UNIVERSITY OF CALIFORNIA,
IRVINE

Using Hybrid Clouds for Secure and Efficient Data Processing

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Kerim Yasin Oktay

Dissertation Committee:
Professor Sharad Mehrotra, Chair
Professor Michael Carey
Professor Chen Li

2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Prof. Sharad Mehrotra for his understanding, passion, wisdom and encouragement and for pushing me farther than i thought i could go at UC Irvine. He has taught me how to identify new research problems and how to address them. Without his persistent guidance, this thesis would have not been possible.

I would also like to thank Prof. Murat Kantarcioglu and his student Dr. Vaibhav Khadilkar, for their assistance and suggestions throughout my research projects.

I would like to extend my thanks to my committee members, Prof. Michael Carey and Prof. Chen Li for their guidance and support over the years and for letting me a member of the amazing ISG group.

I take this opportunity to record my sincere thanks to the Computer Science Department of University of California Irvine for letting me pursue my Ph.D. dream in Computer Science. I also thank to the staff of this department and ICS graduate division for helping me to do everything according to the procedures.

I would like to thank my parents, Tulay and Ibadi Oktay, and my siblings, Afsin and Esra Oktay, who have patiently supported me during the last six years. I would like to extend my thanks to all my family members, especially to my grandmothers, Fikriye Oktay and Muserref Senocak for praying for me to come up to this point. Without their unconditional support, it would not be possible for me to succeed any of my academic achievements.

Last of all, i would like to thank my friends for keeping me their company in our countless meetings. Without their companionship, my life in here apart from my family would have been very tough.

Above all, Elhamdulillah, all praises are belonging to Allah S.W.T who has given me the health and necessary skills to finish this dissertation. Without his numerous blessings, this would not have been possible.

CURRICULUM VITAE

Kerim Yasin Oktay

Education

University of California– Irvine, Irvine, CA, USA 2015
Doctor of Philosophy, Information and Computer Science.

University of California – Irvine, Irvine, CA, USA 2011
Master of Science, Computer Science.

Bilkent University, Ankara, Turkey 2009
Bachelor of Science, Computer Science.

Publications

Kerim Yasin Oktay, Vaibhav Khadilkar, Murat Kantarcioglu, Sharad Mehrotra. SEMROD: Secure and Efficient MapReduce over Hybrid Clouds. In: *SIGMOD*, 2015.

Kerim Yasin Oktay, Mahadevan Gomathisankaran, Murat Kantarcioglu, Sharad Mehrotra, Anoop Singhal. Towards Data Confidentiality and a Vulnerability Analysis Framework for Cloud Computing. In *Secure Cloud Computing*, 2014.

Kerim Yasin Oktay, Vaibhav Khadilkar, Murat Kantarcioglu, Sharad Mehrotra. Risk Aware Approach to Data Confidentiality in Cloud Computing. In: *ICISS*, 2014.

Ata Turk, Kerim Yasin Oktay, and Cevdet Aykanat. Query-Log Aware Replicated Declustering. In: *TPDS*, 2013.

Vaibhav Khadilkar, Kerim Yasin Oktay, Murat Kantarcioglu, Sharad Mehrotra. Secure Data Processing over Hybrid Clouds. In: *IEEE DEB*, 2012.

Kerim Oktay, Vaibhav Khadilkar, Bijit Hore, Murat Kantarcioglu, Sharad Mehrotra, Bhavani Thuraisingham. Risk-Aware Workload Distribution in Hybrid Clouds. In: *IEEE Cloud*, 2012.

Kerim Yasin Oktay, Ata Turk, Cevdet Aykanat. Selective Replicated Declustering for Arbi-

trary Queries In: *Euro-Par*, 2009.

Selected Awards

2009-2015: PhD Fellowship, University of California, Irvine.
2015: Travel Award Fellowship, SIGMOD.
2004: Fellowship from TUBITAK (The Scientific and Technological Research Council of Turkey) 2004: Bachelor Fellowship, Bilkent University.
2004: Silver Medal at 45th International Mathemarical Olympiads
2004: Bronze Medal at 21th International Balkan Mathemarical Olympiads

ABSTRACT OF THE DISSERTATION

Using Hybrid Clouds for Secure and Efficient Data Processing

By

Kerim Yasin Oktay

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 2015

Professor Sharad Mehrotra, Chair

Fueled with the advances in virtualization and high-speed networking, cloud computing has emerged as a dominant computing paradigm. Loss of control over the data due to migration to the cloud poses numerous concerns about data privacy and confidentiality, *e.g.*,the sensitive data could be misused by the attackers, other tenants or service provider itself. A possible approach to overcome such concerns is to encrypt the data prior to outsourcing it to the cloud and to perform data processing over encrypted data in the cloud. Although the database and cryptography communities have made significant progress on developing systems that allow limited computation over encrypted data, no generic and efficient solution for practical use has emerged yet.

