left), the difference is the extra statement (in the dashed box) to compute the sensitivity of each word based on some sensitivity rules (see below) and an API `setIsSensitive()` to set the tags of output tuples.

When it is clear from the context, we will omit the word "tagged" and call tagged-tuple, tagged-map and tagged-reduce as tuple, map and reduce respectively.

3.3.1 Sensitivity Policy

In addition to normal MapReduce programs (which do not have code for data sensitivity), with explicit tagging, programmers can now implement MapReduce programs which are sensitivity-aware, applying a policy to govern the sensitivity of tuples created during

[7] The overall MapReduce computation may be non-deterministic, hence we consider the possible outputs to be a distribution. In the case that they are deterministic, they always output the same key-value pairs.

```
public void map(LongWritable key, Text value,
                                Context context)
{
    // key: line number
    // value: content of this line
    String line = value.toString();
    StringTokenizer tokenizer =
                    new StringTokenizer(line);
    while (tokenizer.hasMoreTokens())
    {
        String val = tokenizer.nextToken();
        word.set(val);
        context.write(word, one);
    }
}
```

```
public void map(LongWritable key, Text value,
                                Context context)
{
    Boolean sensitive = false;
    String line = value.toString();
    StringTokenizer tokenizer =
                    new StringTokenizer(line);
    while (tokenizer.hasMoreTokens())
    {
        String val = tokenizer.nextToken();
        sensitive = context.getInputSplit().getSensitivity()
                    && !inStopWords(val);
        word.setIsSensitive(sensitive);
        word.set(val);
        context.write(word, one);
    }
}
```

Figure 3.2: Example code corresponding to original map (left) and tagged-map (right) for the `WordCount` job. The difference is the code within the dashed box that computes and sets the tags of the output tuples.

execution in the map and reduce routines. Such policies are broadly called the *sensitivity policies* in our framework. An example policy has the following rules: (1) A map $\hat{\mu}$ does not modify the sensitivity of the data, i.e., each tuple in the output of $\hat{\mu}$ has the same sensitivity as the input tuple; and (2) the output of the reduce is sensitive iff at least one input is sensitive. Our prototype uses this policy as the default if the program does not specify any policy. It is also how legacy (normal) MapReduce programs are supported. Programmers can choose to implement more sophisticated and application-specific policies to override the default policy. Figure 3.2 gives an example of programming sensitivity policies.

We now address the question of whether the sensitivity of an output tuple can be upgraded or downgraded. Let us first consider upgrading.

Non-upgrading policy

An upgrading happens if, for either a map or reduce, (all of) the input is non-sensitive but the output contains sensitive tuples. Assume that all map and reduce algorithms are public knowledge, the public servers can collude and all non-sensitive tuples are stored in the public servers, then it is meaningless to have a policy that deems the output as sensitive when all of the input data are non-sensitive, given that an adversary in the public cloud can compute the output anyway. This gives the following condition for tagged-map

$\hat{\mu}$ and tagged-reduce $\hat{\rho}$:

CONDITION 3.1 (NON-UPGRADING MAP AND REDUCE) *Consider map $\hat{\mu}(t)$, if input tuple t is non-sensitive then all tuples output from $\hat{\mu}$ will be non-sensitive. Similarly for reduce $\hat{\rho}(k, \{t_1, \ldots, t_n\})$, if all input tuples t_i's are non-sensitive, all tuples output from $\hat{\rho}$ will be non-sensitive.*

Violation of the above condition during execution does not compromise confidentiality of tuples previously tagged as sensitive, and thus may not be harmful in terms of security. Nevertheless, it tags data already known to the public cloud as sensitive, and hence imposes unnecessary constrains which in turn lowers the effectiveness of the scheduler.

Downgrading policy

However, there are situations where the sensitivity may be "downgraded" to non-sensitive, even if the input is sensitive. The downgrading can occur in either a map or reduce. For example, consider a tagged-map that takes in a surveillance video (tagged as sensitive), analyzes the video, and outputs a set of short video clips. Video clips with a low-level of activity are to be tagged as non-sensitive, whereas video clips with a high-level of activity are to be tagged as sensitive. Here, the final sensitivity is derived from both the key and value of the input, allowing certain video clips to be downgraded from sensitive to non-sensitive. Another application is data anonymization where a tagged-reduce takes as input a list of sensitive values and outputs an aggregated value. The output value is considered "anonymized" and thus tagged as non-sensitive. In general, downgrading allows to further push computation to the public cloud and is useful for applications where the input data are sensitive but only few of them turn out to be important after simple pre-processing. Explicit tagging makes such downgrading possible.

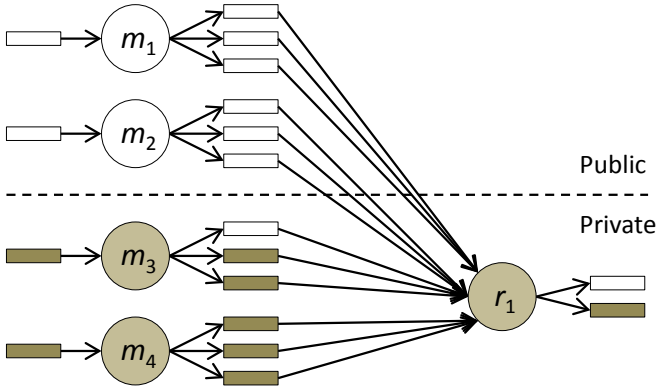


Figure 3.3: Default scheduling mode: Single-Phase (SP).

3.4 Scheduling Modes

After being tagged, input data are selectively distributed and replicated to the public and private clouds based on their sensitivity status with sensitive data placed in private nodes. Upon the data placement, a set of tagged-map tasks are then created, across the private and public clouds, to operate on the sensitive and non-sensitive data accordingly. After all tagged-map tasks have completed, a key can appear in tuples that are produced by both the public and private clouds with different sensitivities in different tuples. As a tagged-reduce task may receive both sensitive and non-sensitive tuples, it cannot be directly executed on the public cloud.

To prevent data leakage, a conservative scheduler might push all intermediate results produced in both clouds to the private cloud for reducing. This scheduling strategy is illustrated in Figure 3.3 which we call the *single-phase (SP) mode*. However, SP mode may overload the private servers (i.e., public servers are not enrolled in the reduce phase) and also lead to high volumes of data flowing from the public to the private cloud during MapReduce shuffling. Inter-cloud data traffic can be significant as inter-cloud bandwidth may be much smaller than the intra-cloud one (internally, within each cloud). Inter-cloud traffic may also be charged by the cloud provider.[8] What is desired is to outsource more reduce computation to the public cloud when needed while reducing the total amount of

[8]For example, Amazon does not charge for data transfer in the same Availability Zone within the Amazon AWS, but charges as high as $0.19 per GB for data transfer out from Amazon EC2 to the Internet.

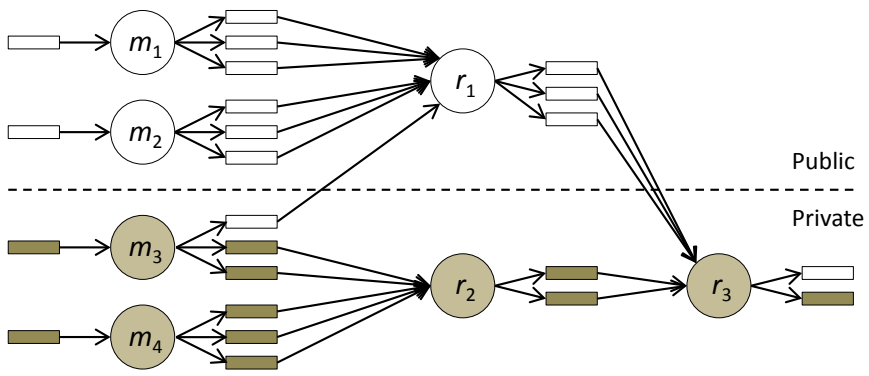


Figure 3.4: Two-Phase Crossing (TPC) mode.

inter-cloud data movement.

To this end, we investigate certain properties of the map and reduce functions which allow re-scheduling of the reduce computation on the two clouds for more effective load-balancing and reduced inter-cloud network traffic while maintaining MapReduce correctness. More specifically, we consider two properties: *partitionable reduce* and *unique tag* with three *scheduling modes*: *two-phase crossing* (TPC), *two-phase non-crossing* (TPNC) and *hand-off* (HO) modes, as follows.

3.4.1 Two-Phase Crossing Mode (Partitionable Reduce)

If a reduce function can be carried out in a "divide-and-conquer" manner, one could first enroll public workers to aggregate the non-sensitive tuples, and then combine them with the sensitive data. Let us first define the following form of distributive property on the reduce function which holds for many regular MapReduce programs:[9]

PROPERTY 3.1 (PARTITIONABLE REDUCE) *We say that a tagged-reduce $\hat{\rho}$ is partitionable if*

$$\hat{\rho}(k, L_1 \cup L_2) = \hat{\rho}(k, \hat{\rho}(k, L_1) \cup \hat{\rho}(k, L_2))$$

for all k, L_1 and L_2.

If $\hat{\rho}$ is partitionable, then it can be performed in two phases:

[9]This is the commutative and associative property that the `Combiner` function satisfies in MapReduce.

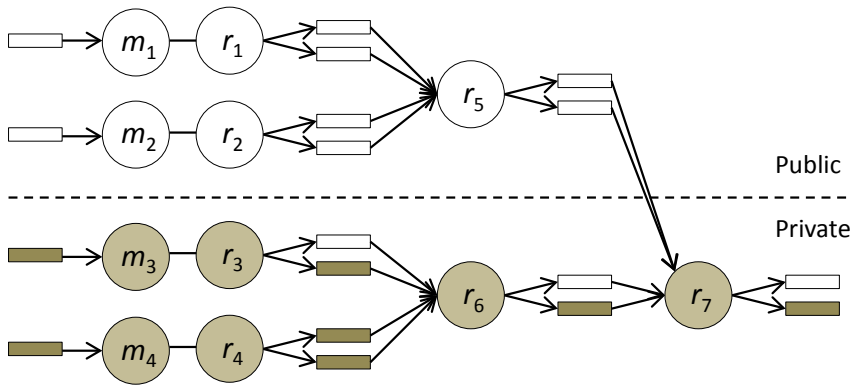


Figure 3.5: Two-Phase Non-Crossing (TPNC) mode.

1. (Phase 1) For each key k, a worker p_k (can be private or public) is selected and assigned to perform a reduce task on all non-sensitive tuples with k. A private worker q_k (possibly different) is selected and assigned to perform a reduce task on all sensitive-tuples with k.

2. (Phase 2) A private worker is selected and assigned to perform a reduce task on the output of p_k and q_k for each key k.

Figure 3.4 illustrates the above process. Since sensitivity can be downgraded, map tasks running on the private cloud may produce many non-sensitive tuples. This mode allows such tuples to be passed to the public cloud for partial reducing. In general, this mode leads to higher utilization of public servers but may incur increased dataflow from the private to the public cloud during shuffling.

3.4.2 Two-Phase Non-Crossing Mode

This mode is a potential improvement of the above two-phase crossing mode, whereby the reduce function is first applied on each map-task's output locally (like the `Combiner` in Hadoop). This local-reduce phase typically can reduce the size of the intermediate results, thus, speed up the internal shuffling and sorting phase. After the local-reduce phase, the produced intermediate results are first aggregated on the public and the private cloud separately, and then combined on the private cloud, as illustrated in Figure 3.5. In

other words, data downgraded in the local-reduce phase still remains in the private cloud for subsequent processing to prevent unintentional information leakage. Specifically, this scheduler performs the following steps.

1. (Local-reduce) Suppose that worker p is assigned to a map-task. After p has completed the map-task giving the output L, the same p is selected to perform the reduce-task on tuples in L.

2. (Phase 1) For each key k, the output from the local-reduce tasks are partitioned into two groups: P_k consisting of tagged-tuples generated by private workers, and Q_k consisting of tagged-tuples generated by public workers. Next, a private worker is selected and assigned to perform reduce-task on P_k, and a worker (can be private or public) is selected and assigned to perform reduce-task on Q_k respectively.

3. (Phase 2) For each k, a private worker is selected and assigned to perform reduce-task on the output from phase 1.

Compared to the two-phase crossing mode, under this mode, the utilization of public servers is expected to be lower, but the volume of inter-cloud data traffic may decrease.

3.4.3 Hand-Off Mode (Unique Tag)

In both of the above TPC and TPNC modes, an additional phase (i.e., phase 2) is required to combine the partial reduce outputs produced in the two clouds as a key may be associated with both sensitive and non-sensitive tuples. Now we consider a property of the map function whereby this additional combining phase is not required.

PROPERTY 3.2 (UNIQUE TAG) *Given a multiset of tagged-tuples U, we say that the keys in U have unique tag if there does not exist a key k such that both $\langle k, v; \textit{sensitive}\rangle$ and $\langle k, v'; \textit{non-sensitive}\rangle$ are in U for some v and v'.*

We say that a map function meets the unique tag property if, on any input and any execution, completion of the map phase gives a set of tagged tuples with unique tag.

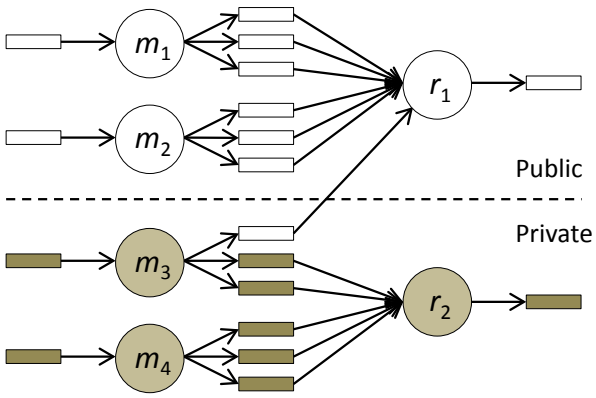


Figure 3.6: Hand-Off (HO) mode.

An example of the unique tag property is a map function that outputs $\langle k, v; t\rangle$ where t is a deterministic function of k. When a map function meets the unique tag property, after the map phase, a key appears either in the sensitive tuples or in the non-sensitive tuples, but never in both. Then, there is an easy way to schedule the reduce tasks – simply assign keys tagged as sensitive to private workers, and keys tagged as non-sensitive to either a public or a private worker. Since no combination or morphing of tasks is required, we call this mode the *hand-off* mode. Figure 3.6 illustrates this mode.

3.4.4 Mode Selection

Selection of the scheduling modes can be done automatically by the system if properties of the MapReduce computation are specified. Algorithm 1 provides a simple logic for deciding the scheduling mode to be used. The parameter `isAggregateReduce` indicates whether the reduce is an aggregating function or not, that is, whether the reduce output is expected to be smaller in size than its input. However, as shown by the experiments in Section 3.7, the best mode also depends on other factors like the ratio of sensitive data and the scale of the public and private clouds. We allow programmers to directly set the scheduling mode by providing an additional API `setSchedulingMode()`.

Algorithm 1 Simple logic for deciding the scheduling mode

```
// compProperty: property of the computation
// isAggregateReduce: whether reduce is an aggregating function
if (compProperty == unique_tag)
    mode = HO
else
    if (compProperty == partitionable_reduce)
        if (isAggregateReduce == true)
            mode = TPNC
        else
            mode = TPC
else
    mode = SP
```

3.5 Security Analysis

A key observation we have in designing the above scheduling modes is that, different schedulers could leak different information and some schedulers leak much more information than others. This motivates us to give a security model to compare the information leakage made by different schedulers and to determine whether a scheduler is secure or not. This section describes the proposed security model.

We consider public servers to be *honest-but-curious*. That is, the public servers will follow the protocol as expected and carry out the computations honestly, but may retain knowledge derived from the computations for malicious purpose. We allow public servers to collude, thus, we assume that our adversaries have control of all the public servers. In addition, we assume that the identities of the private servers, the scheduling algorithms and the map/reduce operations are public information. Although the scheduling algorithms are public, the scheduler resides in the private cloud and thus our adversaries do not have direct access to the scheduler's internal states, rather, they are limited to observing the interactions with the public cloud.

3.5.1 Motivating Examples

A subtle consequence of allowing schedulers to rearrange the computations is that, different scheduling algorithms could lead to different dataflows during execution, resulting in sending different data to the public cloud. Hence, from a curious public server's viewpoint, the amount and types of information leaked by different schedulers can be different. In particular, a scheduler that aggressively rearranges the tuples and tasks, while preserving MapReduce correctness and improving efficiency, may leak more information than a "conservative" scheduler that carries out all reduce computation in the private cloud. To illustrate the concerns, let us consider the following two examples:

Example I

Consider a simple reduce function that on input a list of values with the same key k, outputs $\langle k, (s, m)\rangle$ where s is the sum of the values, and m is the total number of input tuples. The output is tagged as *non-sensitive* iff m is greater than a threshold, say 50. Since this reduce function is partitionable, it can be computed in a divide-and-conquer manner. An ambitious scheduler might divide the sensitive input tuples into groups of 50, and assign the reduce-task on each group to a private worker. Next, the aggregated non-sensitive output from each group is sent to a public worker for further aggregation. Now, let us consider a conservative scheduler whereby all reduce-tasks are assigned to private workers. Compare to this conservative scheduler, the ambitious scheduler will reveal the sum of each group to the public cloud. One may argue that the sum of any sufficiently large group is deemed to be non-sensitive by the programmer and thus it is acceptable to reveal the sums of many large subgroups. However, that may not be the intention of the programmer and hence we need a clear security model to establish a baseline.

Example II

Here is a more subtle example. Consider another ambitious scheduler who dynamically tracks the intermediate tuples generated by the map-tasks. If a particular key k has only

non-sensitive intermediate tuples, then the scheduler will assign the reduce-task on k to a public worker. Since no sensitive tuple is sent to the public worker, it seems that this scheduler does not leak sensitive information. Now, compare to the conservative scheduler described in the previous example, the action of this scheduler reveals an additional piece of information on a key k: the fact of whether there exists a sensitive intermediate tuple with the key k. Although this piece of information might be insignificant or irrelevant in some applications, we need a security model that clearly accepts or disallows such leakage.

The above two examples bring out the subtlety and challenges in formulating the security model. What should be the "baseline" of leakage that is acceptable, and how to compare the leakages incurred by different schedulers? We handle this issue by treating the conservative scheduler described above as the baseline, and propose a security model to compare a scheduler with this conservative scheduler. Essentially, we say that a scheduler S_1 does not leak more than another scheduler S_2 iff we can simulate S_1 and generate the information revealed by S_1 based on the information revealed by S_2. Schedulers that do not leak more than the baseline are considered secure.

3.5.2 Scheduler-View and Public-View

A scheduler assigns map and reduce tasks to workers based on some scheduling algorithm. Hence, the scheduler has knowledge about all the input and output relationships between the tuples and workers. Let $\mathcal{V} = \mathcal{V}_p \cup \mathcal{V}_q$ be the set of entities of all the public and private workers. Let us first define the *scheduler-view* as what information a scheduler can gather during an execution.

DEFINITION 3.3 (SCHEDULER-VIEW) *Given a scheduler S, a MapReduce job C and an input D, the* scheduler-view *of an execution of C on D under S is a (possibly randomized) directed, acyclic graph $\mathcal{G}^S_{C,D} = \langle T, V, E \rangle$ where T is the set of all (input, intermediate and output) tagged-tuples, $V \subseteq \mathcal{V}$ is the set of involved workers and E represents the*

input/output relations between the tuples and workers.

Figure 3.3–3.6 give examples of the scheduler-view. In a scheduler-view, each tuple t is associated with a source src_t indicating the identity of the worker that outputs t, or a destination dst_t indicating the identity of the worker that takes t as input, or both. Note that on a same instance of C and D, the execution could be different since the scheduler and/or the computation could be non-deterministic. Hence, we are interested in the distribution of the scheduler-view.

Next, let us define *public-view* as what information the public cloud can gather during an execution.

DEFINITION 3.4 (PUBLIC-VIEW) *Given a scheduler S, a MapReduce job C and an input D, the* public-view *of an execution of C on D under S is a (possibly randomized) directed, acyclic graph $\mathcal{P}^S_{C,D} = \langle T_p, V_p, E_p \rangle$ which is a subgraph of the corresponding scheduler-view $\mathcal{G}^S_{C,D} = \langle T, V, E \rangle$, where $T_p = \{t | t \in T \wedge (src_t \in \mathcal{V}_p \vee dst_t \in \mathcal{V}_p)\}, V_p = \{v | v \in V \wedge \exists t \in T_p, src_t = v \vee dst_t = v\}$, and $E_p \subseteq E$.*

Essentially, a public-view contains information that is visible to the public cloud during an execution, which includes:

I.1. All the tuples that are taken as input or generated by the public workers (i.e., T_p).

I.2. The identities of the private workers that output or take as input any tuples in T_p.

I.3. The internal states of all the public workers in $\mathcal{V}_p$.

Figure 3.7 gives examples of the public-view corresponding to the scheduler-views shown in Figure 3.3–3.6. Similar to the scheduler-view, we are also interested in the distribution of the public-view.

3.5.3 Baseline - the Conservative Scheduler

Now let us consider a conservative scheduler which performs in the follow way:

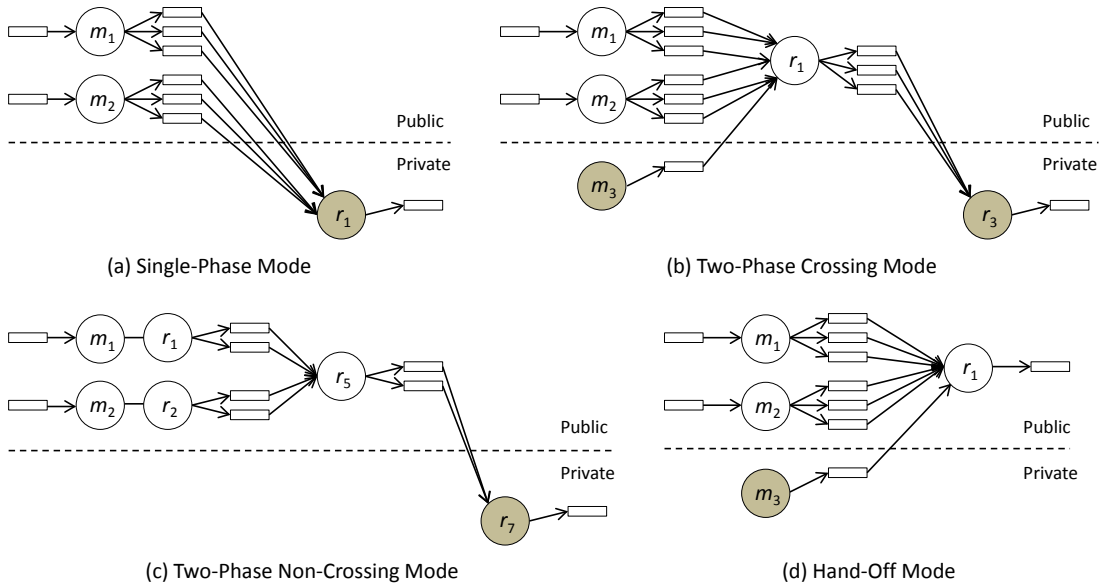


Figure 3.7: Examples of the public-view corresponding to the scheduler-views illustrated in Figure 3.3–3.6.

1. It assigns map-tasks operating on sensitive tuples to randomly chosen private workers, and map-tasks on non-sensitive tuples to randomly chosen public workers respectively;

2. It assigns all reduce-tasks to randomly chosen private workers;

3. All non-sensitive intermediate tuples will be sent to some public workers for temporary storage. [10]

This scheduler is "conservative" since it does not attempt to re-arrange the tasks for better performance. While there may be still some information leakage by virtue of data going to the public cloud, one assumes by definition that non-sensitive tuples can be disclosed. Since the goal is outsourcing of "some" computations to the public cloud, such leakage is considered to be acceptable. In this sense, the conservative scheduler is reasonable for analyzing the security of scheduling algorithms. Hence, we choose this simple execution model as the baseline and call it the *baseline scheduler*. We denote the baseline scheduler as S_B.