In this thesis, we explore a radically different approach by using hybrid clouds as a vehicle to achieve secure and efficient data processing in the cloud. We explore a design of secure and efficient systems that steers the data and computation through public and private machines in such a way that no (or user-specified amount of) sensitive data is leaked to public machines. For this purpose, we first propose a fully secure and efficient MapReduce framework over hybrid clouds, named as *SEMROD*. Second, we design a fully secure and efficient execution strategy, called *split-strategy*, to partition relational data and SQL style queries across a hybrid cloud. Third, we propose a principled conceptual framework, called as Hybridizer,

that adjusts the data and workload that will be outsourced to the public cloud based on maximizing the workload performance while meeting user's risk and cost constraints. In Hybridizer, our aim is to provide more performance gain by allowing to expose a user-bounded amount of sensitive data to the public cloud (risk).

Overall, our experiments demonstrate outstanding results in terms of performance – that is, organizations can have significant performance gains compared to other secure solutions by using our secure and risk-aware data processing frameworks over their hybrid clouds.

Chapter 1

Introduction

Organizations today collect and store large volumes of data that they would like to analyze for a multitude of purposes. For instance, an e-commerce company may combine click stream data with customer information to better understand customer behavior for purposes ranging from serving appropriate advertisements to deciding on discount offerings or product pricing. Often the in-house computational capabilities of organizations cannot easily support complex data analyses. While such limitations were a serious impediment in the past, emerging public cloud computing platforms (*e.g.*, Amazon's EC2) offer a viable alternative. Public cloud provide a cost-effective storage and computational infrastructure.

Despite numerous benefits, many organizations, specially for whom security and data confidentiality is paramount, have significant concerns in embracing the cloud model [3], since they deal with potentially sensitive data (*e.g.*, business secrets, sensitive client information such as credit card and social security numbers, medical records, *etc.*).

Security concerns of clients have led cloud providers to support mechanisms that empower data owners to encrypt data prior to storage, effectively preventing many of the possible data vulnerabilities. Encryption, however, raises a new challenge of computing on the encrypted

domain if one wishes to use the cloud as a computing infrastructure (not just for storage).

The challenge of data processing over the encrypted data has been addressed extensively in the cryptography and database literature over the past decade. Numerous encryption techniques have been developed to support basic computations over encrypted data, including searchable encryption [4–10], order-preserving encryption [11–13], deterministic encryption [14], fully-homomorphic encryption [15], partially-homomorphic encryption [16–19], and bucketization technique [20]. For instance, an order preserving encryption (OPE) technique encrypts a set of inputs in such a way that the order of the inputs are preserved within ciphertexts. Another notable work, fully-homomorphic encryption (FHE) based on lattice-based cryptography can support both multiplication and addition within the encrypted domain.

Fueled with such advances, a new class of database systems, that enable SQL like queries to run over the encrypted data have emerged. For instance, early work on database as a service mode (DAS) [20] explored techniques to execute as many operators as possible in the given query tree over the encrypted data and when processing could not continue on the encrypted domain, the data is transferred to the secure client, which could then decrypt the data and continue with query execution. Another system, Cipherbase [21] explores the role of secure coprocessors in the cloud environment to quickly decrypt data when its needed and continue processing. CryptDB [22] introduced onion encryption methods to guarantee that data is selectively decrypted at the outsourced server (public cloud), just enough to support the functionality desired. MONOMI [23] presented a split strategy between trusted client and untrusted server, which can execute arbitrarily complex queries over encrypted data. Influenced by the success of research prototypes, many similar systems are being designed by commercial vendors (e.g., SAP, Google) and by government research labs (e.g., NIS, Lincoln Labs). Examples would be Google BigQuery [24] and No-SQL D4M Accumulo engine of Lincoln Labs [25].

Such systems do not strive to ensure complete data confidentiality – instead they are designed

to preserve as much confidentiality as possible while still ensuring efficient data processing. Systems typically explore tradeoffs between level of confidentiality achieved and performance overheads (as in [20]) or the level of functionality supported (as in CryptDB and other systems inspired by it). Unfortunately, recent studies [26, 27] have highlighted that when order preserving and deterministic encryption techniques are used together, as in CryptDB, on a dataset in which the entropy of the values is not high-enough, an attacker might be able to construct the entire plaintext by doing a frequency analysis over the encrypted data. So, it is difficult to precisely characterize the degree of security offered by such systems. Systems based on encryption that completely prevent leakage of sensitive information remains an open challenge – while such systems could possibly be designed based on the seminal work on FHE technique [15], and on the work on oblivious RAM [28], such techniques are too inefficient[1] to date to form the basis of building technology that offers secure data processing. The quest for secure and efficient data processing in cloud environments, thus, remains, an illusive goal.

Additionally, the goal of encryption based systems[2] is to provide *functional completeness* [30] wherein the entire computation is outsourced to untrusted servers (public cloud). The only task performed at the local machines is to decrypt the results. An organization, however, might have a considerable size storage and computational power in their own site. But, such systems will never be able to utilize this in-house computational resources for the data processing.

Finally, to achieve full security, encrypted database systems end up encrypting all the data items, regardless of their sensitivity. Whether they are sensitive or not for an organization, all the records must be encrypted in such systems in order to prevent sensitive information leakage. For instance, CryptDB pays the huge overhead of encrypted query processing,

[1]In FHE, each integer is represented using 2^{14} bits. Doing addition/multiplication over the encrypted data takes cosmic time [29].

[2]Except for DAS which does allow computation to be split between public and private machines

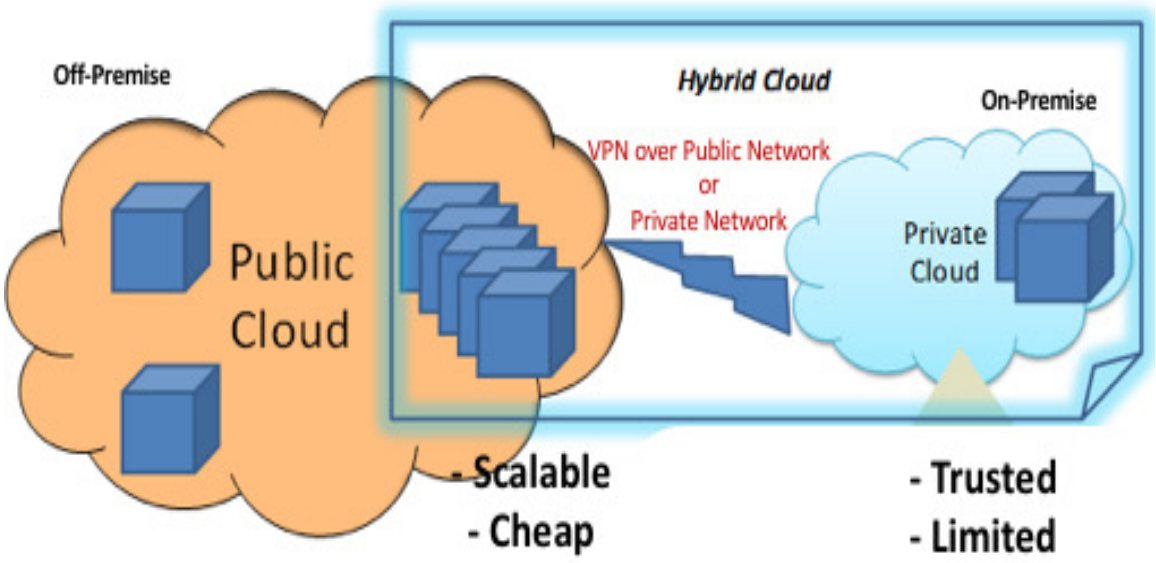

Figure 1.1: Hybrid Cloud Architecture

even though the entire query is executed on a few records that are not sensitive for the user. Thereby, for an organization whose data is mostly non-sensitive, encrypted databases systems will not be an efficient option.

In this thesis, we build upon a radically different (but complementary) approach to secure data processing in the cloud. Instead of exploring encryption as a basis to build secure data processing systems , we explore a design of secure systems that prevents sensitive data from leaking to the public domain. In the envisioned approach, the data and computation is partitioned across machines trusted by the data owner and the untrusted public cloud in such a way that sensitive information never leaves the trusted machines. The approach is intended for situations where

- owners/organizations have an existing storage / computational resources of their own, though the resources may be too limited to process/analyze data in its entirety, and
- sensitive data is only a small part of the overall data over which data owners may need to compute.