[10] In other words, all non-sensitive tuples are public information.

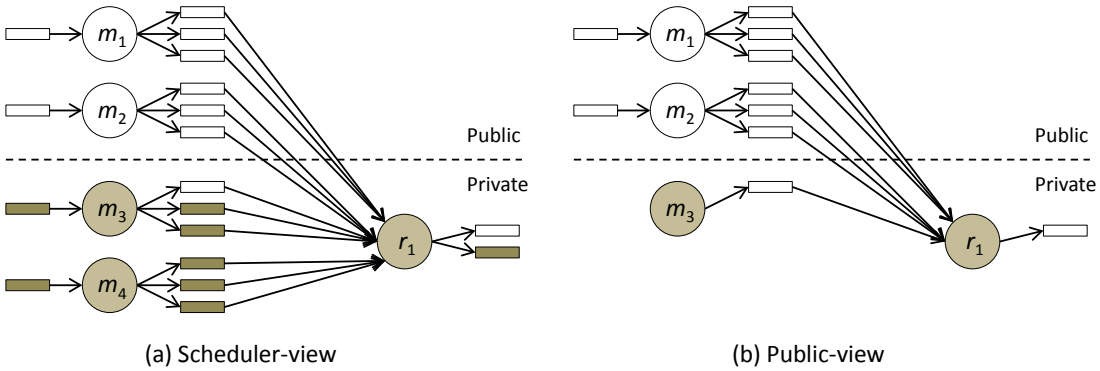


Figure 3.8: Illustration of the baseline scheduler: (a) scheduler-view; (b) the corresponding public-view.

DEFINITION 3.5 (PUBLIC-VIEW OF THE BASELINE SCHEDULER) *Given the baseline scheduler S_B, a MapReduce job C and an input D, the public-view $\mathcal{P}_{C,D}^{S_B} = \langle T_B, V_B, E_B \rangle$ of executing C on D under S_B is a subgraph of the corresponding scheduler-view $\mathcal{G}_{C,D}^{S_B} = \langle T, V, E \rangle$ where $T_B = \{t | t \in T \wedge t$ is non-sensitive$\}$, $V_B = \{v | v \in V \wedge \exists t \in T_B,\ src_t = v \vee dst_t = v\}$, and $E_B \subseteq E$.*

In other words, the public-view of an execution under the baseline scheduler includes the content of all non-sensitive tuples and the identities of the workers that generate or read those non-sensitive tuples. Figure 3.8 illustrates the scheduler-view and corresponding public-view of the baseline scheduler. Note the difference from the public-view of the single-phase mode as shown in Figure 3.7(a).

3.5.4 Security Model

Given a particular scheduler S, we want to analyze and determine whether it leaks "more" than the baseline scheduler S_B. Let us first define what it means by "a scheduler S leaking more information than another scheduler $\widetilde{S}$".

For a MapReduce job C, let us consider an oracle $\mathcal{O}$ that, on any input dataset D, generates a public-view $\mathcal{P}_{C,D}^{\widetilde{S}}$ of the scheduler $\widetilde{S}$. Note that this is just one sample from the distribution of the public-view of $\widetilde{S}$. Now, let us consider a simulator $\mathcal{M}$ that has access to $\mathcal{O}$ once. Based on the generated public-view $\mathcal{P}_{C,D}^{\widetilde{S}}$, this simulator simulates

the behavior of S and attempts to generate a public-view $\widetilde{\mathcal{P}}^S_{C,D}$ of the scheduler S in question. We say that a scheduler S does not leak additional information than another $\widetilde{S}$ on a particular job C if, there exists a simulator $\mathcal{M}$ such that, for any input D, $\widetilde{\mathcal{P}}^S_{C,D}$ and $\mathcal{P}^S_{C,D}$ are statistically close on D. Note that the simulator $\mathcal{M}$ can be different for different jobs. A scheduler S does not leak more than another $\widetilde{S}$ if the above holds for any job C. More specifically,

DEFINITION 3.6 *A scheduler S does not leak more information than another scheduler $\widetilde{S}$ if, for any MapReduce job C, there exists a simulator $\mathcal{M}$ that, for any input D, $\mathcal{M}$ can simulate and generate a public-view $\widetilde{\mathcal{P}}^S_{C,D}$ for S from the public-view $\mathcal{P}^{\widetilde{S}}_{C,D}$ of $\widetilde{S}$ such that*

$$\sum_x |Pr[\widetilde{\mathcal{P}}^S_{C,D} = x] - Pr[\mathcal{P}^S_{C,D} = x]| \leq \epsilon$$

We say that the leakage of a scheduler S is *acceptable* if S does not leak more information than the baseline scheduler S_B.

3.5.5 Leaky Implementation

Given the security model defined in Section 3.5.4, we are able to prove that the four scheduling modes presented in Section 3.4 are secure. Before that, let us first revisit the two examples given in Section 3.5.1 and show that they indeed leak more information than the baseline scheduler.

Two-Phase Crossing with Local-Reducer

Let us consider a scheduling mode that is related to **Example I**. This mode is another "optimization" of the TPC mode, where the reduce function is first applied locally to each map-task's output, and then followed by the original two phases in TPC, as illustrated in Figure 3.9. The difference from the TPNC mode is that, under this mode, tuples downgraded to be non-sensitive from local-reduce tasks are allowed to go to the public workers for partial reducing. Potentially, this mode increases the utilization of the public servers

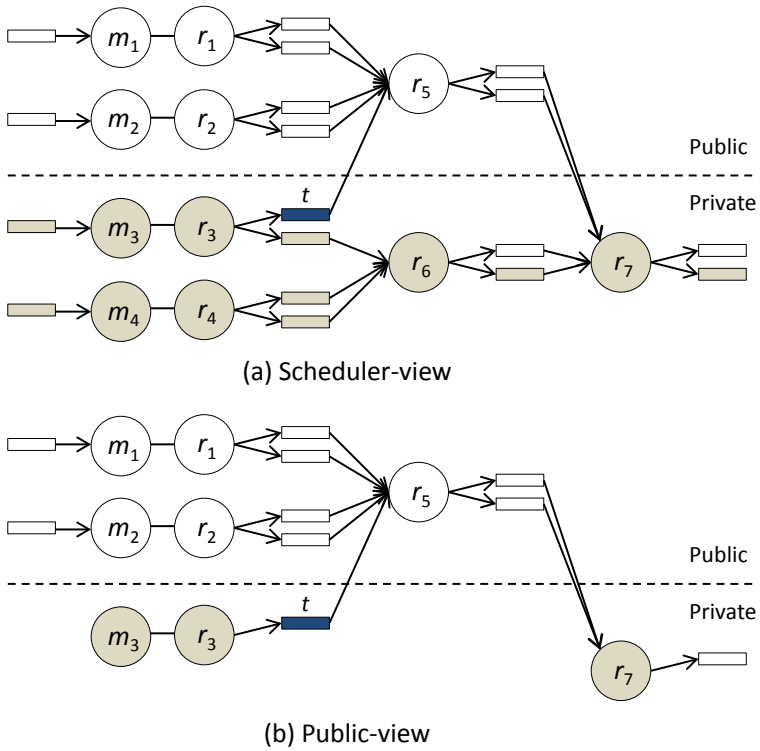


Figure 3.9: Two-Phase Crossing with Local-Reducer.

and thus may give more effective load-balancing. Let us denote this scheduling mode as S_L.

However, this S_L mode leaks more information than the baseline scheduler S_B. Suppose that on a particular job C with an input D, a tuple t (marked as dark blue in Figure 3.9) is tagged as non-sensitive from the local-reducer of r_3. The tuple t is then sent to the public cloud for partial reducing, hence, $t \in \mathcal{P}_{C,D}^{S_L}$. However, $t \notin \mathcal{P}_{C,D}^{S_B}$ according to our definition of the baseline scheduler S_B. Hence, any simulator is unable to generate the public-view of this scheduler.

Dynamic Scheduling

The dynamic scheduling described before in **Example II** can potentially lead to more effective load-balancing. Unfortunately, it also leaks additional information. A simulator is unable to generate the public-view of the dynamic scheduler (denoted as S_Y). This is because the simulator does not know the existent of the sensitive tuples, and thus is unable

to decide whether to assign the task to a private or a public worker. More precisely, for a particular job C, we can construct two different data inputs D_0 and D_1 such that the distributions of $\mathcal{P}^{S_B}_{C,D_0}$ and $\mathcal{P}^{S_B}_{C,D_1}$ are the same but $\mathcal{P}^{S_Y}_{C,D_0}$ and $\mathcal{P}^{S_Y}_{C,D_1}$ are different. Hence, no simulator can generate the correct public-view for this dynamic scheduler.

3.5.6 Security of the Proposed Modes

Now we show that the proposed modes are secure in the sense that they do not leak more than the baseline scheduler.

THEOREM 3.1 *The leakage of each of the proposed SP, TPC, TPNC and HO scheduling modes is acceptable, that is, they do not leak more information than the baseline scheduler* S_B.

Proof In order to demonstrate the security of the proposed modes, it is sufficient to show that a simulator can be constructed for each mode so that, for any MapReduce computation C and any input D, the simulator's output is statistically close to the public-view of the scheduling mode in question. We will only analyze the security of the Two-Phase Non-Crossing mode (denoted as S_N). Analysis of the other three modes can be similarly done. For each $\mathcal{P}^{S_B}_{C,D}$ obtained from the oracle $\mathcal{O}$, our simulator constructs a public-view $\widetilde{\mathcal{P}}^{S_N}_{C,D}$ for S_N by simulating the process described in Section 3.4.2:

1. First, it simulates the TPNC scheduler in selecting the workers for map-tasks on non-sensitive tuples, and carries out the map-tasks (with local reduction). Note the some private workers might be selected to perform on the non-sensitive tuples. Ignore these private workers.

2. Next, it performs a reduce-task on the non-sensitive partition produced by the public workers for each k (i.e., phase 1).

3. Finally, it simulates the TPNC scheduler in selecting the private workers for final reducing in phase 2. Recall that the private workers are randomly selected from the

pool of all private workers, and thus can be simulated.

It is clear that the constructed public-view $\widetilde{\mathcal{P}}_{C,D}^{S_N}$ is statistically equal to the public-view $\mathcal{P}_{C,D}^{S_N}$ of S_N in distribution. Since we can simulate the TPNC mode and generate its public-view from that of the baseline scheduler, the leakage of the TPNC mode is acceptable under our security model.

3.5.7 Side-Channel Information

During execution, adversaries in the public cloud could measure the size and timing of packets received from each private server. Analysis of such network traffic might provide information on the workload of individual private servers. The workload may depend on the actual content of sensitive tuples, which although unlikely, could potentially leak information of the content. Since the network traffic is heavily influenced by other factors like overall network conditions and time of the day, and these information may still present even if encrypted computation (e.g., homomorphic encryption) is used, we consider these as "side-channel" information and do not capture them in our security model. Nevertheless, there are mechanisms to reduce such leakage such as hiding the identities of the private servers by routing the traffic to a proxy and "translating" the identities (which is similar to NAT (Network Address Translation)), inserting random delays to the traffic, adding noise in the scheduling, etc. The issue of side-channels is orthogonal to this work. In Section 3.8 later, we will explore in detail how one of the mechanisms can be used to reduce such side-channel leakage.

3.6 Implementation

We have prototyped tagged-MapReduce based on Hadoop 1.0.1. To help understand our implementation, we first provide an overview of Hadoop.

3.6.1 Hadoop Overview

Apache Hadoop [18] is a popular open-source implementation of MapReduce. Hadoop consists of two main components: the Distributed File System (HDFS) and the MapReduce engine. The HDFS is a distributed file system that supports huge amount of data storage across the cluster nodes, providing very high availability and bandwidth access to the applications' data. Above HDFS is the MapReduce engine where the work is divided into many small pieces of tasks, each of which may be executed or re-executed on any node in the cluster.

Hadoop Distributed File System (HDFS)

The design of HDFS is based on the design of GFS, the Google File System [87]. HDFS is a block-structured file system: individual files are broken into blocks of a fixed size (e.g., 64 MB) when they are being loaded in. These blocks are stored across a cluster of one or more machines with data storage capacity. Individual machines in the cluster are referred to as **DataNodes**. A file can be made of several blocks, and they are not necessarily stored on the same machine; the target machines which hold each block are chosen randomly on a block-by-block basis. Thus access to a file may require the cooperation of multiple machines. For reliability, each file block is replicated across a number of machines (3, by default), such that the loss of any one copy of the block will not render the whole file as unavailable. An overview of the HDFS architecture is illustrated in figure 3.10.

HDFS uses a dedicated single machine, called the **NameNode**, to store and maintain all its metadata (e.g., the names of files and directories, the locations of each block of each file). To open a file, a client contacts the NameNode and retrieves a list of locations for the blocks that comprise the file. These locations identify the DataNodes which hold each block. Clients then read file data directly from the DataNode servers, possibly in parallel. The NameNode is not directly involved in this bulk data transfer, keeping its overhead to a minimum. Of course, NameNode information must be preserved even if the NameNode machine fails; there are multiple redundant systems that allow the NameNode

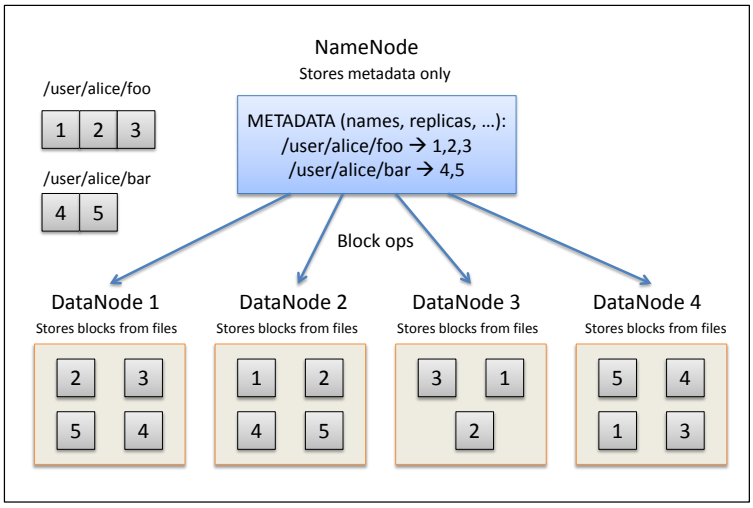


Figure 3.10: Overview of Hadoop distributed file system (HDFS). DataNodes store actual blocks from files while NameNode stores only the metadata.

to preserve the file system's metadata even if the NameNode itself crashes irrecoverably. Fortunately, as the NameNode's involvement is relatively minimal, the odds of it failing are considerably lower than the odds of an arbitrary DataNode failing at any given point in time. More details about the design and implementation of HDFS can be found in a guide document [47].

Hadoop MapReduce

Above HDFS comes the Hadoop MapReduce engine, which is designed for easily writing applications that process vast amount of data in-parallel. Typically the MapReduce framework and HDFS are working on the same set of cluster nodes, allowing the system to effectively schedule tasks to the nodes where data is already present there. Such manner results in very high aggregate bandwidth across the cluster while minimizing the communication cost. Figure 3.11 gives a high-level overview of the MapReduce work flow.

MapReduce inputs are typically loaded from the HDFS, where data files are evenly distributed across all the nodes. Running a MapReduce program involves running mappers on many or all of the nodes in the cluster, in a parallel manner. Each of these mappers is equivalent: no mappers have particular "identities" associated with them. Therefore,

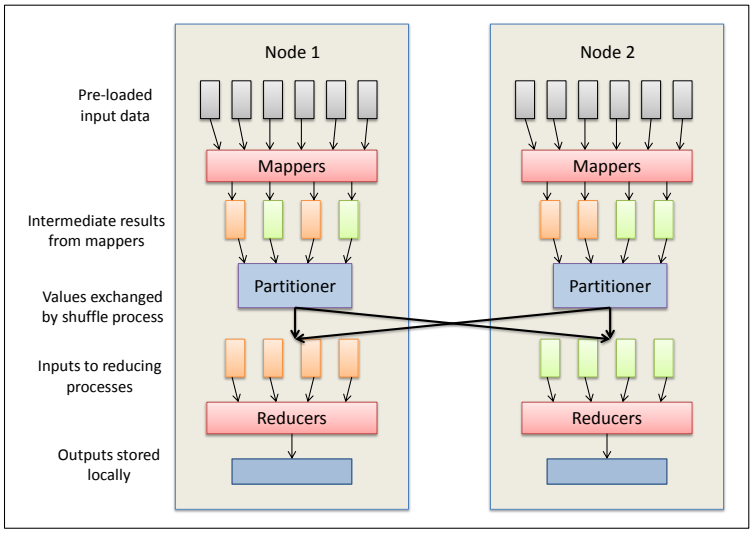


Figure 3.11: High-level overview of Hadoop MapReduce workflow.

any mapper can process any input file. Each mapper loads the set of files local to that machine and processes the interesting user-specified work. Given an input (i.e., a key and a value), the Mapper emits new $\langle key, value \rangle$ pair(s) which are forwarded to the reducers.

When the mapping phase has completed, the intermediate $\langle key, value \rangle$ pairs are exchanged between machines to send all values with the same key to a single reducer, which is known as *shuffling*. The intermediate key space is divided into several different subsets, each of which is assigned to a different reduce node; these subsets (known as "partitions") are the inputs to the reducer. Each mapper may emit $\langle key, value \rangle$ pairs to any partition; all values for the same key are always reduced together regardless of which mapper it comes from. The *Partitioner* class determines which partition a given $\langle key, value \rangle$ pair will go to. The default partitioner computes a hash value for the key and assigns the partition based on this result. Of course, users can write their own Partitioner implementation. Note that this is the only communication step in MapReduce. Individual mappers do not exchange information with one another, nor do the reducers discussed later. The user never explicitly manipulates information from one machine to another; all data transfer is handled by the Hadoop MapReduce platform itself, guided implicitly by the different keys associated with values.

Several reducers are also created which spread across the same nodes in the cluster as

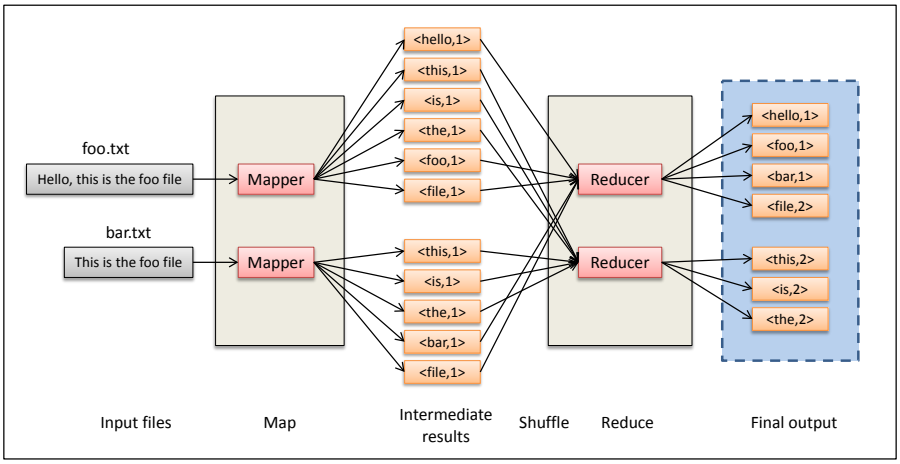


Figure 3.12: The word count example on Hadoop. Suppose we have two files, foo.txt and bar.txt. Two mappers (and reducers) were created to process them. Intermediate results with the same key were sent to the same reducer.

the mappers. They are instances of user-provided code that perform the second important phase of program-specific work. Each reducer receives a key as well as an iterator over all the values associated with the key. The values associated with a key are returned by the iterator in an undefined order. The reducer will go through all these values, performing the user-specified computations. Reducing results are written back to an output file, typically. Figure 3.12 illustrates the "word count" computation on Hadoop.

3.6.2 Input Data Tagging

Note that our framework works on tagged key-value pairs, while normal input data, e.g., network log files or surveillance videos, are typically not tagged. To perform a privacy-preserving computation over our tagged-MapReduce, one has to first tag the input data by indicating the sensitive part. While in principle each key-value pair should be tagged, doing so for big data may be cumbersome and time-consuming, and may itself require another MapReduce job. For usability, we implemented a simple and yet practical labeling process, *file-level tagging*, whereby each file in the input is labeled as sensitive or non-sensitive. As mentioned in Section 3.2.2, this file-level tagging is simple but does not lose generality. For example, a file containing both sensitive and non-sensitive data can

be simply tagged as sensitive and fed into an initial tagged-MapReduce job that outputs two separate files, each containing all the records with a particular sensitivity. To label the input, the user creates a meta file specifying the sensitivity of each file in the input. The meta file is then submitted, together with the input data, to the modified HDFS. Note that this tagging step is only required for the initial input data. For inputs to the subsequent jobs (in the case of chained MapReduce), the data are already associated with sensitivity tags.

3.6.3 Data Uploading and Replication

The Hadoop's data replication process is modified to prevent sensitive files from being stored in the public cloud. Specifically, we extended Hadoop's `INodeFile` class by adding a private boolean field, `sensitive`, whose value is to be determined from the sensitivity of the corresponding file, as specified in the meta file. The namenode, which runs in the private cloud, now allocates data blocks in a privacy-aware manner: a data bock from a sensitive file is always allocated and replicated to the private datanodes, while a data block from a non-sensitive file is first allocated to a public datanode, and then replicated to random targets among all the remaining public and private datanodes.

3.6.4 Map Task Management

Upon receiving a computing job from the user, the jobtracker in Hadoop creates a list of map and reduce tasks for that job where each map task is associated with one data block (i.e., `InputSplit`). Task scheduling is done by the jobtracker based on the heartbeat mechanism: whenever a heartbeat message comes indicating that a tasktracker is ready to run a task, the jobtracker chooses from the unassigned task list the most appropriate task for that node (e.g., considering data locality when choosing a map task), and then assigns it to that tasktracker.

We modified the above process by adding to each task (the `TaskInProgress` class) a binary label which is either *sensitive* or *non-sensitive*. A map task is sensitive iff the

associated `InputSplit` comes from a sensitive input file. A sensitive task will always be assigned to a tasktracker running in the private cloud while there is no constraint for non-sensitive tasks.

3.6.5 Reduce Task Management

The assignment of sensitive and non-sensitive reduce tasks depends on the scheduling modes. For example, in the single-phase mode, all reduce tasks are created as sensitive. In other modes, the ratio of sensitive and non-sensitive reduce tasks is automatically determined by the system according to the computing power of the private and the public cloud. Suppose the cluster has N_p private nodes and N_q public nodes with uniform processing capability, and the user wants to create n reduce tasks, then the first $L(N_p, N_q, n) = \left\lceil \frac{nN_p}{N_p+N_q} \right\rceil$ reduce tasks are reserved as sensitive. The choice of the function $L()$ here is a reasonable default to balance the workload between the public and the private cloud, however, a different $L()$ is feasible.

For TPC and TPNC modes that require two phases of reduce, additional $L(N_p, N_q, n)$ sensitive reduce tasks are then created to combine the partial reduce-output produced in the first phase. The data shuffling and sorting phase between reduce-reduce is similarly implemented to that between map-reduce.

We also extended the default `HashPartitioner` class of Hadoop to consider tags when partitioning the intermediate data. Specifically, a map task generates a set of intermediate tuples where each tuple $(k, v; t)$ is to be stored in the $Hash(k, t)$-th partition, where $Hash(\cdot, \cdot)$ is some deterministic hash function. The hash function is defined in such a way that, if t is sensitive, then

$$0 \leq H(k, t) < L(N_p, N_q, n)$$

otherwise

$$L(N_p, N_q, n) \leq H(k, t) < n$$

Note that the i-th partition is going to be processed by the i-reduce task. Hence, the above hash function ensures that a sensitive tuple will always be passed to a sensitive reduce task running in the private cloud. Note that the intermediate shuffling and sorting phase between map-reduce is not affected.

3.7 Evaluation

We experiment with our prototype on Amazon EC2 to evaluate the practicality of the system and the effectiveness of the proposed scheduling modes in terms of: (i) inter-cloud communication cost; (ii) total job running time; and (iii) computation outsourcing ratio. The experiments are run using both simple (single) and complex (chained) jobs. In addition, we also study the effect of different public cloud sizes.