A perfect embodiment of a system for which our approach is designed is a mixed computation environment such as *hybrid cloud*. A hybrid cloud, as shown in Figure 1.1, enables composition of two distinct cloud infrastructures, private and public, that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability [31]. In a typical hybrid cloud, the private cloud machines are connected to the public cloud machines over a public network or a special private network offered by the public cloud provider. The hybrid cloud paradigm allows end-users to seamlessly integrate their trusted on-premise computing resources with the cheap and scalable public cloud services and construct potent, secure and economical data processing solutions. the For instance, hybrid clouds can empower organizations to partition data and computation amongst public and private machines in such a way that they can leverage the power of the public cloud while ensuring that the sensitive data or computation never leaves private machines. Indeed, the potential for using hybrid clouds for secure data processing has been recently recognized in the Beckman Workshop on Data Management [32] as well as, in popular media [33] that lists 31% of IT managers (projected to reach 74% in the near future) preferring hybrid clouds rather than public or private deployments both for security and cost reasons. Indeed, Big cloud enterprises such as Microsoft and VMWare are increasingly investing in hybrid cloud solutions [33].

Distributing data processing in a secure and efficient way across the hybrid cloud is not an easy task. The first challenge is to ensure that the execution is secure. This constraints data processing to be such that no sensitive data migrate to the public cloud. However, as shown in [1], leakage of sensitive data may occur indirectly, even when no sensitive data is exposed to the public cloud. The execution must be appropriately distributed to ensure that no direct or indirect sensitive information leakage possible. To overcome such security concerns, one can trivially bring the non-sensitive data to the private cloud (or initially store it at the private cloud) and handle the entire data processing at the private side. Of course, such an approach would perform poorly in terms of efficiency, since public cloud resources

are not exploited for data processing. The challenge is to simultaneously satisfy the security while providing efficient execution.

Our goal in this thesis is to use hybrid clouds as a vehicle to achieve secure and efficient data processing in cloud computing. This thesis will explore how partitioning computation based on data sensitivity by itself; in conjunction with risk-based processing techniques (that explore trading bounded amount of sensitive data exposure for efficiency) can be used to support secure yet efficient data processing in hybrid clouds.

First, we present a fully secure and efficient MapReduce (MR) framework over the hybrid clouds. For large scale data analyses, MR systems have been widely used because of their high scalability, fault tolerance and easy-to-use features. In the MR paradigm, users specify a map task that processes an input record to generate a set of intermediate key-value (KV) pairs, and a reduce task that merges all intermediate values with the same key. The runtime system executes the map and reduce tasks in parallel over a cluster of machines. Current MR systems, when deployed over hybrid clouds, do not take into account the sensitivity of information when storing/processing data. Such MR systems could disclose potentially sensitive information to other tenants residing on the public cloud or to privileged insiders having access to the public cloud infrastructure. To overcome these challenges, we propose a new framework, named as SEMROD, which provides a secure and efficient MR processing over hybrid clouds.

Second, we explore how to partition SQL-like structured queries across hybrid clouds in a fully secure and efficient way. Although supporting generic MR jobs is a significant step, many MR jobs that are executed in practice are generated based on the queries written in structured query languages such as HiveQL [34] and Pig Latin [35]. One approach to implementing secure SQL style queries is to transform them into corresponding MR jobs and use our SEMROD framework. SQL offers much higher level of semantics that can be exploited to significantly improve performance without sacrificing semantic security. To

achieve this, we develop a strategy to partition SQL queries, called *split-strategy*, which provides semantically secure and efficient query processing compared to running the same set of queries on the private cloud.

Third, we explore a cost and risk aware mechanism to partition workloads of SQL-style queries across a hybrid cloud. Different data and workload partitionings display different trade-offs in terms of the running time of the workload (performance), the monetary cost of using resources from a public cloud (cost) and the risk of sensitive data disclosure (risk) metrics. For instance, storing all the data and executing the given workload only on private cloud leads to zero risk and zero cost, but will most likely incur low performance due to not exploiting public cloud resources. Alternatively, shifting the entire data and computation to the public cloud may have a high performance benefit, but at the same time incurs the maximum risk and monetary costs. We propose a conceptual framework, called as Hybridizer, that adjusts the data and workload that will be outsourced to the public cloud based on maximizing the workload performance while meeting user's risk and cost constraints. Note that our previous approaches focuses on providing efficiency without violating semantic security (i.e no sensitive data exposed to the public cloud), whereas Hybridizer aims to provide a performance gain by exposing a user specified amount of sensitive data to the public cloud.

The rest of the thesis is organized as follows: Chapter 2 reviews the related work in the area of secure data processing. Chapter 3 provides an overview of our security model, namely the sensitivity and adversarial models deployed in this thesis. In Chapter 4, we present our secure and efficient MR framework over hybrid clouds. Chapter 5 provides the details about our algebraic approach to constructing a secure and efficient execution strategy for SQL-like queries over a hybrid cloud. In Chapter 6, we introduce our general-purpose *Hybridizer* framework to automatically partition a workload of tasks (or queries), and in turn, its data, over a hybrid cloud. Finally, we present our conclusions and future work in Chapter 7.

This thesis is partially supported by the following grants: Air Force Office of Scientific Re-

search FA9550-12-1-0082, National Science Foundation (NSF) Grants Career-CNS-0845803, CCF-1212943, CNS-1118127, CNS-1059436, CNS-1111529, CNS-1228198 and Army Research Office Grant W911NF-12-1-0558.

Chapter 2

Related Work

This thesis builds upon a significant body of prior research on MapReduce, distributed data (query) processing and more importantly secure data processing in the cloud. In here, we will only present the prior work on secure data processing in the cloud and we will leave the discussion on the previous work in other areas (e.g. distributed data processing) to the corresponding chapter.

The secure data processing systems heavily relies upon the data encryption. The challenge of data processing over the encrypted data has been addressed extensively in the cryptography and database literature over the past decade. Numerous encryption techniques have been developed to support basic computations over the encrypted data. Unfortunately, none of these techniques is alone enough to support all types of computation. We will first review these techniques and their functionalities, many of which is summarized in [36, 37] and continue our discussion with the secure data processing tools.

2.1 Encryption Techniques

Searchable Encryption (SE): There are numerous searchable encryption techniques available up to date. [4–10] are only few of them and a recent work [38] provides a survey about the SE techniques. A searchable encryption schema enables an untrusted server to search in encrypted data without learning any information about the plaintext data. While some of the SE techniques directly applies the searching over the ciphertext [7], the other techniques work by creating a searchable encrypted index [5]. In such approaches, the encrypted documents and encrypted index is stored on a *honest-but-curious* server. To search the encrypted documents on the server, the client first generates a trapdoor using his/her secret key and then searches the encrypted index using this trapdoor. Without knowledge of trapdoors, the encrypted index leaks no information about its content [4]. Finally, once the search on encrypted index is done, the pointers to the appropriate encrypted documents are returned. The research in SE has three different dimensions: 1) efficiency of the approach 2) the security of the approach and 3) the search queries supported by the approach. Each different technique displays a different tradeoff across these three dimensions.

Order Preserving Encryption (OPE) : Introduced by [11], this technique encrypts the plaintexts in such a way that the order in their ciphertexts is same as the order in original plaintexts. In other words,

$$\text{If } x \geq y, \text{ then } Enc(x) \geq Enc(y) \tag{2.1}$$

This technique allows the untrusted server to perform order by, range and sort operations over the encrypted data. The recent work [12, 13], so called modular OPE, is a promising extensions of OPE that increases the security of the basic OPE. These new techniques adds a secret modular offset to each data value before encrypting them, i.e.

$$MOPE(x) = OPE(x + y) \tag{2.2}$$

where y is the secret offset. In this new approach, the adversary cannot learn any information about the locations of data values by looking at the encrypted data; whereas in basic OPE, half of the most important bits of plaintext locations can be leaked [12].

Deterministic Encryption (DTE) : A DTE technique encrypts a set of plaintexts in such a way the ciphertexts of two plaintexts will be equal if and only if the plaintexts are equal. Namely,

$$x = y \iff Enc(x) = Enc(y) \tag{2.3}$$

This is achieved through using the fixed encryption key. DTE enables the untrusted server to do equality checks or run point queries over the encrypted data. One of the most notable DTE techniques is presented in [14].

Fully Homomorphic Encryption (FHE) : Introduced by [15], FHE is a public-key encryption scheme that allows to evaluate various functions over the encrypted data. The function that will be evaluated is typically represented as a boolean circuit of polynomial size in the input size. These circuits often enable to compute both addition and multiplication. But, with the recent research on FHE, the new techniques are able to run any function over the encrypted data. FHE guarantees a strong security, that is the adversary who has the ciphertext and public key cannot learn any information about the underlying text, other than its length [37].

Even though there has been a huge ongoing research on improving the performance of FHE, FHE techniques are still too slow and costly in terms of storage. For instance, when an integer is encrypted using FHE, the ciphertext size is around 2^{14} bits. According to [37],

PHE CryptoSystem	Functionality
Paillier [17]	Addition
ElGamal [16]	Multiplication
Goldwasser-Micali [18]	XOR
BGN [19]	Arbitrary number of addition, 1 multiplication, Arbitrary number of addition

Table 2.1: PHE Techniques and Functionalities

running an AES block[1] [39] on the data encrypted with FHE takes around 40 minutes on a machine with a very large memory, which is six orders of magnitude slower than performing an AES block on the unencrypted data. Due to their efficiency impediments, FHE techniques are not used by the existing secure data processing tools.