3.7.1 Experimental Setting

We describe here the setting of our experimental study, including the data and computing jobs, and the hybrid-cloud environment over which our system is run.

Computing Jobs and Datasets

We run 5 simple MapReduce jobs where it is natural to have input data with mixed-sensitivity: *Word Count*, *Sort*, *Inverted Index*, *Traffic Statistics* and *Face Detection*. The word count job, which is an extension to the classic MapReduce example [74], counts the occurrences of each word in a large set of text files. The map output is sensitive iff the word is from a sensitive input file and not in the set of "stop words"; the reduce output is sensitive iff at least one of its input is sensitive. We run this job with the Wikipedia dataset [9] that contains English wikipedia articles up to July 2012. The sort job, working on the Google 1-Ngram dataset [21], sorts the input tuples by the number of times that each word occurs. The inverted index job, as a representative usage of MapReduce in large-scale search indexing systems, computes in which files and lines each word occurs,

Table 3.1: Summary of the computing jobs and datasets

Job	Dataset	Descriptions
Word Count	Wikipedia dataset (36.8GB)	Count the occurrences of each word
Sort	Google 1-gram dataset (9.7GB)	Sort the records by the number of occurrence
Inverted Index	Wikipedia dataset (36.8GB)	Compute the locations of each word
Traffic Statistics	DARPA IDS dataset (17GB)	Count the total traffic generated by each host
Face Detection	Image dataset (17.2GB)	Detect human faces from the images

with Wikipedia dataset as the input. Both the sort and inverted index jobs have similar sensitivity policies as word count. The traffic statistics job, on input a set of network logs from the DARPA intrusion detection systems (IDS) dataset [20], outputs the total amount of traffic generated by individual hosts (for detecting DoS attacks). The default tagging policy described in Section 3.3.1 is applied for this job. The face detection job detects human faces from a database of 80,000 images crawled from the web via Google Images. The output are extracted images of successfully detected faces where a face is tagged as sensitive if it is from a sensitive image which contains no more than 3 faces. The computing jobs and datasets are summarized in Table 3.1.

We split the Wikipedia, Google Ngram and IDS datasets each into 10 separate files of roughly the same size. Our experiment will vary the ratio of sensitive data over the whole input dataset by randomly tagging files approximately up to the ratio as sensitive. For the image dataset, to obtain a particular ratio of sensitive data, an appropriate number of randomly selected images are tagged as sensitive.

The Hybrid Cloud Setting

We build a hybrid cloud on Amazon EC2 across Singapore and US West. The private cloud consists of 3 instances located at Singapore and the public cloud has 0, 3 or 6 instances at N. California (i.e., we experiment on different scales of the public cloud). All instances (m1.large) run Ubuntu 12.04, and each provides 2 virtual cores with 4 ECUs (EC2 Compute Unit, each provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or Xeon processor), 7.5 GB memory and 850 GB storage. The bandwidth between these instances is not specified by Amazon. An informal test of file transfer using

`scp` gives 35–40 MB/s within the same region (e.g., from Singapore to Singapore), and 3–8 MB/s across different regions (e.g., from Singapore to US West). Though our cluster is small, our dataset size is quite modest and hence matches the cluster.

3.7.2 Experiments on Scheduling Modes

In this experiment, 3 instances are utilized on the public cloud, hence, the whole hybrid cluster has 6 instances in total. We run the above 5 jobs under the four scheduling modes presented in Section 3.4. As baselines, we also run them on original Hadoop over the whole hybrid cloud. Note that original Hadoop run is only meant for comparisons because Hadoop cannot be securely used in a hybrid cloud with sensitive data. For each case, we vary the ratio of sensitive data over the whole input between 20% to 80%. We record the total job running time (job elapsed time), the execution time of each task (individual CPU time) and the total amount of traffic across the public and private cloud (inter-cloud communication) by analyzing the Hadoop log files.

The hand-off mode requires the unique tag property which is not met by the word count, sort, inverted index and traffic statistics jobs as a same key can occur in both sensitive and non-sensitive files. Fortunately, we can use the following observation. Given a map $\hat{\mu}$ which produces $\langle k, v_1, s\rangle$ and $\langle k, v_2, n\rangle$ where s and n denote sensitive and non-sensitive, a simple transformation to meet the unique tag property is to instead output $\langle 1.k, v_1, s\rangle$ and $\langle 0.k, v_2, n\rangle$ where the dot is bit concatenation. Face detection uses image name as the key which is either sensitive or not, thus fits the unique tag property.

Inter-Cloud Communication

This measures how much data is transferred across the two clouds during computation. Note that due to the limited inter-cloud bandwidth, this could be a potential performance bottleneck. The result is shown in Figure 3.13.

The result of the word count, inverted index and traffic statistics jobs shows the effectiveness of our proposed modes in reducing the inter-cloud communication. In these jobs,

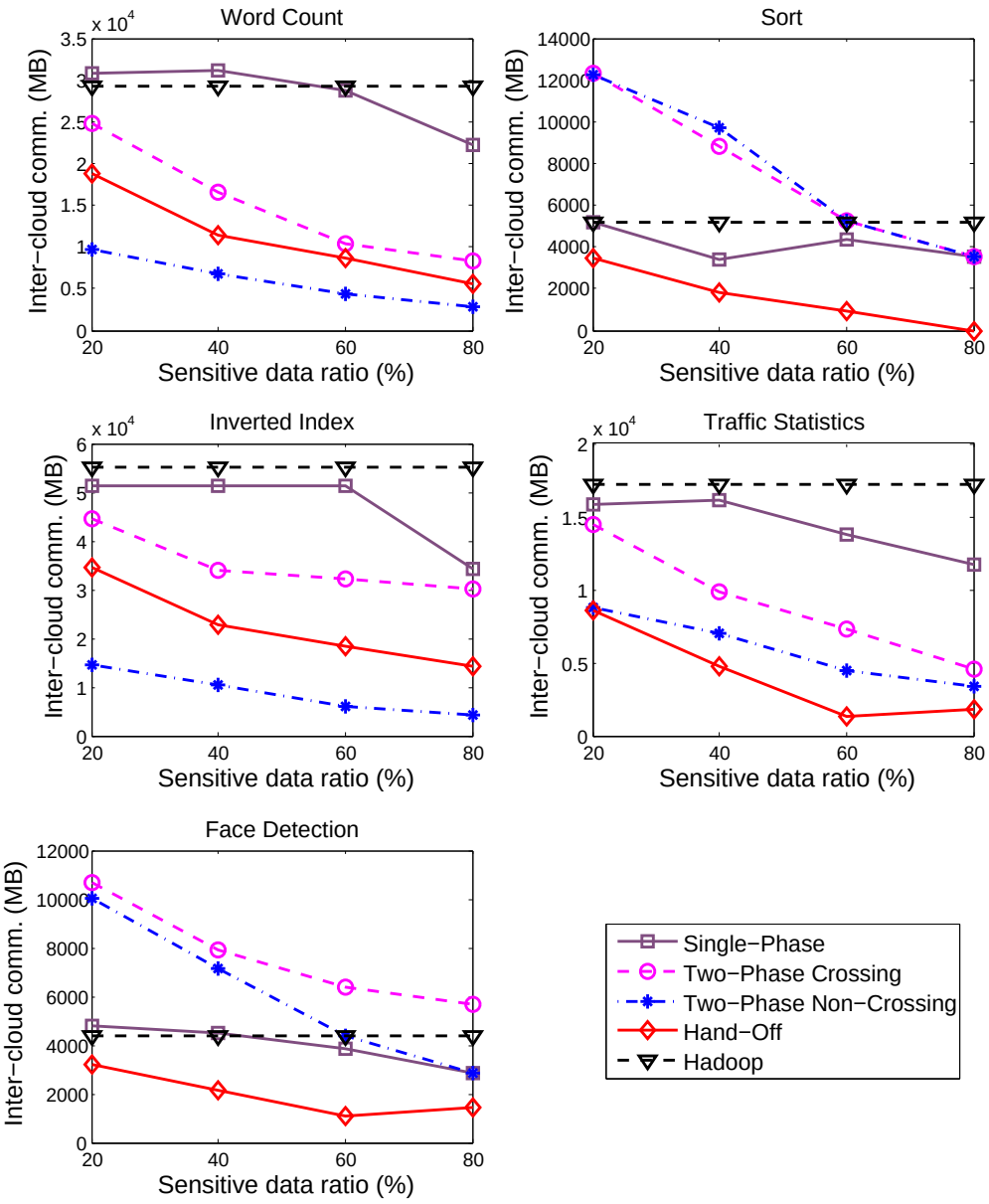


Figure 3.13: Inter-cloud communication.

our TPC, TPNC and HO modes can lessen the total amount of inter-cloud data traffic by orders of magnitude as compared to original Hadoop or the basic SP mode. In particular, the TPNC mode mostly incurs the least inter-cloud communication overhead as the reduce output is much smaller in size than the map output, hence the data (i.e., partial reduce output) transferred from the public to the private cloud is small.

The result of the sort and face detection jobs is different. Surprisingly, both TPC and TPNC modes incur larger inter-cloud communication than the SP mode or original

Hadoop. We observed that this is due to the behavior of the reduce function in these two jobs, which simply takes the input and outputs them. This means that the output from reduce is similar in size to the map output. The TPC and TPNC modes move all the intermediate results produced on the public cloud, together with those non-sensitive ones from the private cloud, back to the private cloud for final reduce, and hence incurs the largest inter-cloud data movement. In contrast, the SP mode only involves one step of data movement, giving less inter-cloud data traffic. The HO mode in these two jobs has the least inter-cloud communication cost indicating that very few sensitive data are downgraded to be non-sensitive.

Job Elapsed Time

The job elapsed time measures how long it takes to compute a MapReduce job. The result is illustrated in Figure 3.14. It is not surprising that most of our modes require a longer time to compute a same job than original Hadoop mainly due to the additional security constraint. However, the original Hadoop run does not preserve data-privacy. When the ratio of sensitive data is lower than 50%, there are roughly equal data processed on the public and private cloud, hence the time is low. As the ratio increases, the private nodes bear the burden of increased data so the time increases. Our modes can outperform the basic SP mode in the word count, inverted index and traffic statistics jobs. However, in the sort and face detection jobs, both the TPC and TPNC modes incur longer time than SP. Again this is due to the reduce behavior of these two jobs, which does not decrease the data size but incurs computational overhead. The HO mode in the word count, inverted index and face detection jobs can approach the performance of original Hadoop at a ratio of around 50% while constantly outperforming Hadoop in the sort and traffic statistics jobs, mainly due to the optimized inter-cloud data traffic. The TPNC mode in word count and inverted index completes even much faster than Hadoop. This indicates that inter-cloud data movement is indeed a significant performance bottleneck.

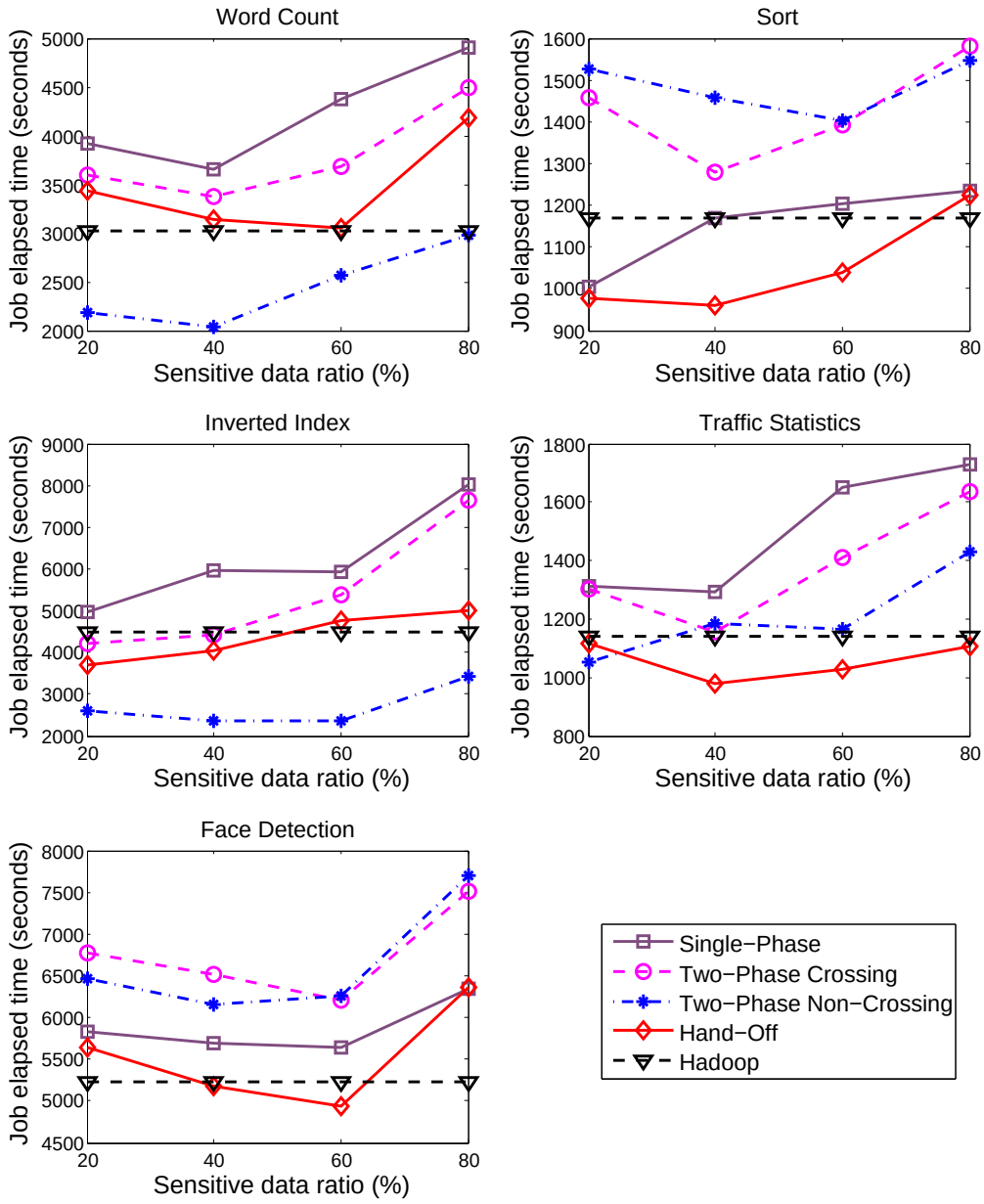


Figure 3.14: Job elapsed time.

Computation Outsourcing Ratio

The computation outsourcing ratio gives the percentage of total CPU time used in the public cloud over the total CPU time. It measures how much compute is outsourced to the public cloud. The result is given in Figure 3.15. The Hadoop baseline is around 50% which is expected as it randomly assigns tasks to all the nodes. The SP mode assigns all reduce tasks to the private cloud and thus serves as a lower bound of the other modes. The

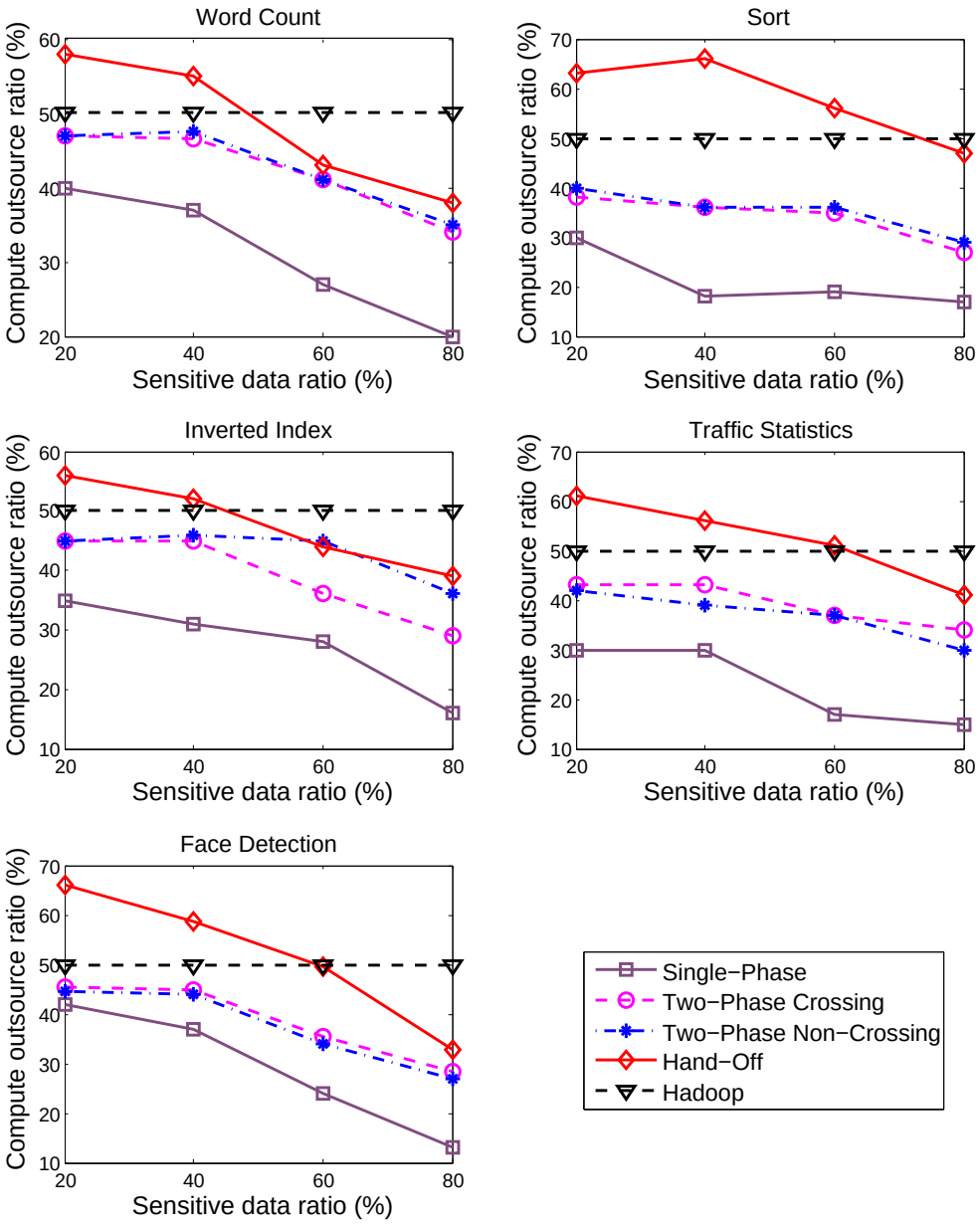


Figure 3.15: Computation outsourcing ratio.

HO mode achieves the best outsourcing as it saves the final reduce phase in the private cloud. The TPC and TPNC modes are not far behind and close to each other. Further, as the sensitive data ratio increases, naturally the outsourcing ratio decreases since less work is available to be outsourced.

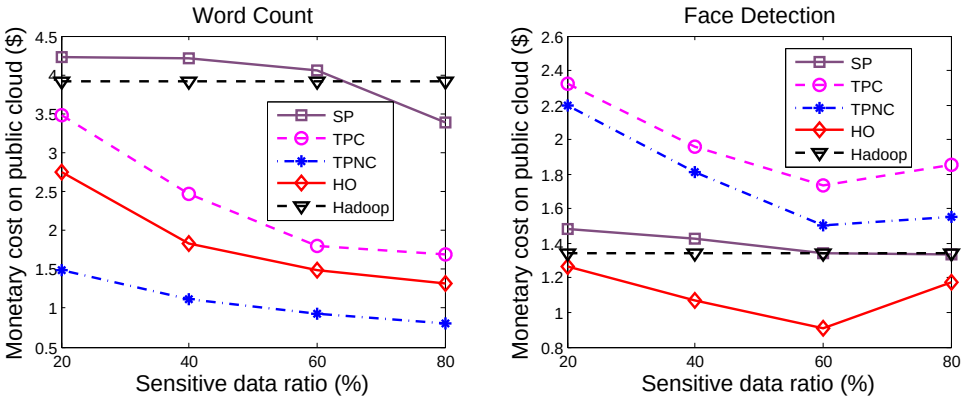


Figure 3.16: Monetary cost incurred on the public cloud.

Monetary Cost

Though our scheduling modes are not particularly designed for minimizing monetary cost, evaluating the cost with the above experiments is also instructive. Let us take Amazon EC2's pricing model as an example, the computation costs \$0.19 per hour for each standard large Linux/UNIX instance and bandwidth costs \$0.12 per GB for the inter-cloud data traffic.[11] We want to estimate how much will be charged by the public cloud provider (i.e., Amazon) when the jobs are computed on our hybrid cloud. The total cost is CPU + bandwidth costs. Taking the word count job with the SP mode as an example, at a sensitive ratio of 60%, the SP mode requires 4375 seconds to complete the job, giving a computation cost of $(4375/3600)\times 0.19\times 3\approx\$0.693$, and incurs 28.72GB inter-cloud data traffic, giving a bandwidth cost of $28.72\times 0.12\approx\$3.446$. This amounts to a total cost of \$4.139. We similarly calculate the costs for the other modes of the word count job, and also for all the modes of the face detection job. The results are shown in Figure 3.16

Figure 3.16 shows that our optimized scheduling modes, e.g., the TPC, TPNC and HO modes in word count and HO mode in face detection, can also reduce the monetary cost, as compared to the default SP mode or original Hadoop runs. Figure 3.16 shows similar trends to Figure 3.13 of inter-cloud communication as bandwidth cost dominates within

[11]Prices are taken from the N. California region in August 2013. Note that while Amazon only charges for data traffic from Amazon EC2 out to the Internet, for simplicity, we assume that both two-way traffics incur monetary cost.

the total cost.

Summary The above results demonstrate that, our proposed scheduling modes (i.e., TPC, TPNC and HO) can effectively reduce the inter-cloud communication and job execution time while being able to outsource more computation to the public cloud, as compared to the general SP mode. The time overheads are also reasonable as compared to original Hadoop runs on the same hybrid cluster which ignore the data security and privacy constraints. Besides, the monetary costs could also be reduced with carefully chosen scheduling modes.

3.7.3 Experiments on Different Baselines

The above Hadoop runs on the hybrid cloud do not meet the security requirement. In practice, one can only run Hadoop on private servers in order to prevent data leakage. We therefore experiment with Hadoop on the sole private cloud (i.e., only 3 nodes) as the baseline. We only compare the job elapsed time since there is no inter-cloud communication or computation outsourcing for Hadoop with the pure private cloud setting.

The result is illustrated in Figure 3.17. In the word count, inverted index and traffic statistics jobs, all of our modes give smaller job elapsed time compared to the Hadoop runs on the sole private cloud, as much work is outsourced to the public cloud. In the sort and face detection jobs, the TPC and TPNC modes incur high elapsed time which is comparable to that of the private Hadoop runs, mainly due to the overheads in two phases of computation and the high amount of inter-cloud data traffic.

In summary, while both preserving data security and privacy, our system has the advantages of reducing job execution time and outsourcing partial computation, as compared to Hadoop runs on the pure private cloud.