Partially Homomorphic Encryption (PHE) : PHE and FHE are similar in spirit. The only difference is that PHE only allows to run a specific function over the encrypted data, such as addition or multiplication. PHE and FHE techniques offers the same strong security. Various implementations of FHE and their functionalities over the encrypted data are given in Table 2.1 .

Since the PHE schemes are more efficient and more practical than FHE, the encrypted databases systems use PHE instead of FHE. For instance, encrypting values takes around 9.5 ms and adding two encrypted values takes around 0.005 ms in Paillier cryptosystem on a commodity machine [37].

Bucketization : Bucketization technique, introduced in [20], partitions encrypted attributes into buckets by a privacy-aware partitioning function so that the exact information of which records are requested by a query is never revealed to the untrusted server. The partitioning function can be either order-preserving or random (not-order preserving). Such an

[1]The AES or Advanced Encryption Standard is a symmetric block cipher used by the U.S. government to protect classified information. It is implemented in software and hardware throughout the world to encrypt sensitive data.

approach effectively makes every record within a bucket indistinguishable from another. [40] and [41] analyze the loss of privacy due to bucketization. But, one can control its privacy loss by varying the sizes of the buckets. Pretty much any type of database operation can be computed over the bucketized data, but the false positives within the answer set must be filtered out on a secure side (client or private cloud).

Non-deterministic Encryption (NDTE) : Such encryption techniques use a *randomness* during encryption, so that when same plaintext is encrypted several times, as opposed to DTE, different ciphertexts will be yielded. NDTEs provide strong security, however they do no allow to run any kind of computation over the encrypted data. A typical implementation of NDET is to use block ciphers such as AES or Blowfish [42] in cipher block chaining mode with a random initialization vector.

2.2 Secure Data Processing Systems

Fueled with the advances in cryptography and cloud world, a new class of secure data processing systems that enable to use public clouds have emerged. In here we will give more details about such systems.

One of the early works in this area was database as a service model (DAS) [20]. DAS model explored support for SQL style queries with basic operations such as selection, projection, join etc. over the encrypted data. In DAS, when the execution could not continue on the encrypted domain, the intermediate data is transferred back to a secure side (client or private cloud), which could then decrypt the data and continue with the computation. The goal in DAS is to partition the query execution between the untrusted server and trusted client in such a way that the computation on the server side is maximized. DAS model enables users to make a tradeoff between the efficiency and the data confidentiality via bucketization

technique. In spirit, DAS model is similar to what we are trying to achieve in this thesis. Thereby, we can easily say that our research in this thesis has its roots in DAS model.

Another notable system, CryptDB [22], supports a wide-variety of SQL-like queries over encrypted data. To achieve this, each attribute in the table is encrypted in one or more *onions*. That is, each value is encrypted using a layers of increasingly stronger encryption techniques, where every layer of a onion supports a specific type of operation. Once data is encrypted using *onions*, the data is selectively decrypted at the untrusted server, just enough to support functionality required. Many similar systems to CryptDB are being designed by commercial vendors (e.g., SAP, Google) and by government research labs (e.g., NIS, Lincoln Labs). For instance, the design of Google Encrypted BigQuery [24] and No-SQL D4M Accumulo engine of Lincoln Labs [25] have the design influenced by CryptDB.

Cipherbase [21], developed at Microsoft Research, achieves high performance and high data confidentiality by storing and processing strongly encrypted data. The data is encrypted using FHE in such a way that addition, multiplication and comparison can be performed without decrypting the data. The operations that cannot be computed over the encrypted data, is handled over a FPGA based secure co-processors in the cloud. This work is in spirit very similar to the DAS model.

MONOMI [23], whose design is built upon CryptDB, was developed for executing analytical queries on encrypted data. MONOMI introduces several space and performance optimizations to speed up the query processing on the encrypted data. For instance, it creates some new attributes on the rows of the tables by doing some pre-computation in order to support certain class of operations given in the workload such as multiplication of two different attributes. Additionally, similar to DAS model, it applies conservative pre-filtering over the encrypted intermediate data before shipping them to the secure client to continue execution on the client side. Finally, MONOMI uses only the encryption techniques that is enough to execute given workload.