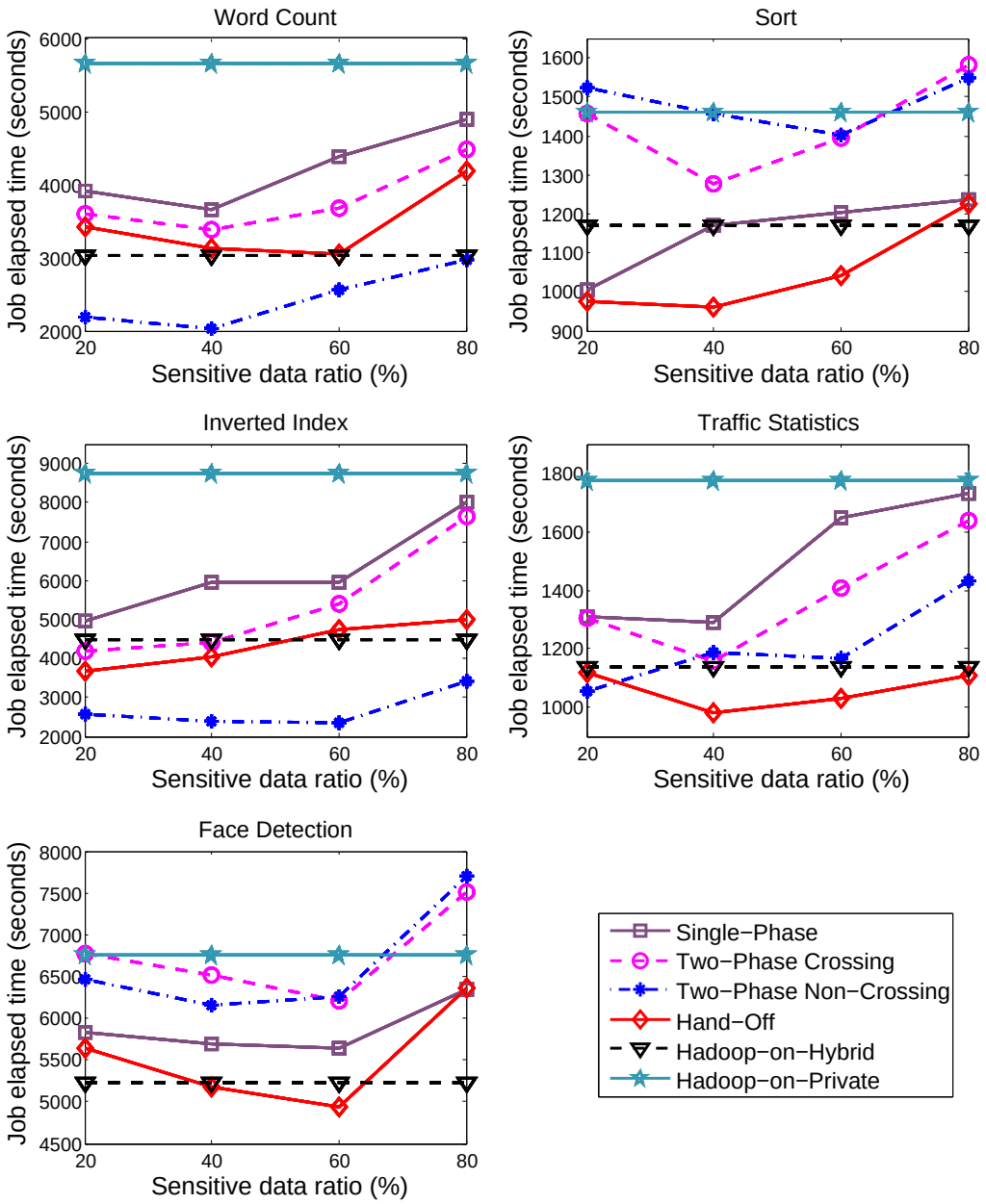


Figure 3.17: Job elapsed time with different baselines.

3.7.4 Experiments on Different Public Cloud Sizes

Next, we are interested in the effect of different public cloud sizes. We rerun the word count and sort jobs with our prototype under 3 different cloud settings: S_1 – *3 private nodes* only; S_2 – *3 private + 3 public nodes*; and S_3 – *3 private + 6 public nodes*. We record the job elapsed time, inter-cloud communication and computation outsourcing ratio similarly as in Section 3.7.2. The ratio of sensitive data over the whole input is around

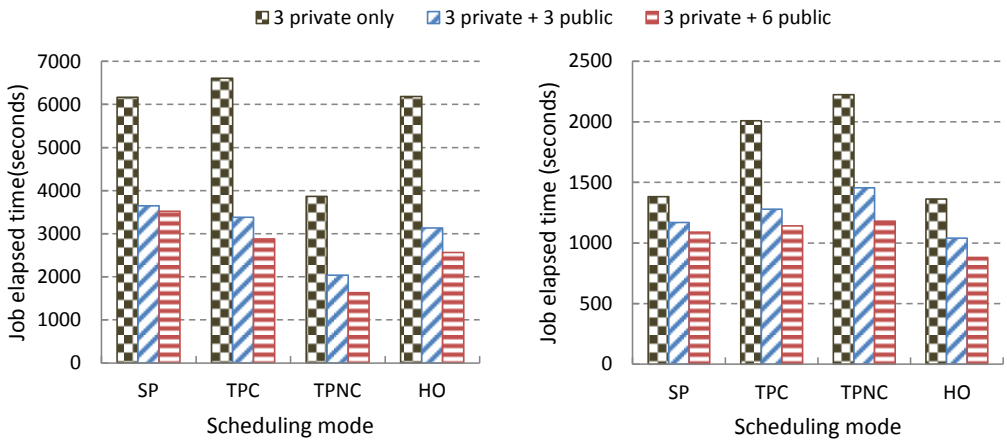


Figure 3.18: Job elapsed time with different public cloud sizes (left: word count; right: sort).

40% for both jobs.

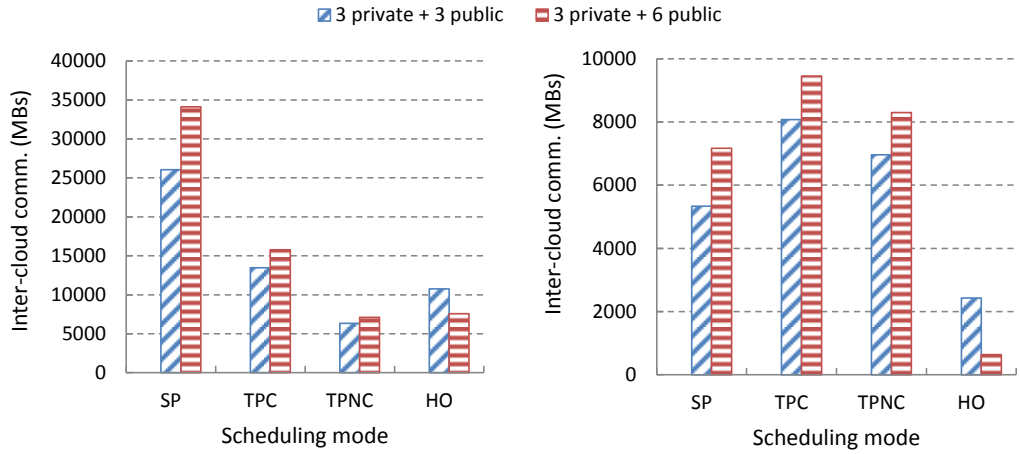


Figure 3.19: Inter-cloud data traffic with different public cloud sizes (left: word count; right: sort). The 3 private nodes only setting does not incur inter-cloud communication, and hence is not shown in the figure.

The results are shown in Figure 3.18–3.20. As expected, with a larger public cloud, we are able to outsource more work, giving a higher outsourcing ratio and lower job elapsed time. The time difference between S_2 and S_3 is small indicating that not much more work is available to be outsourced when increasing the public nodes from 3 to 6. For the SP, TPC and TPNC modes, the inter-cloud data traffic increases with a larger public cloud, since more data are computed on the public cloud which are finally shuffled back to the private cloud for merging. In contrast, the HO mode incurs less inter-cloud data transfer

Table 3.2: Mode assignment for individual MapReduce jobs

Application	Job	Mode
Wordcount+Sort	Word Count	Two-Phase Non-Crossing
	Sort	Single-Phase
Face Anonymization	Face Detection	Hand-off
	Average	Hand-off
	Sort	Single-Phase

when the public cloud size increases as it does not require the second-reduce phase.

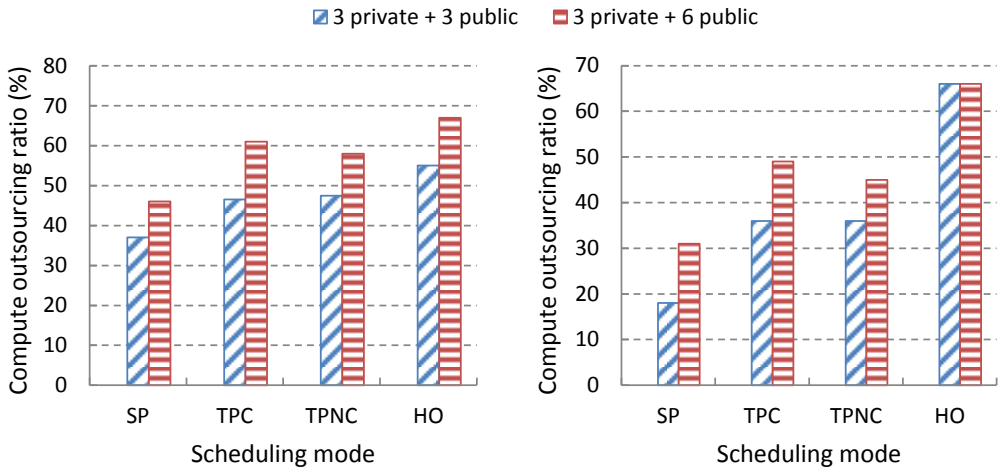


Figure 3.20: Computation outsourcing ratio with different public cloud sizes (left: word count; right: sort). The 3 private nodes only setting does not involve computation outsourcing, and hence is not shown in the figure.

3.7.5 Experiments with Chained MapReduce

Last, we experiment on more complex MapReduce jobs that involve chained MapReduce: *wordcount+sort* and *face anonymization*. We choose the most appropriate mode, as summarized in Table 3.2, for each individual job in the chain according to the properties of the computation. The ratio of sensitive data over the whole input is around 50% in both jobs. The result is compared with two columns, running with the (default) SP mode and the original Hadoop.

Table 3.3 gives the overall job execution time and inter-cloud data traffic for each complex job. The result shows that the total amount of inter-cloud data traffic can be significantly reduced with the appropriate modes. For example, knowing that the first two

Table 3.3: Experimental results on chained MapReduce

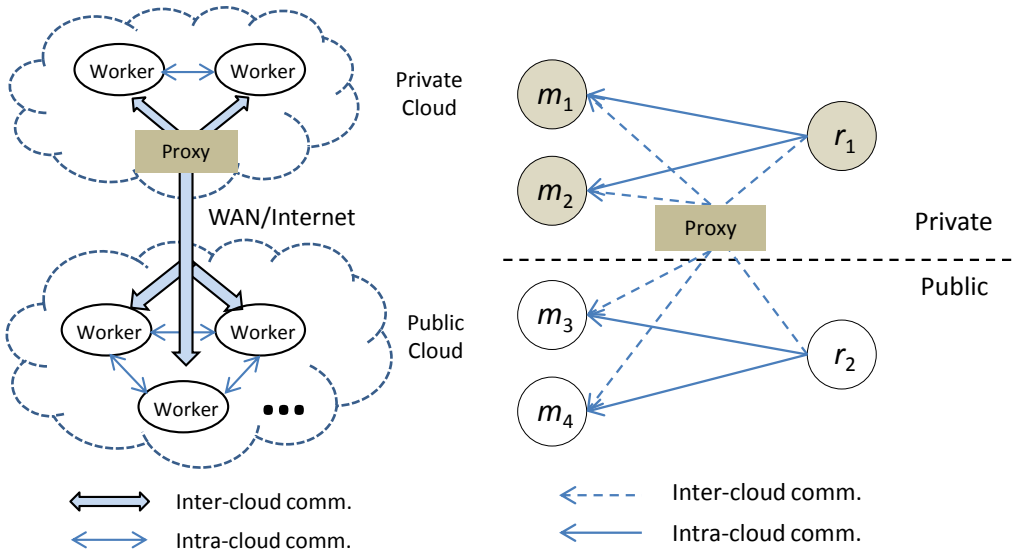
Application	Job	Hadoop		SP Mode Only		Assigned Modes	
		Time (sec)	Traffic (MBs)	Time (sec)	Traffic (MBs)	Time (sec)	Traffic (MBs)
Wordcount + Sort	Word Count	3020	36551	4082	32150	2275	685
	Sort	278	505	307	621	307	621
	Total	**3298**	**37056**	**4389**	**32771**	**2582**	**1305**
Face Anonymization	Face detection	5220	4109	5776	4521	5093	847
	Average	2158	1020	2534	1453	2064	0
	Sort	483	1007	528	1407	528	1407
	Total	**7861**	**6136**	**8838**	**7381**	**7685**	**2254**

jobs of face anonymization meet the unique tag property, we can assign to them the HO mode, where data are separately processed in the public and the private cloud in parallel. The sensitivity information is then naturally handed over to the next job, so there is no need to transfer the output of each job back to the private cloud. Such avoidance of unnecessary data movement leads to further optimization in inter-cloud communication across multiple MapReduce jobs. Overall, we can reduce the inter-cloud data traffic by more than 90% for the wordcount+sort job and around 70% for the face anonymization job as compared to the SP mode or original Hadoop.

The total elapsed time also has significant improvements with correctly chosen modes compared to the SP mode. The times are also comparable to the original Hadoop runs. We remark that choosing the mode can be done automatically by the system if the properties of the MapReduce job are specified. In summary, the total overheads for the hybrid framework are reasonable for realistic complex MapReduce jobs in the hybrid cloud setting with data confidentiality constraints. We believe that in many cases, the conditions for the non-single phase modes can be met which lead to further optimizations.

3.8 Extension – Routing Traffic through a Proxy

Recall that under our security model in Section 3.5, adversaries in the public cloud have knowledge about the identities (e.g., IP addresses) of the private servers who interact with the public cloud (i.e., the information I.2 as defined in Section 3.5.2). Through analyzing the timing and size of packets received from individual private servers, one could



(a) High-level view from workers

(b) Low-level view from tasks

Figure 3.21: Routing inter-cloud traffic through a trusted proxy to prevent side-channel leakage on private worker identities.

potentially collect additional information about the input. In Section 3.5.7 we discussed multiple mechanisms to prevent such side-channel information leakage. To facilitate the security analysis and remove the leakage of private worker identities, we implement the mechanism of routing inter-cloud data traffic through a trusted proxy server.

3.8.1 Main Idea

The intension is to hide the identities of the private servers. To this end, we add one more proxy server in the private cloud, as illustrated in Figure 3.21(a), whose responsibility is to route the traffic across the public and the private cloud. Workers in one cloud can only contact the proxy server to retrieve the output of workers in the other cloud. Traffics within the public or the private cloud are as usual. Figure 3.21(b) provides a low-level view of the dataflow during a MapReduce execution from the perspective of tasks.

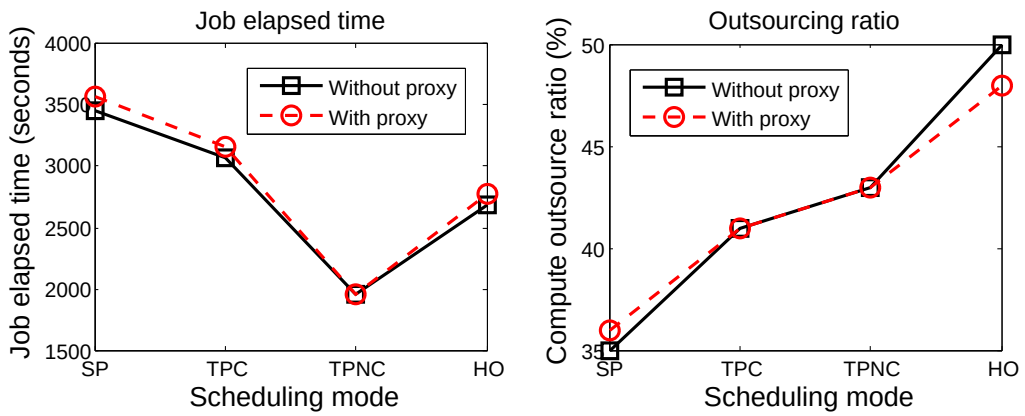


Figure 3.22: Overheads of routing inter-cloud data traffic through a proxy.

3.8.2 Implementation and Evaluation

We implemented the above mechanism as an extension to our tagged-MapReduce prototype. Essentially, for each job, the proxy maintains the locations of its map and reduce tasks. Such information can be easily obtained from the JobTracker. Whenever a reduce-task contacts the proxy, it knows exactly where to retrieve the corresponding map-tasks' output. The proxy then copies the data and forwards them to the requesting reduce-task "on the fly".[12]

Experiments are conducted to evaluate the overheads incurred by the proxy. Specifically, we run the word count job under our prototype system, *with* and *without* the proxy enabled. The hybrid cloud contains 3 private and 3 public nodes as described in Section 3.7.2. The proxy is deployed on an additional standard large instance in the private cloud. The ratio of sensitive data over the input is 40%. We compare only the job elapsed time and computation outsourcing ratio since there will be no difference in the inter-cloud data traffic.

The result in Figure 3.22 shows that there are only small differences between the cases of using and not using a proxy. The time overhead of routing traffic through a proxy is less than 4% in our experiments. This is foreseeable since the original inter-cloud data traffic

[12]Relaying data on the fly can generally reduce the overhead but introduce difficulties in fail-over. Alternatively, one can start to forward the data only after it is fully copied to the local storage.

needs to go through some gateway anyway. Hence, we conclude that, routing inter-cloud data traffic through a trusted proxy server is an effective approach to prevent side-channel leakage, with only negligible overhead in the job execution time.

3.9 Discussion

Comparison with Sedic

Sedic [175] is closely related to our work but with fundamental differences. It takes a different sanitization approach whereby data are duplicated to both clouds, but with sensitive portions sanitized in the public cloud. This approach, however, is less flexible for complex MapReduce computation with chained or iterative MapReduce. We address this problem by explicitly tagging. With tagging, data directly carry sensitivity information which can be fed to the next job, and thus multiple MapReduce computation can be carried out naturally. This flexibility also allows legacy MapReduce code to be easily supported. Besides flexibility issues, the sanitization approach also reveals relative locations and length of sensitive data, which potentially could leak important information [118]. In contrast, data in our framework are segregated according to their sensitivity and distributed to the two clouds separately. Since the segregation of data does not explicitly reveal the locations of sensitive data, the propose approach is arguably more secure. Furthermore, tagged-MapReduce is also more expressive. Sedic can be expressed as a special case of our model (with a single-phase mode and default tagging policy) but tagged-MapReduce programs with expressive security policies and sensitivity downgrading are not catered to in Sedic. In addition, Sedic does not consider the problem of a general security framework for analyzing of data leakage on a hybrid cloud which we do.

Practical Considerations

The idea of tagging data with different sensitivity levels for security purposes has been studied in many scenarios [135, 147, 175] and is also implemented in the industrial com-

munity. For example, Oracle introduced the Oracle Label Security[13] (OLS) as an extension of the database where each row can be associated with a multi-level security label (e.g., unclassified, confidential, sensitive, highly-sensitive). Apache Accumulo [8], a distributed key/value store based on Google's BigTable [57], has even finer cell-level access labels where a label is a logical combination of any user-specified strings. Access to the data is controlled by comparing the data label with the requesting user's label or security clearance. We remark that the tagging approach, although conceptually simple, is a practical approach for data access control in a complex environment such as a hybrid cloud. While the tag in our framework is only binary: sensitive or non-sensitive, it can be extended to other types such as the labels used in Accumulo. More sophisticated labels can be useful in applications where data cannot be simply classified as "sensitive" and "non-sensitive", or situations where a piece of information can be considered as sensitive only within a specific context or only when it is associated with another piece of information.

Potential Improvements

The design of the scheduling modes can also be improved. For example, the experimental study on job elapsed time gave a rough "V-shape" curve over the sensitive data ratio (e.g., the word count job in Figure 3.14). With a sensitive ratio lower than 50%, the time decreases when the sensitive data ratio increases, which is counter-intuitive. This is because a "smart" scheduler can choose to push non-sensitive data to either the private or public cloud, and thus the performance of this "smart" scheduler with x-percent sensitive data should not be worse than that with y-percent sensitive data if $x < y$. This observation indicates a potential improvement to the scheduler: by dynamically monitoring the ratio of sensitive/non-sensitive data and the processing capability of both clouds, a scheduler could move some computation on non-sensitive data to the private cloud to achieve optimal load-balancing and get a constant job execution time when the sensitive ratio is below 50%. However, such "smart" scheduler may not be easy to realize.

[13] http://www.oracle-base.com/articles/9i/oracle-label-security-9i.php

Our current implementation requires programmers to indicate whether a map and reduce function is partition-able or unique-tag. However, the specified properties may not be correct due to programmers' misunderstanding or limited knowledge of the programs. An interesting question is whether it is possible to apply certain program analysis techniques to automatically extract the properties from a given MapReduce program. However, the question of whether this property is achievable remains open.

3.10 Summary

In this chapter, we present a general solution for computing in the hybrid cloud by extending MapReduce with sensitivity tags. Our goal is to give a simple and generic framework for programmers who are already familiar with MapReduce and want to have data privacy awareness in the hybrid cloud setting. However, it allows complex MapReduce programs which can have sophisticated security policies as well as chained MapReduce jobs. We also pair it with a general security framework to analyze what kind of security leaks can occur through execution in the hybrid cloud. It considers subtle cases when there can be a difference between the programmers' views of the MapReduce programs versus information leakage that may be gained by the public cloud during actual execution. Our security model can be used to compare and determine what additional information a scheduler leaks over the baseline scheduler.

We exploit properties of the map and reduce functions which allow the scheduler to optimize and rearrange the computation, pushing more work to the public cloud while reducing the inter-cloud communication. We present scheduling modes which can utilize these properties while being secure under our model. Our experiments demonstrate the importance of the optimizations. Tagged-MapReduce in the hybrid cloud only incurs small overheads as compared to a Hadoop run which ignores the data confidentiality and security constraints, and thus is fairly practical. In addition, the framework can also handle legacy MapReduce code naturally.

Chapter 4

Privacy-preserving Video Surveillance Stream Processing on Hybrid Clouds

4.1 Introduction

This chapter studies the issues of processing large-scale video surveillance streams on the hybrid cloud.

Video surveillance systems, as an effective means to ensure safety of individuals and communities, have been widely deployed in various environments like homes, shops and banks etc. Video surveillance systems are inherently data-intensive, and often compute-intensive with the needs to carry out various computation like transcoding, indexing and video analysis. Such computational requirement could be seasonal, for example, heavier workload in the morning of workdays while lighter workload during weekend nights, as observed in typical video streaming systems [174]. An organization's in-house private datacenter may be overloaded during peak hours due to its limited computing capability. While with cloud computing, it is possible to offload all the video streams and computation to a public cloud like Amazon AWS, such strategy can incur high monetary cost [127]. More importantly, video surveillance streams often contain sensitive information that cannot be directly handled on the public cloud due to potential data leak-

ages [30, 141].

The hybrid cloud turns out to be a feasible solution where one can push out partial video streams to the public cloud while keeping those sensitive streams in the local private cloud, addressing both of the aforementioned issues on seasonal workload and security. However, the scheduling decisions of how much and which part of the computation to outsource is generally hard to make due to frequent changes of system status (e.g., system configuration and stream sensitivity) as well as multiple factors to consider including costs, performance and security etc. Also, from application developers' point of view, it is preferred that they only need to specify how computations are to be carried out, without caring about where they are executed and how data are moved during execution. Such transparency makes it easier for developers with no experience in parallel/distributed systems to write applications working on large clusters. Therefore, it is desired to have a middleware that can unify the two clouds and effectively schedule the processing on large video surveillance streams.

Many stream processing systems have been developed in the past few decades, from centralized settings like Aurora [63] to distributed settings like Borealis [32], Nephele [117], S4 [128] and Storm [16]. Together with these systems, there are a large number of studies focusing on scheduling among multiple servers, with various goals such as to minimize the end-to-end application latency [113, 172], to maximize the aggregated throughput [36, 112], to optimize a combination of latency and throughput referred to as network-usage [35, 136], to balance the workload and resource usage among all servers [40, 76], or to maximize the reuse among multiple queries [111, 153]. Although our problem can be treated as a special case of some known general scheduling models, its specialized settings can be exploited for more effective solutions, making it scalable to larger instances. Observed that in our setting, servers within each cloud are typically connected by a high-bandwidth, low-latency network (e.g., Gigabit LAN), whereas connections across the two clouds have to go through a wide area network or the Internet, having relatively smaller bandwidth and higher latency. Also, according to today's typical cloud pricing mod-

els [25], data transmission within each single cloud is free-of-charge while data traffic across the two clouds incurs high monetary cost, e.g., $0.19/GB. Hence, we can group and treat the servers in our setting as two servers: one private server with a fixed amount of computing power and one public server with elastic resources, and connections between these two servers are costly. In addition, in many scenarios the effect of public cloud's computation cost is much lower than the inter-cloud communication cost [25]. Therefore, we can stress less on the public servers' computation cost in the cost model. The security requirement places another hard constraint on where certain streams can be processed. These specialized properties in turn allow us to focus on the processing of larger number of streams (e.g., hundreds) with reasonable length of tasks, e.g., around 10 operations per task.

We model stream processing as a set of task templates whereby each template can be independently instantiated to multiple video streams. Each task template is represented as a loop-free, directed graph of operations, with the code provided by application developers. In addition, the developers can specify multiple connection points [63] in a task graph whereby clients can dynamically tap into the stream data during execution. The locations of the connection points provide useful information to the scheduler, so that dynamic changes to the task graphs do not necessarily trigger rescheduling. However, as sensitivity of video streams can change during run-time, rescheduling might be required occasionally. In particular, if the sensitivity of a stream in the public cloud changes from non-sensitive to sensitive, the stream must be rescheduled to the private cloud to prevent potential data leakages. This can be done by buffering or dropping data frames before the rescheduling is completed.

We formalize the scheduling problem as an optimization problem that minimizes the overall monetary cost to be incurred on the public cloud, with the resource, security and Quality-of-Service (QoS) constraints. We propose an algorithm that exploits the aforementioned specialized properties of hybrid clouds for more efficient solutions. Essentially, for each task template of the input, we search for the set of "minimal configurations"

and then employ integer programming to select the desired configurations. For templates that are reasonably short (around 10 operations), the set of minimal configurations is typically sufficiently small for state-of-the-art solvers [125]. In cases where the number of minimal configurations is large, we provide a heuristic that selects only a few representatives to further improve the performance. Informally speaking, the challenge in our scheduling problem is more on determining how a large number of relatively short tasks are to be scheduled on the two servers, instead of scheduling a single large task among multiple servers considered by many existing works.

The proposed stream processing model and scheduling mechanism can be built on top of existing stream processing systems like Storm [16]. To facilitate experiments and testing, instead of using existing platforms, we implemented a proof-of-concept system with basic functionality of video streaming and several operations including transcoding, face detection, etc. We conducted extensive experiments, through both large-scale simulations and actual runs with our proof-of-concept system on Amazon EC2. The result shows that it is feasible to process large-scale video streams in a hybrid cloud, preserving data-privacy and reducing monetary cost as compared to a pure public cloud deployment. The overheads of our scheduler are much lower than other alternatives.

4.2 Background on Video Surveillance

We first give a brief introduction to traditional video surveillance systems as well as the current moving trend to the cloud computing environment. We then summarize traditional approaches to preserving privacy in surveillance videos.

4.2.1 Video Surveillance Systems

Video surveillance systems, as illustrated in Figure 4.1, use multiple cameras to observe the events and activity of an area. They are often connected to multiple recording devices, and may be inspected by security personals. Over the past few decades, video surveillance

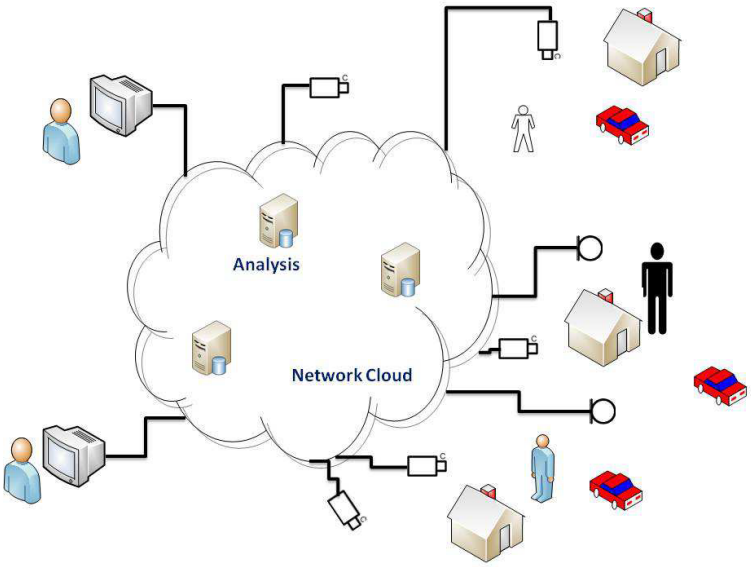


Figure 4.1: Illustration of a video surveillance system [150]. Video cameras (and microphones) capture the events and activities of the environment.

systems have grown from the original closed-circuit video transmission (CCTV) environments into the self-contained digital video recorder (DVR) environments, and now into the centrally managed Internet Protocol (IP) cameras. IP cameras can be easily deployed on current computer networks, and video streams can be sent to anywhere on the Internet including mobile devices and smartphones.

Video surveillance systems have been deployed in a wide spectrum of scenarios like traffic surveillance in cities, detection of military targets, and physical perimeter security [139]. Modern video surveillance systems often involve a large number of cameras (Figure 4.2). For example, the city of Chicago, Illinois, used a $5.1 million Homeland Security grant to install an additional 250 surveillance cameras, and connect them to a centralized monitoring center, along with its preexisting network of over 2000 cameras, in a program known as Operation Virtual Shield [159]. As another example, the China's Golden Shield Project aims to install millions of surveillance cameras throughout China, along with advanced video analytic and facial recognition techniques, which will identify and track individuals everywhere they go [107].

Video surveillance systems used to require human personnel to monitor camera footage,

(a) (b)

Figure 4.2: Multiple cameras installed at a single site.

which now becomes difficult due to the large number of cameras. Automated video analysis and understanding techniques are therefore needed to ease the workload of human operators and reduce the risk of miss-detection. Most of current surveillance systems employ various video analysis techniques, including motion/object detection, objection classification, objection tracking, behavior and activity analysis and understanding, face detection and person identification etc. Moreover, video surveillance systems are also preferred to be interactive in the sense that operators can feed in parameters and control commands during runtime. For example, once a particular object is detected in one camera, the operator can instruct the system to detect whether the same object appears in any of the other 100 cameras in the next 10 minutes. A large number of automated video surveillance systems have been proposed over the past few decades, from research prototypes [69, 86, 89, 101, 106, 167] to industry productions [11, 13, 14].

4.2.2 Video Surveillance in the Cloud

Traditional surveillance systems send video streams to a centralized location, e.g., an in-house private datacenter, whereby video data are analyzed and stored. However, as computation in video surveillance system becomes increasingly intensive, the local system could be overloaded during peak hours due to its limited computing capability. With recent advances in cloud computing, one natural solution is to offload all video streams and computation to a public cloud like Amazon AWS. There are a number of works dis-

cussing such possibility of transitions.

Karimaa [103] reviews the transition alternatives from a traditional to a cloud-based surveillance system, points out the advantages and analyzes the dependability characteristics, namely, availability, security, reliability and maintainability. Chang et al. [58] propose an architecture for ubiquitous video surveillance (UVS) services over the Internet. To protect privacy against the public Internet, they apply various techniques including encryption, multicast overlay network and forward error correction (FEC). Saini et al. [149] investigate how to dynamically balance the workload of video processing over multiple servers on the cloud. Li et al. [116] investigate how to process massive floating car data (FCD) in cloud environments, exploiting emerging cloud computing technologies to solve data-intensive geospatial problems in urban traffic systems. Pereira et al. [133] present Split&Merge, a MapReduce-based architecture for high-performance video processing (i.e., encoding) on the cloud. Neal et al. [127] investigate the economy of moving video surveillance to the cloud. They conclude, after examining various SaaS, PaaS and IaaS providers that, it is more expensive and requires additional reviews for legal implications as well as emerging security threats. Our work will demonstrate that, with the hybrid cloud and effective scheduling, the above issues can be significantly mitigated.

4.2.3 Security and Privacy in Video Surveillance

Video surveillance provides increased safety for individuals and community. However, this increased safety also comes at the cost of privacy loss of individuals. Privacy concerns have prohibited video cameras from being deployed at many critical places that are needed to be monitored. Existing works on video privacy can be broadly summarized as the following two steps: privacy modeling, followed by data transformation.

Privacy Modeling To protect video data, the first step is to understand what characteristics of a video cause privacy loss. Generally, existing work on privacy modeling can be divided to two categories: sensitive information as privacy loss and identity as privacy

loss. In the first category, it is assumed that the identity is known through other means and the semantic information of the video, such as the activity or location of an object, causes privacy loss [33, 114, 160, 177]. For instance, Zhao et al. [177] filter the videos so that the individuals are identified, but other sensitive information like where they are, what they are doing, or who are with them, cannot be detected.

In the second category, it is assumed that if the adversary can recognize a person in the video, it leads to privacy loss. With this assumption, they model the privacy loss in terms of the characteristics of the video which reveal the identity of the person. This is the most common approach of privacy modeling in video surveillance community [48, 60, 95, 164, 169, 176]. For example, Wickramasuriya et al. [169] define four levels of privacy: original image, blurred silhouette, monotonically colored silhouette, and bounding box. Venkatesh et al. [164] use object detection to determine the bounding boxes covering the whole human body and replace these regions with background. However, identity loss can still occur when the human silhouette or face is obfuscated, through implicit channels like priori-knowledge or environments/activities in the video. Saini et al. [148] provide a more comprehensive privacy modeling considering both explicit and implicit channels.

Data Transformation Once the characteristics of a video are identified that cause privacy loss such as image regions or event sequences, the next step is to transform the video data such that the privacy can be protected. One trivial solution is to remove everything from the images, but such video has no utility. Most researchers have used selective obfuscation to preserve the privacy in the surveillance videos [55, 98, 109, 152]. They adopt the traditional approach which first detects the region of interest (e.g., face) and then hides it. Since this approach is limited by the accuracy of the detectors, privacy cannot be guaranteed. The other set of works go for global transformation of the whole image [33, 48, 114]. In these works, the obfuscation function (blurring, pixelization etc.) is applied on the whole image to hide the private information. This approach is too pessimistic and affects the utility of the video data. Chinomi et al. [64] give a survey comparing different methods

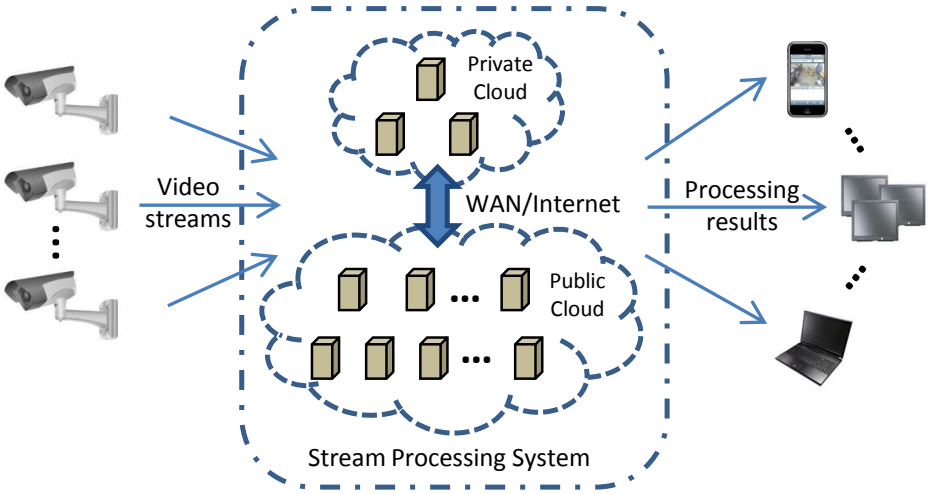


Figure 4.3: System model for hybrid cloud video surveillance.

for obscuring people in the video data, including monotone, blur, pixelization, border and etc.

4.3 Hybrid Cloud Video Surveillance Model

The above traditional approaches for video privacy are generally expensive for large-scale video surveillance data with the real-time requirement. In this work, we assume that video surveillance streams can be tagged in some way indicating the sensitivity level and the privacy is achieved by preventing sensitive streams from being processed on untrusted servers. Now we start to describe our model for the hybrid cloud-based video surveillance system.

4.3.1 System Model

We consider a hybrid cloud model, as shown in Figure 4.3, for video surveillance systems with highly dynamic workloads. In this model, the private cloud has a fixed number of servers, each of which has limited computing power. In contrast, the public cloud has elastic computing resources that can be allocated and de-allocated on-demand. Servers within each cloud are connected to each other by a high-bandwidth, low-latency network

(e.g., Gigabit LAN) whereas connections between servers across the two clouds have to go through a WAN or the Internet. Hence, the inter-cloud connections have relatively smaller bandwidth and larger delay. In addition, under current typical cloud pricing models, data transmission within each single cloud is free-of-charge while data traffic to the Internet (i.e., inter-cloud data traffic) incurs high monetary cost, e.g., \$0.19/GB [25]. Based on these observations, we group and treat the servers in our system as two servers: a private server with a limited aggregated computing power, denoted as C, and a public server with elastic computing capacity. These two servers are connected by a long-distance link with an estimated bandwidth B and link delay L, while data transfer within each server incurs no cost.

The system contains a large number of surveillance cameras distributed over a wide area. Each camera generates a video stream that is to be sent to some servers in the two clouds. The stream is processed in the hybrid cloud, in a way specified by application developers, and then streamed to multiple clients. Besides, input/output streams can also be originated from/written to storage servers. Note that both the cameras and clients can be within or outside the local area network of the private cloud. Hence, streaming to the public and private servers incur different costs.

4.3.2 Stream Processing Model

The system has to process a large number of video surveillance streams in parallel. We model the processing as a set of *task templates* where each task template consists of a sequence of operations that can be applied to multiple input streams. Similar to many previous works [63,94,165,172], each task template T is represented as a directed, acyclic and labeled graph $G(V, E)$ where $V = V_{src} \cup V_{op} \cup V_{sink}$. The set V_{src} is the set of stream sources which could be cameras or recorded videos retrieved from storage systems; V_{sink} is the set of streaming sinks which could be display devices or storage systems as well. V_{op} contains the set of *operations* such as transcoding, background extraction, object detection etc. Application developers can provide the code for each operation or

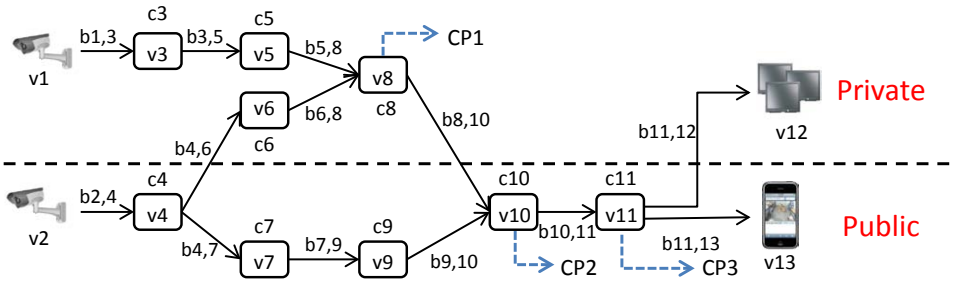


Figure 4.4: Illustration of a stream processing task.

select it from a library. The edges in E define the data flows between the vertexes in V. Figure 4.4 gives an example of task template. Recall that a single template can be independently instantiated to multiple input streams. In an *instantiated task*, each source and sink node is associated with a location, e.g., IP address, indicating in which cloud, private or public, the node resides, while operations in V_{op} have not been assigned. In contrast, in an *assigned task*, not only the source and sink nodes are instantiated, each operation in V_{op} is also assigned a label, *private* or *public*, indicating in which cloud the operation is to be executed.

Similar to Aurora* [63], an application developer can specify multiple connection points in a task graph where data streams will be cached in some storage for a certain amount of time e.g., 10 hours. Such connection points are useful in supporting ad-hoc queries and dynamically joined clients. For example, one might be interested in finding out whether a particular person appeared in a building yesterday evening between 6-12pm. Such a query can be similarly defined and attached to the connection points of existing running tasks that have the required data, without rescheduling the whole set of tasks. The introduction of connection points is especially useful for our system since rescheduling and operation migration are more expensive in the hybrid cloud setting.

4.3.3 Security Model

We consider the servers on the public cloud to be *honest-but-curious*, that is, they will follow the protocol and carry out the required computation honestly, but may retain in-

formation collected from the computation for malicious purposes. In contrast, the private servers are fully trusted.

Every stream in an instantiated task is tagged with an attribute, *sensitive* or *non-sensitive*, while in general a larger attribute set is also possible. The tagging can be done manually by the users or by an automated program. In its simplest form, one could specify certain policies for deciding the sensitivity. For example, "the video stream generated by camera x is sensitive iff the time is between 2–4pm"; "the output of operation o is sensitive if at least one of its input is sensitive" etc. In this work, we do not discuss in detail how streams are to be tagged, and simply assume that there is a component (i.e., the Sensitivity Analyzer in Figure 4.5) to do this tagging when instantiating the task templates to different stream sources and sinks. Sensitive streams must not leave the private cloud in order to prevent data leakages while there is no constraint for non-sensitive streams. Unless otherwise specified, all streams in an instantiated task are tagged.

Note that not all possible ways of tagging streams are valid. If all the input streams of an operation are non-sensitive, the output stream has to be non-sensitive as well. We refer it to as the *non-upgrading policy*. The non-upgrading policy imposes a constraint which excludes certain undesired scenarios, e.g., excluding cases where non-sensitive streams that have been pushed to the public cloud are later tagged to be sensitive. On the other hand, it is possible that on sensitive input, the output is non-sensitive. For example, an operation that takes in a sensitive video stream may output a lower resolution stream that is deemed as non-sensitive. We refer it to as the *downgrading policy*. This policy allows pushing more computation to the public cloud which is useful in many scenarios.

The sensitivity of a stream can change during runtime, from sensitive to non-sensitive or vice versa, and a rescheduling might be required due to such realtime changes. In particular, if a stream assigned to the public cloud suddenly becomes sensitive, it must be rescheduled to the private cloud so as to meet the security requirement. Data frames have to be properly "buffered" or dropped before the rescheduling is fully carried out. This will introduce certain performance overhead, either in terms of extra delays or data loss-

es. Fortunately, there are existing techniques supporting fast operation migration during runtime [63, 136].

4.3.4 Cost Model

Each operation in a task graph implements a video processing function, requiring a certain amount of computing power, denoted as c, to generate the output *in realtime*. In this work, we measure computing cost in terms of the number of ECUs (Amazon EC2 Compute Unit, each ECU provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or Xeon processor). We assign the cost of 1 ECU to an operation if the operation can be carried out in realtime by a machine with 1 ECU capacity. The computing cost can be estimated based on the input streams' frame size and data rate, combined with pre-conducted resource profiling baselines for the operations. Each connection in the graph represents a data flow from one operation to another, which requires a certain amount of bandwidth, denoted as b, to transfer the data in realtime. The bandwidth cost is measured in MB/s, which can be similarly estimated from the user-desired stream rate and frame size.

One advantage of our cost model is that, it directly approximates monetary cost, giving system administrators a good overview of the projected cost. Potentially, this cost model could also lead to further cost-saving. For example, to handle an operation of 0.75 ECU requirement, one can allocate a virtual machine of 1 ECU capacity instead of 2, since the latter usually costs more (e.g., $0.08/hour vs. $0.16/hour by Amazon).

4.3.5 System Architecture

Figure 4.5 shows the overall architecture of the proposed hybrid cloud video surveillance system. The Sensitivity Analyzer, provided by application developers, takes as input a set of task templates and the corresponding source and sink locations, evaluates the sensitivity of the streams, and outputs a set of instantiated tasks (whose streams are all tagged).

The instantiated tasks are then fed into the Scheduler, together with the information of

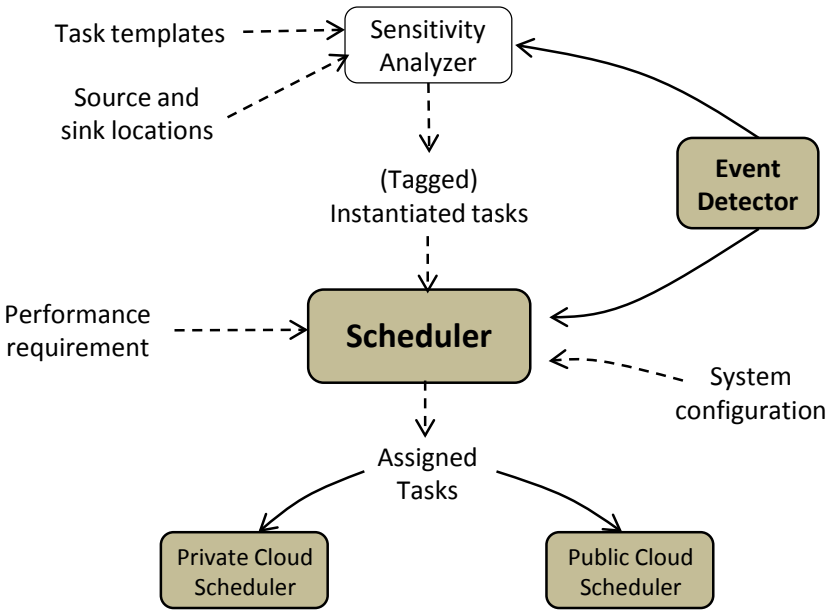


Figure 4.5: Architecture of the hybrid cloud video surveillance system.

performance requirement and system configuration including private cloud's computing power, inter-cloud bandwidth and link delay etc. Based on these inputs, the Scheduler decides how to assign the operations in each task to the two clouds.

Each cloud, public or private, has an intra-cloud scheduler. Scheduling within the public and private cloud could be different, and have different objectives. As scheduling within a single cluster is not within the scope of this work, in our experiment, we use a simple greedy algorithm that always picks the next available server.

The Event Detector is responsible for detecting changes in the task graph structure (i.e., dynamic clients and ad-hoc queries), changes in stream sensitivity as well as in other system configurations. Whenever a rescheduling is required due to such changes, the Event Detector notifies the Scheduler to initiate the rescheduling.

4.4 Problem Formulation

Given a set of stream processing tasks, we want to decide the proper assignment of each operation to the public or private cloud. As mentioned before, although our scheduling problem can be treated as a special case of some known general scheduling models, its

specialized "two-server" setting could lead to more efficient solutions and allow scaling up to larger instances.

To meet the security requirement, one natural approach is to assign tasks with (any) sensitive streams to the private cloud and tasks with only non-sensitive streams to either the public or private cloud. This approach, while being simple to implement, has two main issues in terms of scalability and cost. First, it does not scale well due to the short-sightness in using the private cloud resources. Second, this approach can also incur high monetary cost mainly due to the non-optimized inter-cloud bandwidth usage. Hence, a more scalable and efficient solution is desired. This section formalizes the scheduling problem, followed by a potential extension to the stream processing model. The proposed approach will be elaborated in Section 4.5.

4.4.1 Optimization Problem

Usage of public cloud resources incurs additional monetary cost, including both the compute and bandwidth cost. Essentially, given a set of tasks each consisting of multiple operations, our goal is to assign each operation to either the public or private cloud, such that the total monetary cost to be incurred on the public cloud is minimized, subject to the constraints that the private cloud cannot be overloaded, sensitive streams cannot flow into the public cloud and the QoS requirements can be met. Note that there could be other scheduling objectives such as minimizing the total amount of inter-cloud data traffic (which could potentially lead to improved performance). Fortunately, as shown in Section 4.6 later, minimizing the cost often gives good approximations to solutions that optimize the bandwidth usage, as bandwidth cost dominates within the total cost under our cost model.