Another related work, namely Relational Cloud [43], uses the graph-based partitioning scheme described above to split data into private/public sides. The partitions are encrypted with multiple layers of encryption and stored on a server. A query is executed over encrypted data with multiple rounds of communication between a client and server without considering the cost of decrypting intermediate relations.

While encryption based solutions is an interesting solution for executing queries over outsourced databases, we provide a more general framework that intelligently partitions data and queries for hybrid clouds without using any encryption. Also, encryption based solutions aims to minimize disclosure risk, instead of mitigating it entirely, in order to get performance benefit. However, our sensitivity-based data and computation partitioning solution enables the user to obtain a huge performance gain without even exposing any sensitive information.

[44] considered the problem of securely outsourcing relational data using two non-colluding servers. The goal is to not give either server access to all attributes specified in a policy, which is defined in terms of subsets of attributes. While the two-server model can be mapped to our case, where the private cloud is both, a server and the trusted client issuing a query, their model of sensitivity is completely different from ours. While they do not allow all attributes specified in a confidentiality policy to be exposed to either server at any time, we are willing to expose any non-sensitive data to untrusted servers. This relaxation makes our solution approaches quite distinct from their approach.

Another prototype developed by Microsoft Research, VC3 [45] is the first practical framework that enables users to run secure MapReduce in the cloud. First, VC3 keeps the MapReduce code and data secret (data and code confidentiality). Second, it ensures that the MR job results will be correct and complete (data integrity). VC3 heavily relies on secure SGX processors to isolate memory regions on commodity machines, and to deploy new protocols that secure MapReduce tasks.

Additionally, there has been recent work related to the problem of securely partitioning data/computation over the hybrid clouds. [46, 47] explore this problem in the context of MapReduce. Their aim is to exploit public cloud machines as much as possible without leaking any sensitive information during MapReduce execution. For instance, Sedic, a privacy-aware MapReduce framework, proposed in [46] can shift the computation to the public cloud only in the map phase of the first MR job, which is not a very efficient solution. Our secure and efficient MapReduce framework, SEMROD, is able to utilize public cloud resources throughout the entire MR execution.

Chapter 3

Security Model

Despite numerous benefits, organizations, especially those that deal with potentially sensitive data (e.g., business secrets, sensitive client information such as credit card and social security numbers, medical records), hesitate to embrace the cloud model completely. One of the main impediments is the sense of "loss of control" over ones data wherein the end-users (clients) cannot restrict the access to potentially sensitive data by other entities, whether they be other tenants to the common cloud resources or privileged insiders who have access to the cloud infrastructure. The key operative issue here is the notion of trust. Loss of control, in itself, is not as much of an issue if clients/users could fully trust the service provider. In a world where service providers could be located anywhere, under varying legal jurisdictions; where privacy and confidentiality of ones data is subject to policies and laws that are at best (or under some circumstances) ambiguous; where policy compliance is virtually impossible to check, and the threat of "insider attacks" is very real. Trust is a difficult property to achieve. Loss of control over resources by migrating to the cloud coupled with lack of trust in the service provider poses numerous concerns :

- Data integrity : Service provider may serve the data inaccurately, or attacker may

corrupt the data itself.

- Data availability : User may not have access to its data all the time.
- Data privacy or confidentiality : Sensitive data may not remain confidential and would be vulnerable to misuse by other tenants, attackers or the service provider itself, while data is at rest or while data is being processed

While the cloud providers have to develop systems which addresses all of these concerns in order to attract more customers; in the context of this thesis, we only explore the challenge of data confidentiality in the public cloud environment.

Next, we will first discuss under what kind of sensitivity model we address the data confidentiality challenge. Subsequently, we will describe the adversarial model, against which we guarantee data confidentiality.

3.1 Sensitivity Model

In the context of this thesis, we will consider a *record level sensitivity model* wherein records of relations are classified as either sensitive or non-sensitive. For instance, in a relational dataset, we consider that each tuple could be either sensitive or non-sensitive. Similarly in the context of MR, the input key-value pair could be in either of two states: sensitive or non-sensitive. Note that sensitivity in database literature might be modeled at level of columns/attributes, if one desire finer grained sensitivity to only selected attributes, we can still use our sensitivity model by vertically partitioning the records into two relations collating all the sensitive attributes together. Theoretically, thus, the restriction to record level sensitivity does not sacrifice the generality of our approaches, though it will likely require slight modification to our prototypes to deal with the partitioned relation.

Our sensitivity model is independent of the exact method used to specify sensitivity *e.g.*, users may specify which records are sensitive by defining predicates. The only assumption we make is that non-sensitive records by themselves do not contain any information about sensitive records. That is, even if the adversary could gain access to all non-sensitive records, it would not infer any sensitive information.