The Scheduling Problem

The input is a sequence of task templates $\mathcal{T} = \langle T_1, \ldots, T_m \rangle$ and a sequence of integers $R = \langle r_1, \ldots, r_m \rangle$ where each template T_i is to be instantiated r_i times to different sources

and sinks. For ease of exposition, let us rewrite the input in the equivalent form of $\hat{\mathcal{T}}$ = $\langle \hat{T}_1, \ldots, \hat{T}_n \rangle$ where each $\hat{T}_i$ is an instantiated task, and $n = \sum_{i=1}^{m} r_i$. Each operation v_j^i in $\hat{T}_i$ is associated with a computing cost c_j^i and each connection from v_j^i to v_k^i in $\hat{T}_i$ requires a bandwidth cost b_{jk}^i. Let the QoS requirement be the maximum allowed end-to-end delay d_i for each $\hat{T}_i$. The scheduling problem is to decide the binary assignment x_j^i for each operation v_j^i in $\hat{T}_i$ (where value 0 and 1 corresponds to being assigned to public and private respectively), in such a way that the total incurred monetary cost on the public cloud

$$\alpha \sum_{i,j} c_j^i (1 - x_j^i) + \beta \sum_{i,j,k} b_{jk}^i |x_j^i - x_k^i| \tag{4.1}$$

is minimized, subject to the following constraints: (1) the private cloud must not be overloaded, i.e., $\sum_{i,j} c_j^i x_j^i \leq C$; (2) sensitive streams must not leave the private cloud; and (3) any task $\hat{T}_i$ with the corresponding assignment X_i can meet the delay requirement, i.e., $\forall i \in [1, n],\ Delay(\hat{T}_i, X_i) \leq d_i$. Details on the determination of the delay $Delay(\cdot, \cdot)$ will be discussed in the following section. Recall that in the input, the source and sink nodes for each task are already labeled to be in either the public or private cloud and thus cannot be reassigned.

Determining α and β

In the above objective function 4.1, the first term represents the computation cost on the public cloud and the latter represents the bandwidth cost for inter-cloud data transmission.[14] Since we are handling stream data, we measure the cost rate, e.g., dollars per hour. The parameters α and β represent the unit-price for computation and bandwidth respectively, whose values could be determined according to the pricing model of the cloud provider. Taking Amazon EC2's pricing model as an example [25], each ECU costs \$0.08/hour and inter-cloud bandwidth usage costs \$0.19/GB; hence, α and β can be set as 0.08 and 0.684 accordingly. Due to the huge volumes of video data, computation cost

[14]Similar to the previous work, we assume that both two-way traffics incur monetary cost. However, the objective function can be easily changed to consider only the out traffic.

is typically much less than the communication cost. To illustrate, let us assume that an instance with 1 ECU is able to handle realtime processing of one high-definition video stream with 1 MB/s inter-cloud bandwidth requirement, then the cost would be \$0.764 for a 1-hour run (\$0.08 for computation + \$0.684 for bandwidth), where the computation cost is around one-eighth of the bandwidth cost. This suggests that minimizing only the bandwidth could give good approximations to solutions that minimize the monetary cost. This is verified in our experiments in Section 4.6. Hence, to speed up the scheduler, one could omit the computation cost in the cost model.

Estimating end-to-end delay

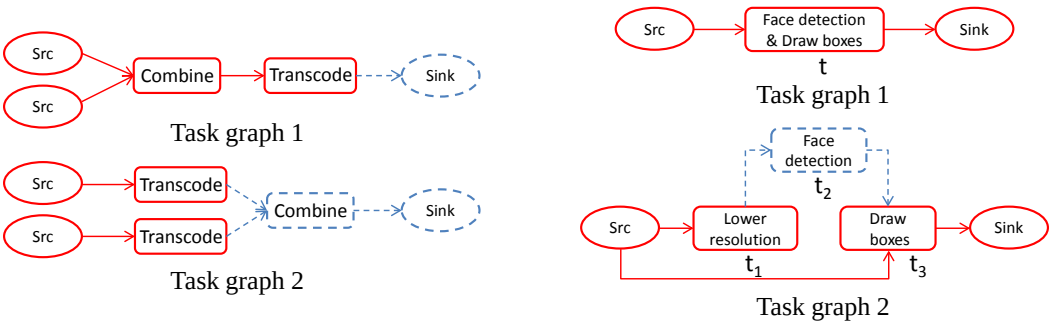
For a task graph, let us call the dataflow from any source to any sink node a *path*. Along a path P with corresponding assignment X for its operations, the total delay is the sum of the processing time and the communication latency, that is,

$$\sum_{\forall i, v_i \in P} proc\_t(v_i) + \sum_{\forall i,j,v_i,v_j \in P, |x_i - x_j| = 1} b_{ij}/B + \sum_{\forall i,j,v_i,v_j \in P, b_{ij} > 0} |x_i - x_j| L$$

where the first term represents the total processing time, the second represents the time for data transmission across the two clouds, and the third represents the total amount of network latency. Recall that the intra-cloud communication is assumed to incur no delay; however, it can be included in the calculation if required. Then, we have $Delay(T, X) = \max_{P_i} Delay(P_i, X_i)$. Hence, we can estimate the end-to-end delay for each assigned task.

4.4.2 Extension of the Stream Processing Model

Our scheduler can also handle the variation where there could be multiple ways to carry out a task. In this variation, an application developer can specify multiple template graphs for a task whereby these graphs are considered to be "functionally equivalent". To illustrate, let us consider the task shown in Figure 4.6(a) which could be carried out in



(a) Example task 1

(b) Example task 2

Figure 4.6: Illustration of the case where there could be multiple ways to complete a task.

2 different ways: (1) first combine the two streams and then lower the resolution; or (2) first lower the resolution of each stream and then combine. Although the results might not be exactly the same, one may consider the differences to be acceptable and deems the two processes to be functionally equivalent. Consider another example as shown in Figure 4.6(b), the task of performing face detection and drawing boxes on detected faces on a high-resolution video stream can also be carried out in another way: first, transcodes the high-resolution video stream to a low-resolution stream; performs face detection on the low-resolution stream; and then draws boxes on the original high-resolution stream. Note that face detection can achieve high accuracy on low-resolution videos [178]. If the low-resolution stream is tagged as non-sensitive, and the above two ways are specified as functionally equivalent, when necessary, the scheduler can push face-detection to the public cloud to reduce load in the private cloud.

For each task T, our proposed approach will consider all the possible alternative graphs $G_1, \ldots, G_k$ when making the scheduling decisions.

4.5 Proposed Approach

Not surprisingly, the scheduling problem defined in Section 4.4 is NP-hard.[15] Nevertheless, by pruning, we are able to handle fairly large instances. Essentially, for each task template of the input, our algorithm searches for the set of "minimal configurations" with

[15]This can be proved by a reduction from the 0-1 knapsack problem.

respect to the private cloud computation load and the public cloud monetary cost, and then employs integer programming to select the desired configurations. For a task template of reasonable length, e.g., with 10 operations, although there are 2^{10} configurations, typically it can be pruned down to around 20, which is sufficiently small for current solvers. For larger templates, we provide a heuristic to further reduce the number of configurations.

4.5.1 Transforming to Integer Programming

A task template with t operations gives 2^t ways of assigning its operations to the two clouds. Let us call each assignment a *configuration* and denote it as $f = \langle f^p, f^q \rangle$ where f^p and f^q are the set of operations assigned to the private and the public cloud respectively. Let us denote $\mathcal{F}(T)$ as the set of all configurations for a task template T.[16]

For each configuration $f \in \mathcal{F}(T)$, we can calculate a 2-tuple load-cost value (a, b) where a is the computing load on the private cloud and b is the cost that one wants to minimize. For our choice of the objective function, b is the monetary cost to be incurred on the public cloud. More specifically,

$$a = \sum_{\forall i, v_i \in f^p} c_i$$

and

$$b = \alpha \sum_{\forall i, v_i \in f^q} c_i + \beta \sum_{\forall i,j, v_i \in f^p, v_j \in f^q \ or \ v_i \in f^q, v_j \in f^p} b_{i,j}$$

We can also estimate the end-to-end latency $\ell(f)$ for each f using the $Delay(\cdot, \cdot)$ function described in Section 4.4.1. Additionally, let $\mathcal{F}(\mathcal{T}) = \mathcal{F}(T_1) \cup \cdots \cup \mathcal{F}(T_m)$.

The scheduling problem described in Section 4.4 can be easily transformed to the following integer programming problem:

[16] If T has multiple alternative graphs $G_1, \ldots, G_k$, $\mathcal{F}(T) = \mathcal{F}(G_1) \cup \ldots \cup \mathcal{F}(G_k)$.

Integer Programming

Given the input $\mathcal{T} = \langle T_1, T_2, \ldots, T_m \rangle$ and $R = \langle r_1, r_2, \ldots, r_m \rangle$ where each task template T_i is to be instantiated to r_i streams. We want to find x_i, the number of times a configuration f_i in $\mathcal{F}(\mathcal{T})$ is to be instantiated, such that the total monetary cost,

$$\sum_{\forall i, f_i \in \mathcal{F}(\mathcal{T})} b_i x_i$$

is minimized, subject to: (1) the private cloud resource constraint, i.e., $\sum_i a_i x_i \leq C$; (2) the number of instances constraint, i.e., $\forall j \in [1, m]$, $\sum_{\forall i, f_i \in \mathcal{F}(T_j)} x_i = r_j$; (3) the security constraint, i.e., if $x_i > 0$, then the corresponding configuration f_i does not push sensitive streams to the public cloud; and (4) the QoS constraint, i.e., $\forall j \in [1, m]$, $\forall i$, if $f_i \in \mathcal{F}(T_j)$ and $\ell(f_i) > d_j$, $x_i = 0$. This is an integer programming problem [170] with $|\mathcal{F}(\mathcal{T})|$ unknowns and about $3m + 1$ constraints.

4.5.2 Minimal Configurations

There are existing solvers for integer programming problems, for example, `LP_SOLVE`[17] and `CPLEX`[18]. However, integer problems with a large number of unknowns are difficult to solve in general. H. Mittelmann benchmarked existing solvers on a set of real-world testcases taken from the MIPLIB 2010 library[19] (with 338–18360 unknowns and 169–14163 constraints). The result showed that, current non-commercial solvers either fail or are only able to provide sub-optimal solutions within a 1-hour run; commercial solvers, although can solve most of the problems, take as long as tens to hundreds of seconds. Such performance certainly cannot meet the real-time requirement in typical video surveillance systems.

For a task template T, the set of all configurations in $\mathcal{F}(T)$ could be large. Fortunately, only a small number of them need to be considered. Let us consider two different

[17] http://sourceforge.net/projects/lpsolve/
[18] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[19] http://miplib.zib.de/

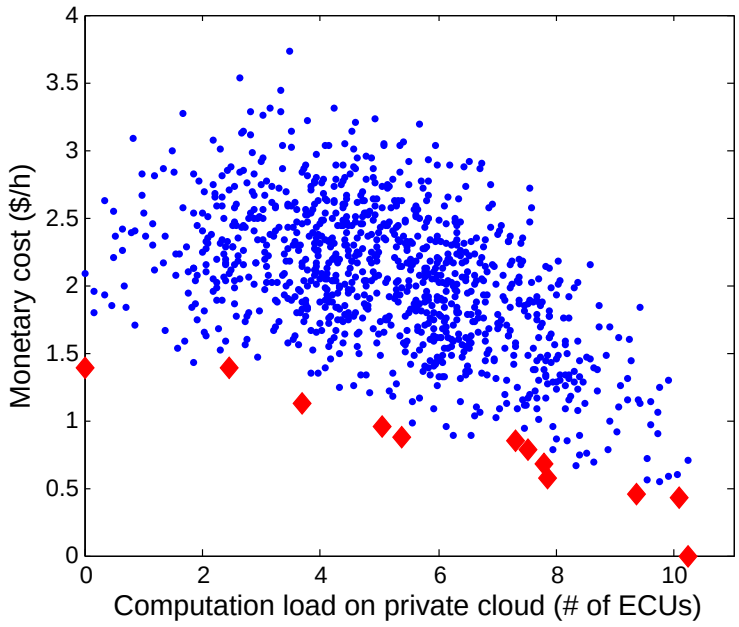


Figure 4.7: Configurations in the 2D load-cost graph. Those marked as dark red diamonds are the minimal configurations.

configurations f and $\widetilde{f}$ with respective load-cost value of (a, b) and $(\widetilde{a}, \widetilde{b})$ satisfying $a \leq \widetilde{a}$ and $b \leq \widetilde{b}$. Note that $\widetilde{f}$ will not appear in an optimal solution (otherwise, we can replace it by f, yielding a solution with smaller cost). Hence, consider the partial order $\preceq$ on $\mathcal{F}(T)$ where $f_i \preceq f_j$ iff $a_i \leq a_j$ and $b_i \leq b_j$, the optimal solution must be in the minimal configurations.[20] Figure 4.7 gives an example of minimal configurations, marked in the dark red diamonds. Let $\mathcal{MF}(T)$ be the set of minimal configurations for each T.

Study on the size of $\mathcal{MF}(T)$

To validate that $\mathcal{MF}(T)$ is typically small, we use the 13 task graphs created in Section 4.6.1 as templates, with a random computing cost within $(0, 2]$ ECUs to each operation and a random bandwidth cost within $(0, 1]$ MB/s to each connection. For the test purpose only, we assume that the source and sink nodes are in the private cloud. The values of α and β are set to be 0.08 and 0.684 respectively. The process is repeated 1,000,000 times for each template.

The result is shown in Table 4.1. Interestingly, for more than 95% of the instances,

[20]Minimal configuration is similar to the concept of Pareto frontier [28] in economics.

Table 4.1: Study on the size of minimal configurations $\mathcal{MF}(T)$.

Task Template	$\|V\|$	$\|\mathcal{F}(T)\|$	$\|\mathcal{MF}(T)\|$			
			min	**max**	**avg**	**95th percentile**
T_1	3	2^3	2	8	4.21	6
T_2	4	2^4	2	10	3.8	6
T_3	5	2^5	2	17	7.33	11
T_4	6	2^6	2	19	7.35	11
T_5	7	2^7	2	23	8.48	13
T_6	8	2^8	3	30	9.68	15
T_7	9	2^9	3	33	11.35	17
T_8	10	2^{10}	3	36	13.44	20
T_9	11	2^{11}	4	37	13.47	21
T_{10}	12	2^{12}	5	39	14.25	22
T_{11}	13	2^{13}	7	42	17.60	27
T_{12}	14	2^{14}	7	45	19.78	30
T_{13}	15	2^{15}	8	53	20.59	31

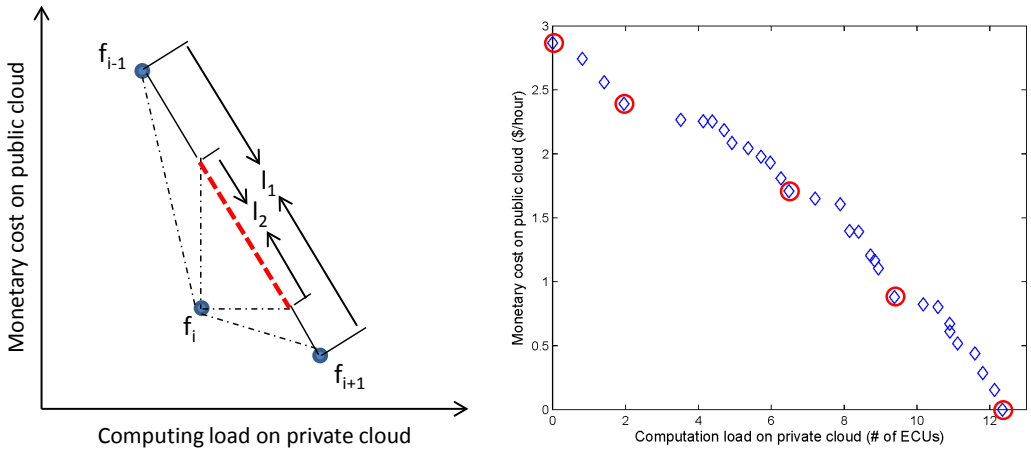
the size of $\mathcal{MF}(T)$ grows linearly rather than exponentially with respect to the number of operations. The maximal value of $|\mathcal{MF}(T)|$ observed in all the runs is only 53. This concludes that, for reasonable length of task graphs, $\mathcal{MF}(T)$ is sufficiently small for state-of-the-art solvers.

4.5.3 Heuristic Selecting Method

However, there could be cases where $|\mathcal{MF}(T)|$ is large, e.g., when T is extremely large. For such cases, we provide a heuristic to select a constant number ϵ (e.g., $\epsilon = 10$) of representatives among the minimal configurations. Different from using all the minimal configurations, the heuristic may not lead to optimal solutions.

As illustrated in Figure 4.8(a), let us consider three consecutive configurations f_{i-1}, f_i, f_{i+1} in $\mathcal{MF}(T)$, which form a concave curve. If both f_{i-1} and f_{i+1} contribute to an optimal solution (with n_{i-1} and n_{i+1} instances respectively), then the aggregated load-cost value (i.e., $(\frac{a_{i-1}n_{i-1} + a_{i+1}n_{i+1}}{n_{i-1} + n_{i+1}}, \frac{b_{i-1}n_{i-1} + b_{i+1}n_{i+1}}{n_{i-1} + n_{i+1}})$) may fall on the dotted line l_2 as shown in Figure 4.8(a), leading to a configuration that is greater than f_i under $\preceq$. In this sense, there is a good chance that both f_{i-1} and f_{i+1} are not in the optimal solutions, and can be replaced by f_i.

Let us define the ratio of l_2 over l_1 as the "likelihood" that f_{i-1} and f_{i+1} can be ex-



(a) Key observation

(b) Selection result ($\epsilon = 5$)

Figure 4.8: Illustration of the heuristic.

cluded. Our heuristic repeatedly picks the f_i with the largest "likelihood" among the current consecutive minimal configurations, until ϵ configurations are selected. Figure 4.8(b) illustrates the selection result of 5 representatives among a total of 31 minimal configurations. The configurations selected are marked in the red circles.

Effectiveness of the heuristic

To investigate the effectiveness of the heuristic, we use the task graphs T_8–T_{13} created in Section 4.6.1 as the templates. For each template, we set $r = 100$ and C to be ranging from 400 to 800 ECUs. Similarly, we assign a random computing cost within $(0, 2]$ ECUs to each operation and a random bandwidth cost within $(0, 1]$ MB/s to each connection. We compare the optimal cost derived from all the minimal configurations, and the optimal cost derived from the selected configurations (with $\epsilon = 5$). The result is shown in Table 4.2.

The empirical result shows that our heuristic is effective in selecting the most significant configurations, giving solutions that are mostly within 0.1% of the optimal ones of using all the minimal configurations.

Table 4.2: Effectiveness of the heuristic.

Template	Configuration Set	Private cloud computing power (# of ECUs)			Max Difference	Avg Difference
		400	600	800		
T_8	Minimal	94.920	65.829	36.739	0.03%	0.014%
	Selected	94.922	65.833	36.744		
T_9	Minimal	128.234	93.930	60.075	0.16%	0.070%
	Selected	128.440	93.981	60.075		
T_{10}	Minimal	142.152	106.684	71.171	0.14%	0.073%
	Selected	142.166	106.761	71.270		
T_{11}	Minimal	85.886	63.003	41.029	0.12%	0.051%
	Selected	85.886	63.074	41.044		
T_{12}	Minimal	137.168	113.363	89.549	0	0
	Selected	137.168	113.363	89.549		
T_{13}	Minimal	175.409	148.481	121.614	0.09%	0.061%
	Selected	175.409	148.624	121.711		

4.6 Evaluation

We conduct experiments to evaluate the feasibility of processing large-scale video streams on hybrid clouds, as well as the effectiveness of the proposed scheduling approach. The experiments are carried out through both simulations and actual runs with our proof-of-concept system on Amazon EC2. The simulations can be repeatedly conducted on large instances, whereas the actual runs involve more accurate running environment but are on relatively smaller instances.

4.6.1 Simulations

The simulations are performed under two different settings: *with* and *without* security constraint. As illustrated in Figure 4.9, 13 different task template graphs are created using the method described in [172]. First, we generate a number of independent operations with random computing cost between (0,2] ECUs. Next, we randomly link the operations with random bandwidth cost between (0,1] MB/s. Since most image processing operations have either one or two inputs and outputs, the average in and out-degree of the task graphs is set to 1.5. Finally, among all the generated DAGs, we randomly choose 13 graphs with differing number of operations ranging from 3 to 15 (with a step of 1). These graphs represent 13 different task templates, each is to be instantiated to 10 video

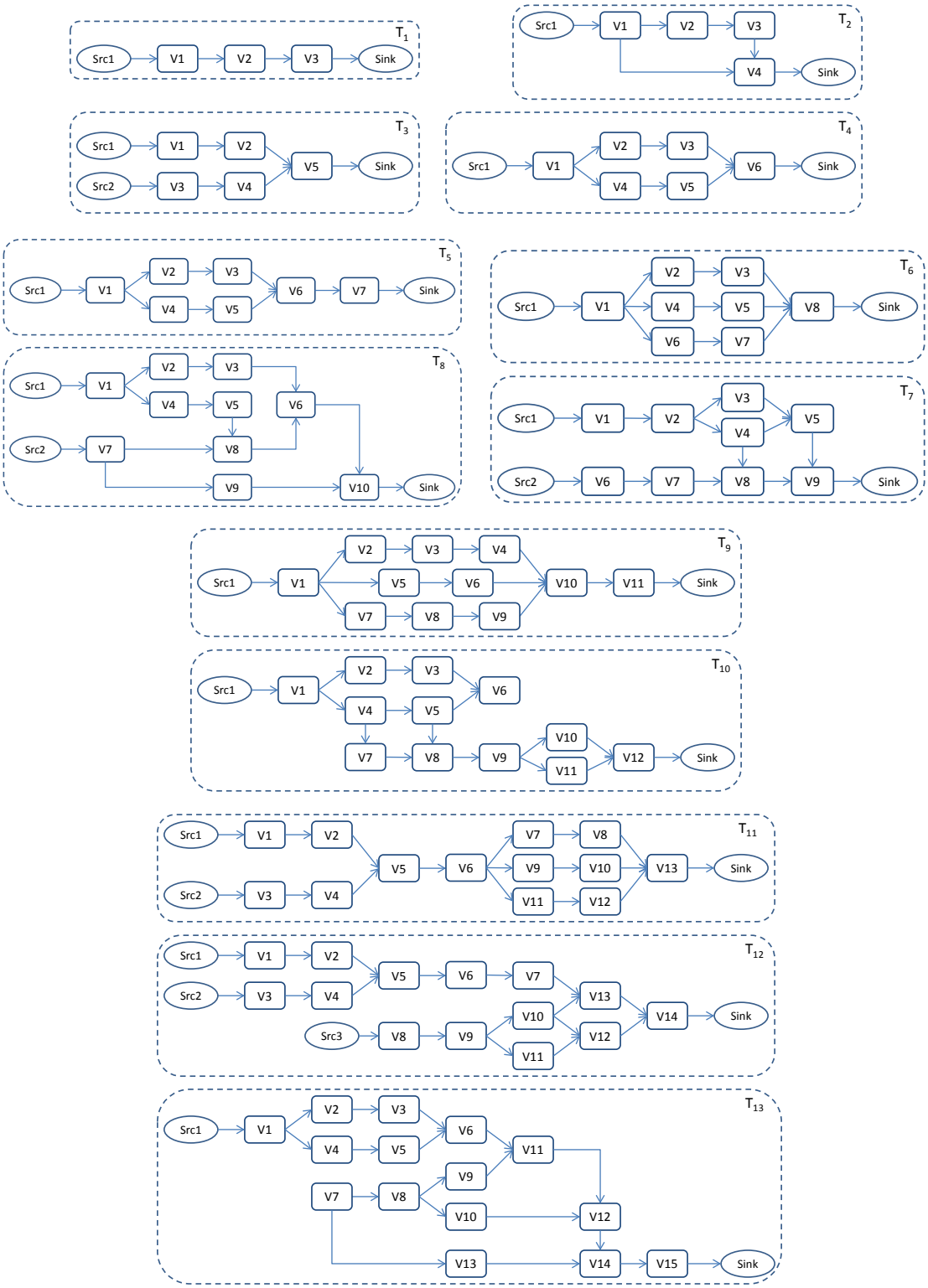


Figure 4.9: The task templates created for simulations.

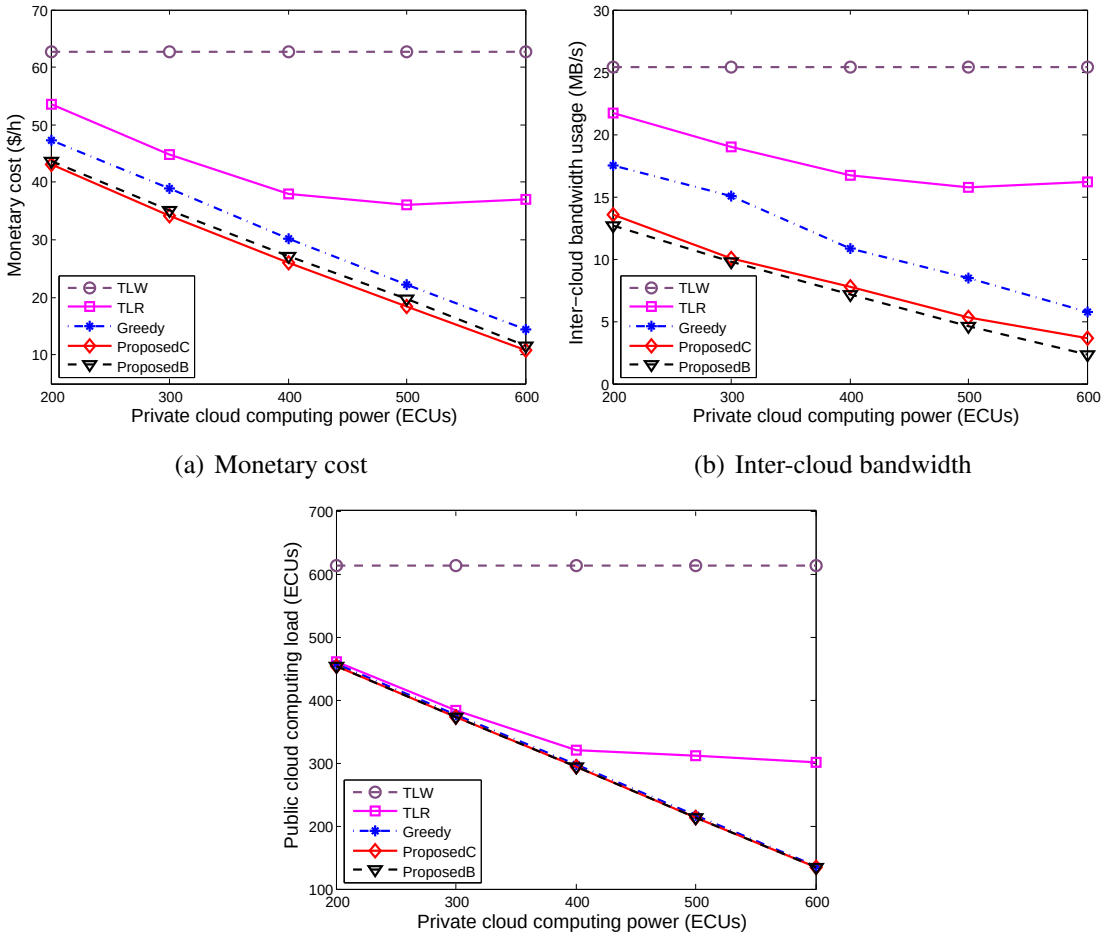
streams, hence there are a total of 130 video streams to process. The source and sink nodes are in the private cloud. We apply a few schedulers (described below) and vary the computing power of the private cloud from 200 to 600 ECUs with a step size of 100. The values of α and β are set to be 0.08 and 0.684 respectively. The bandwidth and delay of the inter-cloud connection are set to be 10 MB/s and 250ms respectively. The end-to-end delay constraint for each template T is set to be $P + 1$s where P is the total processing time along the longest path in T. Hence, the delay incurred by the communication is constrained to be at most 1 second.

We compare among the following 5 scheduling algorithms: 1) *Task-Level Water-filling (TLW)*: assign all operations in a task to private if one of the streams is tagged as sensitive, otherwise assign all operations to the public cloud; 2) *Task-Level Random (TLR)*: same as TLW for tasks with at least one sensitive stream. For tasks tagged with only non-sensitive streams, the whole task is randomly assigned to the public or private cloud; 3) *Greedy*: consider each task one-by-one iteratively. In each round, choose the optimal assignment (i.e., the one minimizing the cost) with respect to the updated resource requirements. 4) *ProposedC*: our proposed approach with objective to minimize the monetary cost; and 5) *ProposedB*: our proposed approach with objective to minimize the inter-cloud bandwidth usage.

Simulation result without security constraint

In this simulation, all of the streams are non-sensitive, i.e., there is no security constraint. Figure 4.10 shows the result under this setting.

The TLW algorithm sends all video streams to the public cloud since all the streams are non-sensitive. Hence, TLW incurs the highest load on the public cloud as well as the highest inter-cloud bandwidth usage, giving the largest monetary cost. TLR can keep some streams in the private cloud, which reduces both the public cloud load and bandwidth usage, hence, giving a smaller cost than TLW. As shown in Figure 4.10(c), both TLR and TLW underutilize the private cloud resources. In contrast, the Greedy algorithm



(a) Monetary cost

(b) Inter-cloud bandwidth

(c) Outsourced computation

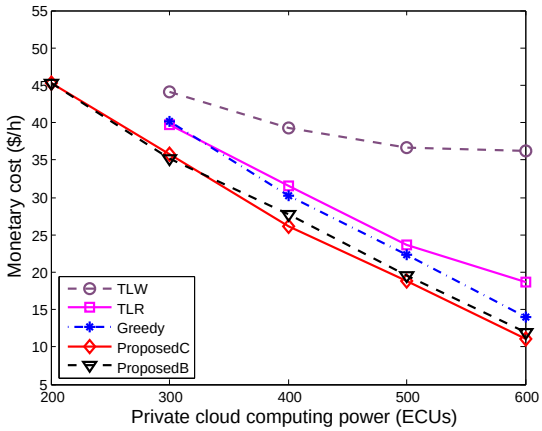
Figure 4.10: Simulation result without security constraint (ProposedC, ProposedB and Greedy are indistinguishable in (c)).

can fully load the private cloud, leading to costs that are smaller than TLW and TLR. Both of our proposed schedulers outperform the others in all the three measures. Note that the differences between ProposedC and ProposedB are indistinguishable as bandwidth cost dominates within the total cost. Hence, they tend to choose the same configurations.

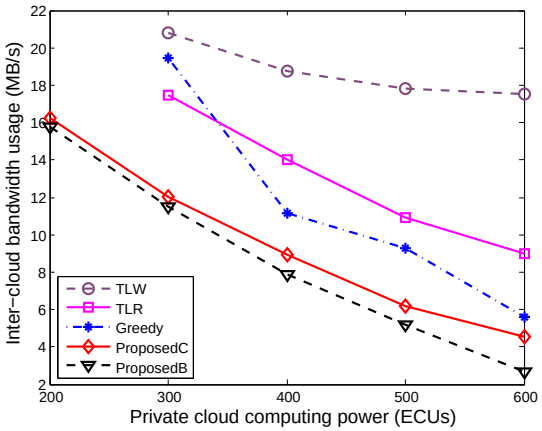
Table 4.3 shows the average time taken by the proposed scheduler. By considering only the minimal configurations, we are able to reduce the total number of unknowns from 65528 to 247, which in turn reduces the computing time from around 143 seconds to be less than 0.1s. This optimization is essential for supporting realtime scheduling. Note that the heuristic is not applied in our evaluation since the number of minimal configurations is already sufficiently small.

Table 4.3: Time taken to solve the integer problem.

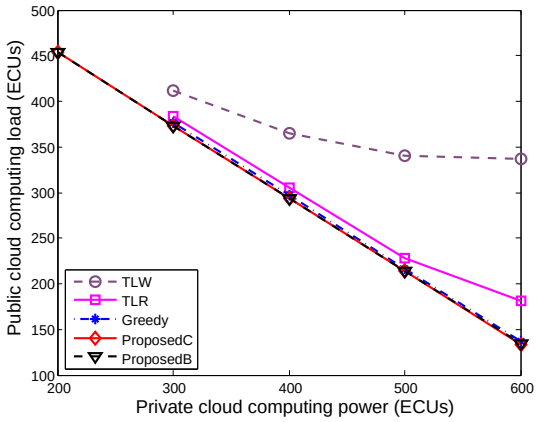
	$\mathcal{F}(\mathcal{T})$	$\mathcal{MF}(\mathcal{T})$
# of Configurations	65528	247
Corresponding Solving Time	143.684s	0.078s



(a) Monetary cost



(b) Inter-cloud bandwidth



(c) Outsourced computation

Figure 4.11: Simulation result with security constraint (ProposedC, ProposedB and Greedy are indistinguishable in (c)).

Simulation result with security constraint

We then randomly tag some of the streams to be sensitive and repeat the above simulation. The result, averaged over 3 random runs, is shown in Figure 4.11. TLW, TLR and Greedy cannot schedule all the tasks when the private cloud has low computing power (i.e., $C = 200$), partly due to the short-sightedness in using the private cloud resources. In contrast, the proposed schedulers exploit global knowledge on all the tasks and hence can schedule all of them. Our schedulers again outperform the other three alternatives.

Monetary cost

Observed that with 200 ECUs in the private cloud, and assuming at the peak overall workload of about 650 ECUs, the number of ECUs employed in the public cloud is about 450 (Figure 4.10(c)), incurring about 13 MB/s inter-cloud bandwidth (Figure 4.10(b)). This amounts to monetary cost of approximately 36 + 8.9 = $44.9/hour during peak. Considering an "offload all" strategy that pushes all video streams and computation to the public cloud, the overall cost would be around $63.1/hour. Hence, we have a reduction in cost of about 29%. With more private resources, the monetary cost can be further reduced, as demonstrated in Figure 4.10 and 4.11.

4.6.2 Proof-of-concept System Evaluation

We also implemented a proof-of-concept system (in C/C++) for this hybrid cloud video surveillance, with basic functionality of video streaming and several operations including transcoding, background/foreground extraction, face detection and recognition etc. For streaming, we simulate a "camera" by writing a program that reads in a video from the local storage, chops the video into 1-second segments and sends them sequentially in real-time (through TCP). The basic operations are implemented using the FFmpeg [26] and OpenCV [27] libraries. We remark that our scheduling mechanism can also be easily incorporated into existing stream processing systems, for example, Apache Storm [16].

Hybrid Cloud Setting

We build a hybrid cloud on Amazon EC2 across Singapore and US West. The private cloud has 6 standard large instances located at Singapore. Each instance provides 2 virtual cores with 4 ECUs, 7.5GB memory and 840GB storage. Hence, the private cloud in our setting has a total computing power of 24 ECUs. The public cloud, located at N. California, has 10 large instances that can be allocated and released on-demand. All instances run Ubuntu 12.04. The available bandwidth between these instances is not

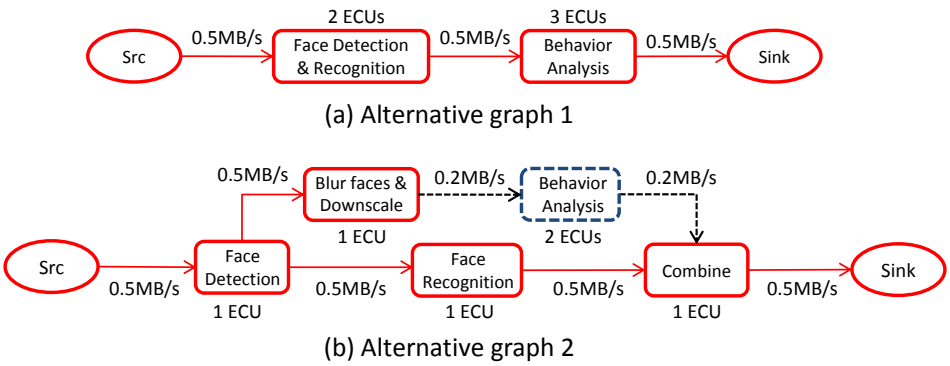
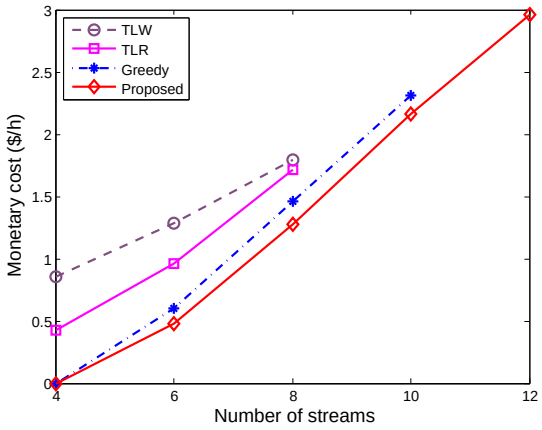


Figure 4.12: Task template for proof-of-concept system evaluation. This task has two alternative ways to complete.

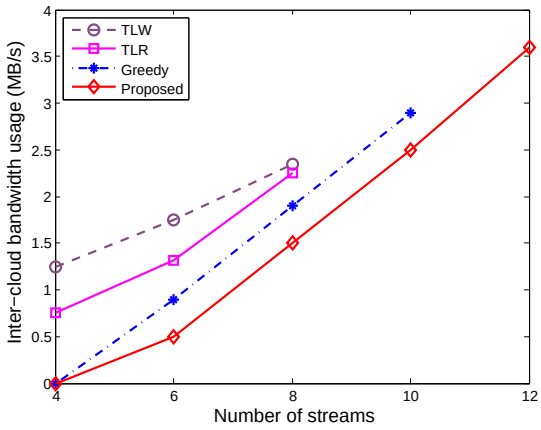
specified by Amazon. An informal test of file transfer using `scp` indicates a speed of around 40 MB/s within the same region (e.g., from Singapore to Singapore), and 5–8 MB/s across different regions (e.g., from Singapore to California). The network delay is less than 1 millisecond for intra-cloud connections and around 250 milliseconds for inter-cloud connections.

Experimental Setting

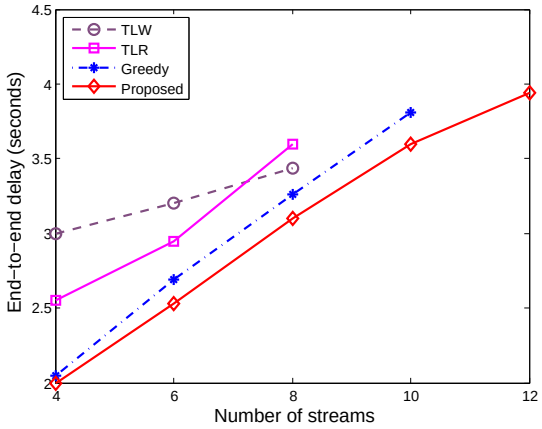
We experiment on one task template, which performs face recognition and behavior analysis that can be carried out in the two different ways illustrated in Figure 4.12. We assign the computation and bandwidth costs to the graph based on a few test runs with our selected video streams for human detection and action recognition [120]. The values are also indicated in Figure 4.12. Note that both the computation and bandwidth costs could vary in practice depending on the data. We gradually increase the number of streams from 4 to 12, with randomly half of them being tagged as sensitive. The maximum allowed end-to-end delay is set to be 5 seconds. We record the actual amount of data transfer across the two clouds, the average end-to-end delay, and calculate the monetary cost spent on the public cloud for a 1-hour run.



(a) Monetary cost



(b) Inter-cloud bandwidth



(c) Average end-to-end delay

Figure 4.13: Experimental result of prototype evaluation. ProposedC and ProposedB produce the same result, hence they are rendered as one line (Proposed).

Result and Analysis

The result is shown in Figure 4.13. Both TLW and TLR fail to schedule all the tasks when the number of streams is greater than 8. Greedy can handle more tasks by pushing some non-sensitive operations to the public cloud, but still fails when the number of streams reaches 12. In contrast, our schedulers can handle all of them. Again, the proposed schedulers give the smallest cost, inter-cloud bandwidth usage and average end-to-end delay. Since bandwidth cost dominates within the total cost, in the experiment, ProposedC and ProposedB always choose the same configurations. Hence, they are rendered as one line (Proposed) in Figure 4.13.

Summary of the Evaluation Both the simulation and proof-of-concept system evaluation demonstrate the effectiveness of our proposed scheduling approach. The scheduler is able to outsource more computation to the public cloud while reducing monetary costs and improving the performance, as compared to other alternative schedulers. The monetary costs can also be reduced as compared to employing a pure public cloud setting.

4.7 Summary

The hybrid cloud naturally addresses the issues on security and seasonal workload in large video surveillance systems. Nevertheless, to fully utilize the potential of the hybrid setting, it is desired to have an effective scheduler, so as to reduce cost and enhance usability. We proposed a scheduler that exploits the two-server setting, and gave empirical result to show that, with an effective scheduler, it is feasible to process large mixed-sensitivity video streams in the cloud. The costs are much lower than a pure public cloud deployment, and overheads are smaller than other alternatives. For future work, it would be interesting to employ existing techniques of fast operation migration [63, 136] to facilitate real-time rescheduling. It is also important to consider different pricing models (varying the ratio between α and β) to show how much improvement we can get in terms of monetary cost under various conditions.

Chapter 5

Conclusions

Data security and privacy have long been recognized as one of the top concerns in cloud computing. While users want to use cloud resources due to the various benefits, they concern about the risk of leaking sensitive data to the cloud. Existing solutions on encrypted domain processing and trusted computing have been found either limited, or impractical, or expensive, which do not qualify as building blocks for a general-purpose, cost-efficient and scalable cloud computing infrastructure. For instance, some cryptographic approaches such as homomorphic encryptions "*would more than undo the economy gained by the outsourcing and show little sign of becoming practical*" [72].

In view of this, this thesis focuses on another light-weight approach of segregating computation under the emerging hybrid cloud setting. If data can be tagged and separated based to their sensitivity, one could simply distribute computation on non-sensitive data to both the public and private clouds, while keeping sensitive data in the private cloud. In this way, privacy-preserving computation can be achieved easily and scalably. The challenge actually shifts to problems of scheduling the non-sensitive computation on the two clouds to reduce performance overhead as well as cost. Unfortunately, there is no platform that can naturally work on hybrid clouds to protect data privacy, not to mention the scheduling for improved performance.

The thesis first considered running computation using the MapReduce paradigm in

a hybrid cloud, and proposed a new programming model for MapReduce that supports tagging of sensitive data. This simple extension provides us with greater flexibility by supporting security policies and complex MapReduce jobs. We then presented multiple scheduling modes to distribute map and reduce tasks on the two clouds appropriately, and showed the efficiency of the proposed algorithms in terms of inter-cloud data traffic, task completion time and monetary cost, through an extension of Hadoop that we have implemented. More interestingly, we observed that keeping sensitive data in the private cloud does not provide full security guarantees, and schedulers can leak information unintentionally in the process of task scheduling and data shuffling. Accordingly, we presented a security model to compare the information leakage by different schedulers and to determine whether a scheduler is secure or not.

The second work shifted the focus to computation using the stream processing paradigm in a hybrid cloud, and dealt with partitioning and scheduling of video processing operations in the domain of video surveillance. In this work, we modeled the scheduling problem as an integer programming problem and solved it using a simple heuristic. Experimental results through both simulations and system runs on Amazon EC2 clouds illustrated the effectiveness and efficiency of the proposed approach. Video surveillance represents a large class of applications in which data streams consist of mixed-sensitivity information. Hence, this approach is able to contribute to many real-world applications.

The hybrid cloud is a fairly practical approach for scaling out computation and data processing needs. Our work investigated the fundamental security and scheduling issues for two widely used programming paradigms on hybrid clouds. Through the above two works, we demonstrated that privacy-preserving computation on hybrid clouds can be made efficient, cost-effective and automatic. In addition, as the main idea of data tagging and segregation is general, it could be easily extended to other cloud computing platforms such as Apache Storm [16] and/or Spark [23].

Future Work

One basic assumption of our work is the ability to tag and separate data based on their sensitivity. This is the common case for many real-world datasets such as business data and healthcare records that involve data with mixed sensitivity. However, there are also cases where the sensitivity cannot be easily determined. For example, a piece of information can be considered as sensitive only within a specific context or only when it is associated with another piece of information. It is an interesting work to investigate how tagging should be done under these situations. On the other hand, the levels of tagging in our work are coarse-grained, either at the file level or at the stream level. It would be valuable to study the effect of different granularities of tagging and their implications to the security as well as to the efficiency.

A second direction for future work is to combine our work with user privilege labelling. Our work only considers machines with different trust levels and we study how to move data across these machines to achieve security and efficiency. Nevertheless, in real-world applications, users may have different privileges to access a same piece of data. For example, different users may be limited to access different portions of the data. We can also label the users in a way similar to tagging, so that users computing on the same data will get different results according to their access privileges. Hence, not only can we prevent information leakage to untrusted machines during execution, but we can also control what information can be presented to each user, preventing information leakage from computation result to unauthorised users. All these can happen automatically inside the system by combining data, machine and user tagging.

Another potential direction is to integrate this hybrid cloud approach with practical encryption techniques. Our work assumes that sensitive data only take up a small portion of the whole input, and can be efficiently handled in the private cloud. It is also possible to encrypt the data using some practical homomorphic encryption schemes and put all the encrypted data on the public cloud. In most cases, computation can be handled on the encrypted data by the public cloud, but when necessary, the private cloud can retrieve the

data, decrypt and compute on them. This kind of combination could be more efficient and scalable than the combination of encrypted domain processing and trusted computing [65, 145]. But it is still not clear how this combination can be made transparent to the users and how much the efficiency can be improved.

Finally, our work also opens a potential research direction for anonymity in cloud computing. Recall that in our tagged-MapReduce framework, revealing of private worker identities leaks certain sensitive information and we introduced a proxy mechanism to prevent such leakage. This is indeed a common issue in many distributed and cloud computing environments. We found that, by only observing the dataflow across different machines in the computing cluster, an adversary is able to derive much sensitive information without tapping into the data. For example, consider a `PageRank` computation using the MapReduce framework. By looking at the input/output relationship between the map and reduce tasks, an adversary could know the in and out-degree of each node and therefore can infer the structure of the whole graph, leading to crucial information leakage. In response, we want to provide an oblivious RAM-like [137] service in cloud computing where each process or machine will receive the data they need to work on, but does not know where the data come from and where the output is sent. This property of anonymity would provide stronger security guarantees for cloud applications and would also complement existing research work on cloud security. This would be an interesting future direction.

Bibliography

[1] 2013 future of cloud computing survey. http://www.northbridge.com/2013-cloud-computing-survey, Accessed in April 2014.

[2] Hadoop wiki. http://wiki.apache.org/hadoop/PoweredBy, Accessed in April 2014.

[3] Hosting and cloud study 2014. http://www.microsoft.com/en-us/news/download/presskits/cloud/docs/hostingstudy2014.pdf, Accessed in April 2014.

[4] North American enterprise survey. http://www.infonetics.com/pr/2013/Cloud-Service-Strategies-Survey-Highlights.asp, Accessed in April 2014.

[5] Rightscale 2014 state of the cloud survey. http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2014-state-cloud-survey, Accessed in April 2014.

[6] The 2012 future of cloud computing survey. http://northbridge.com/2012-cloud-computing-survey, Accessed in August 2013.

[7] 2011 global cloud computing study. http://www.amd.com/Documents/Cloud-Adoption-Approaches-and-Attitudes-Research-Report.pdf, Accessed in August 2014.

[8] Apache Accumulo. http://accumulo.apache.org/, Accessed in December 2012.

[9] English Wikipedia dumps. http://dumps.wikimedia.org/enwiki/, Accessed in December 2012.

[10] Google App Engine. https://appengine.google.com/, Accessed in December 2012.

[11] Logitech alert - video surveillance systems. http://www.logitech.com/en-us/video-security-systems/, Accessed in December 2012.

[12] Microsoft Azure. http://www.windowsazure.com/, Accessed in December 2012.

[13] Swann - advanced security made easy. http://www.swann.com/s/products/view/?product=1161, Accessed in December 2012.

[14] Trendnet SecurView Pro. http://www.trendnet.com/, Accessed in December 2012.

[15] Rackspace 2013 hybrid cloud survey results. http://www.rackspace.com/knowledge_center/article/rackspace-2013-hybrid-cloud-survey-results, Accessed in March 2014.

[16] Storm: Distributed and fault-tolerant realtime computation. http://storm.incubator.apache.org/, Accessed in March 2014.

[17] TPM main specification. http://www.trustedcomputinggroup.org/, Accessed in March 2014.

[18] Apache Hadoop project. http://hadoop.apache.org/, Accessed in May 2012.

[19] Dropbox. https://www.dropbox.com/, Accessed in May 2012.

[20] DARPA Intrusion Detection Data Sets. http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/, Accessed in May 2013.

[21] Google Ngram Viewer. http://storage.googleapis.com/books/ngrams/books/datasetsv2.html, Accessed in May 2013.

[22] Salesforce.com. http://www.salesforce.com/, Accessed in May 2013.

[23] Apache Spark - Lightning-fast cluster computing. https://spark.apache.org/, Accessed in May 2014.

[24] Amazon Web Services (AWS). http://aws.amazon.com, Accessed in November 2012.

[25] Amazon EC2 Pricing. http://aws.amazon.com/ec2/pricing/, Accessed in September 2012.

[26] FFmpeg. https://www.ffmpeg.org/, Accessed in September 2013.

[27] OpenCV (Open Source Computer Vision). http://opencv.org/, Accessed in September 2013.

[28] Pareto efficiency. http://en.wikipedia.org/wiki/Pareto_efficiency#Pareto_frontier, Accessed in September 2014.

[29] Dropbox: Yes, we were hacked. http://gigaom.com/2012/08/01/dropbox-yes-we-were-hacked/, Published in August 2012.

[30] Snowden leak: NSA secretly accessed Yahoo, Google data centers to collect information. http://rt.com/usa/nsa-secretly-access-yahoo-google-982/, Published on 30 October 2013.

[31] Epsilon data breach highlights cloud computing security concerns. http://www.eweek.com/c/a/Security/Epsilon-Data-Breach-

Highlights-Cloud-Computing-Security-Concerns-637161/, Published on 6 April 2011.

[32] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, et al. The design of the Borealis stream processing engine. In *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research*, volume 5, pages 277–289, 2005.

[33] M. S. Ackerman and B. Starr. Social activity indicators: interface components for CSCW systems. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, pages 159–168, 1995.

[34] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *International Conference on Innovative Data Systems Research*, 2005.

[35] Yanif Ahmad and Uğur Çetintemel. Network-aware query processing for stream-based applications. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 456–467, 2004.

[36] Lisa Amini, Navendu Jain, Anshul Sehgal, Jeremy Silber, and Olivier Verscheure. Adaptive control of extreme-scale stream processing systems. In *26th IEEE International Conference on Distributed Computing Systems*, 2006.

[37] Emily Yahr Andrea Peterson and Joby Warrick. Leaks of nude celebrity photos raise concerns about security of the cloud. http://www.washingtonpost.com/politics/leaks-of-nude-celebrity-photos-raise-concerns-about-security-of-the-cloud/2014/09/01/59dcd37e-3219-11e4-8f02-03c644b2d7d0_story.html, Published on September 1, 2014.

[38] M. J. Atallah and K. B. Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 48–59, 2010.

[39] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, pages 39–44, 2003.

[40] Magdalena Balazinska, Hari Balakrishnan, and Mike Stonebraker. Contract-based load management in federated distributed systems. In *1st Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[41] Lucas Ballard, Seny Kamara, and Fabian Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *Information and Communications Security*, pages 414–426. 2005.

[42] M. Blanton and M. Aliasgari. Secure outsourcing of DNA searching via finite automata. In *Data and Applications Security and Privacy*, pages 49–64, 2010.

[43] Erik-Oliver Blass, Roberto Di Pietro, Refik Molva, and Melek Önen. PRISM: Privacy-preserving search in MapReduce. In *Privacy Enhancing Technologies*, pages 180–200, 2012.

[44] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology-Eurocrypt*, pages 506–522, 2004.

[45] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography*, pages 535–554, 2007.

[46] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography*, pages 325–341. 2005.

[47] Dhruba Borthakur. HDFS architecture guide. `http://hadoop.apache.org/common/docs/current/hdfs_design.pdf`, Published in April 2008.

[48] M. Boyle, C. Edwards, and S. Greenberg. The effects of filtered video on awareness and privacy. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 1–10, 2000.

[49] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology–CRYPTO*, pages 868–886. 2012.

[50] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 97–106, 2011.

[51] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D Ernst. HaLoop: Efficient iterative data processing on large clusters. In *Proceedings of the VLDB Endowment*, volume 3, pages 285–296. 2010.

[52] Sven Bugiel, Stefan Nürnberger, Ahmad-Reza Sadeghi, and Thomas Schneider. Twin clouds: An architecture for secure cloud computing. In *Workshop on Cryptography and Security in Clouds*, 2011.

[53] Serdar Cabuk, Chris I Dalton, Konrad Eriksson, Dirk Kuhlmann, Harigovind V Ramasamy, Gianluca Ramunno, Ahmad-Reza Sadeghi, Matthias Schunter, and Christian Stüble. Towards automated security policy enforcement in multi-tenant virtual data centers. *Journal of Computer Security*, 18(1):89–121, 2010.

[54] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *Proceedings of IEEE INFOCOM*, pages 829–837, 2011.

[55] P. Carrillo, H. Kalva, and S. Magliveras. Compression independent object encryption for ensuring privacy in video surveillance. In *IEEE International Conference on Multimedia and Expo*, pages 273–276, 2008.

[56] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. SCOPE: easy and efficient parallel processing of massive data sets. In *Proceedings of the VLDB Endowment*, volume 1, pages 1265–1276. 2008.

[57] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2), 2008.

[58] R. I. Chang, T. C. Wang, C. H. Wang, J. C. Liu, and J. M. Ho. Effective distributed service architecture for ubiquitous video surveillance. *Information Systems Frontiers*, 14(3), 2012.

[59] Y. C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security*, pages 391–421, 2005.

[60] J. Chaudhari, S. S. Cheung, and M. V. Venkatesh. Privacy protection for life-log video. In *IEEE Workshop on Signal Processing Applications for Public Security and Forensics*, pages 1–5, 2007.

[61] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In *IEEE 10th International Conference on Computer and Information Technology (CIT)*, pages 2736–2743, 2010.

[62] Yangyi Chen, Bo Peng, X Wang, and Haixu Tang. Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds. In *Proceeding of the 19th Network and Distributed System Security Symposium*, 2012.

[63] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stanley B Zdonik. Scalable distributed stream processing. In *1st Biennial Conference on Innovative Data Systems Research*, volume 3, pages 257–268, 2003.

[64] K. Chinomi, N. Nitta, Y. Ito, and N. Babaguchi. PriSurv: privacy protected video surveillance system using adaptive visual abstraction. In *Proceedings of the 14th International Conference on Advances in Multimedia Modeling*, pages 144–154, 2008.

[65] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the ACM Workshop on Cloud Computing Security*, pages 85–90, 2009.

[66] K. M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Advances in Cryptology–CRYPTO*, pages 483–501, 2010.

[67] Clavister. Security in the Cloud, white paper. `http://www.clavister.com/Documents/resources/white-papers/clavister-whp-cloud-security-en.pdf`. Accessed in December 2012.

[68] William R Claycomb and Alex Nicoll. Insider threats to cloud computing: Directions for new research challenges. In *IEEE 36th Annual Computer Software and Applications Conference*, pages 387–394, 2012.

[69] Robert T Collins, Alan Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa, Peter Burt,

et al. A system for video surveillance and monitoring. *Robotics Institute, Carnegie Mellon University*, 2000.

[70] C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational cloud: A database-as-a-service for the cloud. In *Conference on Innovative Data Systems Research*, pages 235–241, 2011.

[71] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 79–88, 2006.

[72] David Talbot, MIT Technology Review. How secure is cloud computing? `http://www.technologyreview.com/news/416293/how-secure-is-cloud-computing/`, Published in November 2009.

[73] Marcos Dias De Assunção, Alexandre Di Costanzo, and Rajkumar Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *11th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pages 141–150, 2009.

[74] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 137–150, 2004.

[75] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, and Jörg Schad. Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing). In *Proceedings of the VLDB Endowment*, volume 3, pages 515–529. 2010.

[76] Yannis Drougas, Thomas Repantis, and Vana Kalogeraki. Load balancing techniques for distributed stream processing applications in overlay environments. In

9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2006.

[77] C. Dwork. Differential privacy. In *International Conference on Automata, Languages and Programming*, pages 1–12, 2006.

[78] Joan G Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendert Van Doorn, and Sean W Smith. Building the IBM 4758 secure coprocessor. *Computer*, 34(10):57–66, 2001.

[79] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818, 2010.

[80] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18, 1985.

[81] Jerome Francois, Shaonan Wang, Walter Bronzi, R State, and Thomas Engel. BotCloud: detecting botnets using MapReduce. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2011.

[82] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO*, pages 465–482, 2010.

[83] C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology–EUROCRYPT*, pages 129–148, 2011.

[84] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 169–178, 2009.

[85] Craig Gentry, Shai Halevi, and Nigel P Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology–EUROCRYPT*, pages 465–482. 2012.

[86] N. Ghanem, D. DeMenthon, D. Doermann, and L. Davis. Representation and recognition of events in surveillance video using petri nets. In *Conference on Computer Vision and Pattern Recognition Workshop*, 2004.

[87] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[88] P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security*, pages 31–45, 2004.

[89] R. Goshorn, J. Goshorn, D. Goshorn, and H. Aghajan. Architecture for cluster-based automated surveillance network for detecting and tracking multiple persons. In *First ACM/IEEE International Conference on Distributed Smart Cameras*, pages 219–226, 2007.

[90] John Linwood Griffin, Trent Jaeger, Ronald Perez, Reiner Sailer, Leendert Van Doorn, and Ramón Cáceres. Trusted virtual domains: Toward secure distributed services. In *Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability*, 2005.

[91] Chen He, Ying Lu, and David Swanson. Matchmaking: A new MapReduce scheduling technique. In *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 40–47, 2011.

[92] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *Theory of Cryptography*, pages 264–282, 2005.

[93] Chu Huang, Sencun Zhu, and Dinghao Wu. Towards trusted services: Result verification schemes for MapReduce. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 41–48, 2012.

[94] Yuanqiang Huang, Zhongzhi Luan, Rong He, and Depei Qian. Operator placement with QoS constraints for distributed stream processing. In *7th International Conference on Network and Service Management*, pages 1–7, 2011.

[95] S. E. Hudson and I. Smith. Techniques for addressing fundamental privacy and disruption tradeoffs in awareness support systems. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, pages 248–257, 1996.

[96] Eaman Jahani, Michael J Cafarella, and Christopher Ré. Automatic optimization for MapReduce programs. In *Proceedings of the VLDB Endowment*, volume 4, pages 385–396. 2011.

[97] Shan Jiang, Sean Smith, and Kazuhiro Minami. Securing web servers against insider attack. In *Proceedings of 17th Annual Computer Security Applications Conference*, pages 265–276, 2001.

[98] P. Jithendra K, C. Sen-ching S, H. Michael W, et al. Video data hiding for managing privacy information in surveillance systems. *EURASIP Journal on Information Security*, 2009.

[99] S. Kamara and K. Lauter. Cryptographic cloud storage. In *Financial Cryptography and Data Security*, pages 136–149, 2010.

[100] Seny Kamara and Mariana Raykova. Parallel homomorphic encryption. In *Financial Cryptography and Data Security*, pages 213–225. 2013.

[101] A. Kandhalu, A. Rowe, R. Rajkumar, C. Huang, and C. C. Yeh. Real-time video surveillance over IEEE 802.11 mesh networks. In *15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 205–214, 2009.

[102] Miltiadis Kandias, Nikos Virvilis, and Dimitris Gritzalis. The insider threat in cloud computing. In *Critical Information Infrastructure Security*, pages 93–103. 2013.

[103] A. Karimaa. Video surveillance in the cloud: Dependability analysis. In *The Fourth International Conference on Dependability*, pages 92–95, 2011.

[104] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 938–948, 2010.

[105] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology–EUROCRYPT*, pages 146–162, 2008.

[106] I. Kitahara. Interactive video surveillance by using environmental and mobile cameras. In *World Automation Congress*, pages 1–6, 2008.

[107] Naomi Klein. China's all-seeing eye: A nation under surveillance. Rolling Stone, `http://www.rollingstone.com/`, Published on 29 May 2008.

[108] S. Y. Ko, K. Jeon, and R. Morales. The HybrEx model for confidentiality and privacy in cloud computing. In *Proceedings of the 2011 Conference on Hot Topics in Cloud Computing*, 2011.

[109] T. Koshimizu, T. Toriyama, and N. Babaguchi. Factors on the sense of privacy in video surveillance. In *Proceedings of the 3rd ACM Workshop on Continuous Archival and Retrival of Personal Experences*, pages 35–44, 2006.

[110] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard OS abstractions. *ACM SIGOPS Operating Systems Review*, 41(6):321–334, 2007.

[111] Geetika T Lakshmanan, Ying Li, and Rob Strom. Placement of replicated tasks for distributed stream processing systems. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems*, pages 128–139, 2010.

[112] Geetika T Lakshmanan, Yuri G Rabinovich, and Opher Etzion. A stratified approach for supporting high throughput event processing applications. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, 2009.

[113] Geetika T Lakshmanan and Robert E Strom. Biologically-inspired distributed middleware management for stream processing systems. In *Middleware*, pages 223–242, 2008.

[114] A. Lee, K. Schlueter, and A. Girgensohn. Sensing activity in video images. In *CHI'97 Extended Abstracts on Human Factors in Computing Systems: Looking to the Future*, pages 319–320, 1997.

[115] Jingwei Li, Chunfu Jia, Jin Li, and Xiaofeng Chen. Outsourcing encryption of attribute-based encryption with MapReduce. In *Information and Communications Security*, pages 191–201. 2012.

[116] Q. Li, T. Zhang, and Y. Yu. Using cloud computing to process intensive floating car data for urban traffic surveillance. *International Journal of Geographical Information Science*, 25(8):1303–1322, 2011.

[117] Björn Lohrmann, Daniel Warneke, and Odej Kao. Massively-parallel stream processing under QoS constraints with Nephele. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, pages 271–282, 2012.

[118] D. Lopresti and A. L. Spitz. Quantifying information leakage in document redaction. In *ACM Workshop on Hardcopy Document Processing*, pages 63–69, 2004.

[119] Peng Lu, Young Choon Lee, Chen Wang, Bing Bing Zhou, Junliang Chen, and Albert Y Zomaya. Workload characteristic oriented scheduler for MapReduce. In *Proceedings of the IEEE 18th International Conference on Parallel and Distributed Systems*, pages 156–163, 2012.

[120] Marcin Marszałek, Ivan Laptev, and Cordelia Schmid. Actions in context. In *IEEE Conference on Computer Vision & Pattern Recognition*, 2009.

[121] Michael Mattess, Christian Vecchiola, and Rajkumar Buyya. Managing peak loads by leasing cloud infrastructure services from a spot market. In *12th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pages 180–188, 2010.

[122] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. PIRMAP: Efficient private information retrieval for MapReduce. In *Financial Cryptography and Data Security*, pages 371–385. 2013.

[123] Marianne Kolbasuk McGee. HIPAA Breaches in the Cloud. Healthcare Info Security, `http://www.healthcareinfosecurity.com/hipaa-breaches-in-cloud-a-5959`, Published on August 1, 2013.

[124] P. Mell and T. Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 2009.

[125] H. Mittelmann. Mixed Integer Linear Programming Benchmark (MIPLIB2010). `http://plato.asu.edu/ftp/milpc.html`, Accessed in October 2013.

[126] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. GUPT: privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 349–360, 2012.

[127] David Neal and Syed Rahman. Video surveillance in the cloud? *The International Journal of Cryptography and Information Security (IJCIS)*, 2(3), 2012.

[128] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *IEEE International Conference on Data Mining Workshops*, pages 170–177, 2010.

[129] Niall Kennedy's Weblog. Google processes over 20 petabytes of data per day. `http://www.niallkennedy.com/blog/2008/01/google-mapreduce-stats.html`, Published on 8 January 2008.

[130] Kerim Yasin Oktay, Vaibhav Khadilkar, Bijit Hore, Murat Kantarcioglu, Sharad Mehrotra, and Bhavani Thuraisingham. Risk-aware workload distribution in hybrid clouds. In *IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 229–236, 2012.

[131] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology – EUROCRYPT*, pages 223–238, 1999.

[132] D. F. Parkhill. *The challenge of the computer utility*. Addison-Wesley Educational Publishers Inc. US, 1966.

[133] R. Pereira, M. Azambuja, K. Breitman, and M. Endler. An architecture for distributed high performance video processing in the cloud. In *IEEE 3rd International Conference on Cloud Computing*, pages 482–489, 2010.

[134] Ronald Perez, Reiner Sailer, Leendert van Doorn, et al. vTPM: virtualizing the trusted platform module. In *Proceedings of the 15th Conference on USENIX Security Symposium*, pages 305–320, 2006.

[135] Miodrag Petkovic, Miroslav Popovic, Ilija Basicevic, and Djordje Saric. A host based method for data leak protection by tracking sensitive data flow. In *IEEE*

19th International Conference and Workshops on Engineering of Computer Based Systems, pages 267–274, 2012.

[136] Peter Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo Seltzer. Network-aware operator placement for stream-processing systems. In *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[137] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *Advances in Cryptology–CRYPTO 2010*, pages 502–519. 2010.

[138] Alexander Rasmussen, Michael Conley, George Porter, Rishi Kapoor, Amin Vahdat, et al. Themis: an I/O-efficient MapReduce. In *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012.

[139] T.D. Räty. Survey on contemporary remote surveillance systems for public safety. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(5):493–515, 2010.

[140] Kai Ren, YongChul Kwon, Magdalena Balazinska, and Bill Howe. Hadoop's adolescence: an analysis of Hadoop usage in scientific workloads. In *Proceedings of the VLDB Endowment*, volume 6, pages 853–864, 2013.

[141] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 199–212, 2009.

[142] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[143] Rob Marson, JDSU Enterprise Solutions. More organizations move to the cloud in 2013. http://blogs.jdsu.com/perspectives/archive/2012/12/12/more-organizations-move-to-the-cloud-in-2013.aspx, Published in December 2012.

[144] Indrajit Roy, Srinath TV Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for MapReduce. In *USENIX Symposium on Networked Systems Design and Implementation*, volume 10, pages 297–312, 2010.

[145] Ahmad-Reza Sadeghi, Thomas Schneider, and Marcel Winandy. Token-based cloud computing. In *Trust and Trustworthy Computing*, pages 417–429. 2010.

[146] Ahmad-Reza Sadeghi, Christian Stüble, and Marcel Winandy. Property-based TPM virtualization. In *Information Security*, pages 1–16. 2008.

[147] Reiner Sailer and Matthias Kabatnik. History based distributed filtering: A tagging approach to network-level access control. In *16th Annual Conference Computer Security Applications*, pages 373–382, 2000.

[148] M. Saini, P. K. Atrey, S. Mehrotra, S. Emmanuel, and M. Kankanhalli. Privacy modeling for video data publication. In *IEEE International Conference on Multimedia and Expo*, pages 60–65, 2010.

[149] M. Saini, W. Xiangyu, P. Atrey, and M. Kankanhalli. Dynamic workload assignment in video surveillance systems. In *IEEE International Conference on Multimedia and Expo*, pages 1–6, 2011.

[150] M. K. SAINI. *Privacy aware surveillance system design*. PhD thesis, National University of Singapore, 2011.

[151] Bruce Schneier. Schneier on security: Homomorphic encryption breakthrough. http://www.schneier.com/blog/archives/2009/07/homomorphic_enc.html, Published on 9 July 2009.

[152] A. Senior, S. Pankanti, A. Hampapur, L. Brown, Y. L. Tian, A. Ekin, J. Connell, C. F. Shu, and M. Lu. Enabling video privacy through computer vision. In *IEEE Security and Privacy*, volume 3, pages 50–57, 2005.

[153] Sangeetha Seshadri, Vibhore Kumar, and Brian F Cooper. Optimizing multiple queries in distributed data stream systems. In *Proceedings of 22nd International Conference on Data Engineering Workshops*, 2006.

[154] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Theory of Cryptography*, pages 457–473, 2009.

[155] N. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography–PKC*, pages 420–443, 2010.

[156] Sean W Smith and Steve Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31(8):831–860, 1999.

[157] Sohini Bagchi, CXOtoday.com. More companies moving their data to the cloud. `http://www.cxotoday.com/story/more-companies-moving-their-data-to-the-cloud/`, Published in January 2013.

[158] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[159] Fran Spielman. Surveillance cams help fight crime, city says. Chicago Sun Times, `http://politics.suntimes.com/`, Published on 19 February 2009.

[160] S. Tansuriyavong and S. Hanaki. Privacy protection by concealing persons in circumstantial video image. In *Proceedings of the 2001 Workshop on Perceptive User Interfaces*, pages 1–4, 2001.

[161] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a ware-

housing solution over a Map-Reduce framework. In *Proceedings of the VLDB Endowment*, volume 2, pages 1626–1629. 2009.

[162] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 228–235, 2010.

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

[164] M. V. Venkatesh, S. C. Cheung, J. K. Paruchuri, J. Zhao, and T. Nguyen. Protecting and managing privacy information in video surveillance systems. *Protecting Privacy in Video Surveillance*, 2009.

[165] Dung Vu, Vana Kalogeraki, and Yannis Drougas. Efficient stream processing in the cloud. In *Quality, Reliability, Security and Robustness in Heterogeneous Networks*, pages 265–281. 2012.

[166] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *IEEE 30th International Conference on Distributed Computing Systems (ICDCS)*, pages 253–262, 2010.

[167] J. Wang, W. Q. Yan, M. S. Kankanhalli, R. Jain, and M. J. T. Reinders. Adaptive monitoring for video surveillance. In *Proceedings of the 2003 Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing*, volume 2, pages 1139–1143, 2003.

[168] Rui Wang, XiaoFeng Wang, Zhou Li, Haixu Tang, Michael K Reiter, and Zheng Dong. Privacy-preserving genomic computation through program specialization. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 338–347, 2009.

[169] J. Wickramasuriya, M. Datt, S. Mehrotra, and N. Venkatasubramanian. Privacy protecting data collection in media spaces. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, pages 48–55, 2004.

[170] Laurence A Wolsey. Integer programming. *IIE Transactions*, 32(273-285):2–58, 2000.

[171] Zhifeng Xiao and Yang Xiao. Accountable MapReduce in cloud computing. In *Proceedings of IEEE Conference on Computer Communications Workshops*, pages 1082–1087, 2011.

[172] Mau-Tsuen Yang, Rangachar Kasturi, and Anand Sivasubramaniam. A pipeline-based approach for scheduling video processing algorithms on NOW. *IEEE Transactions on Parallel and Distributed Systems*, 14(2):119–130, 2003.

[173] Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 203–216, 2011.

[174] Hui Zhang, Guofei Jiang, Kenji Yoshihira, Haifeng Chen, and Akhilesh Saxena. Intelligent workload factoring for a hybrid cloud computing model. In *World Conference on Services*, pages 701–708, 2009.

[175] K. Zhang, X. Zhou, Y. Chen, X. F. Wang, and Y. Ruan. Sedic: privacy-aware data intensive computing on hybrid clouds. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 515–526, 2011.

[176] W. Zhang, S. C. Cheung, and M. Chen. Hiding privacy information in video surveillance system. In *Proceedings of the 12th IEEE International Conference on Image Processing*, pages 868–871, 2005.

[177] Q. A. Zhao and J. T. Stasko. Evaluating image filtering based techniques in media space applications. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 11–18, 1998.

[178] Jun Zheng, Geovany A Ramírez, and Olac Fuentes. Face detection in low-resolution color images. In *Image Analysis and Recognition*, pages 454–463. 2010.