Furthermore, regardless of the computation being executed, any data generated in the course of a data processing is intrinsically considered to be sensitive, if it is generated as a result of processing at least one sensitive input. To state this formally, for any given data processing function F and any instance of it, namely $F(r_1, r_2, ...r_t) = r_{out}$ where r_1 to r_t are the input records and r_{out} is the output record,

$$sens(r_{out}) = \begin{cases} true, \exists\ 1 \leq j \leq t, r_j \text{ is sensitive} \\ false, \text{otherwise.} \end{cases} \tag{3.1}$$

Note that, the sensitivity model we have employed during a data processing is conservative, since we consider any function's output to be sensitive if it is sourced from at least one sensitive input record. For many cases, such an assumption can be relaxed. In general, the output of a function F may not reveal any sensitive information about its input data. For such F, a new sensitivity model where all outputs are directly considered to be non-sensitive can be introduced. While exploring support for sensitivity models that vary based on the specifics of the given computations is an interesting direction of exploration, we restrict our attention to only the conservative model for two reasons. First, the conservative model is more general and works regardless of the specific semantics of the given data processing task. Second, it suffices to establish the feasibility of the solution. In particular, if we establish an efficient data processing framework that guarantees that no sensitive data is leaked under a conservative model, we can further improve the framework to exploit specific semantics of data processing tasks being executed.

Note that our sensitivity model does not explicitly deal with the inference control problem wherein users may consider some inferences (derivable from non-sensitive data) such as association rules as sensitive while the data itself is deemed nonsensitive [48]. Inference control problem has been studied in the security literature in various contexts: statistical databases [49], multi-level relational database systems [50, 51], data publishing to name a few [52, 53]. For instance, in [53] the authors discuss a problem of publishing view on data that does not reveal information about sensitive queries. Likewise, [48] focuses on the problem of limiting the disclosure of sensitive association rules while minimally affecting the non-sensitive associations.

Our goal, in this thesis, is limited to preventing loss of sensitive data and we consider inference control to be a related but complementary problem. We note that many proposed secure data processing frameworks take a similar view (e.g., SQL databases that support access control such as Oracle [54]). Such systems prevent exposure of sensitive data but do not directly address sensitive inferences. One of the reasons is that despite significant research, inference control, in general, has remained a difficult open problem even in traditional data processing systems. Of course, our solutions will prevent sensitive inferences, if the user conservatively marks all the data which could lead to the sensitive inferences as sensitive. Exploring and incorporating more effective approaches to inference control wherein user's can specify sensitive inferences and the system steers data in ways to prevent inferences remains an interesting future direction of exploration.

Finally, we note that our sensitivity model only deals with sensitivity related to data. Sensitivity may instead be associated with the computation – e.g., if the user wishes to execute a proprietary analysis in which the SQL code or MapReduce code is sensitive. For such a case, likely hybrid cloud setting is not appropriate since the adversary will gain knowledge of the corresponding computation creating a potential vulnerability. In our framework, we will assume that the computation (e.g. query or MapReduce job) itself is not sensitive.

3.2 Adversary Model

There has been significant amount of research in the literature about providing security against different types of adversaries. Usually, in the context of security as well as cloud computing, the adversaries are classified based on their capabilities and intentions.

First, an attacker can be either computationally bounded or unbounded in terms of the storage, computation and time resources. In the computationally bounded model, the adversary is able to perform a reasonable amount of computation within a limited time to learn sensitive information; whereas in computationally unbounded model, the attacker has no such limits.

Second, an attacker can be classified based on his/her control over the network. The attacker can passively listen the incoming and outgoing channels of the cloud (eavesdropping). Or the attacker is able to actively corrupt the messages in the network channels (Byzantine attacks). Note that both attacker models can be very dangerous for an organization using the cloud environment, since they are usually connected to their cloud provider via a public wide area network, rather than a special trusted network.

Third, an attacker might have a fixed behavior (static) or his/her behavior can change based on the results of the computation on the cloud (adaptive). For instance, in the context of data processing, the static attacker sitting on the public cloud can only learn sensitive information from the given input data, whereas in adaptive model, the attacker can change its strategy, and apply certain statistical/inference attacks based on the intermediate or output data generated by the computation.

In the scope of this thesis, we will assume that the private cloud is trustworthy and that the adversary has neither access to private cloud machines nor their communication channels. Thus, an adversary cannot launch any attack against private cloud machines to acquire sen-

sitive information. In contrast, the adversary can have full control of the public cloud and can easily access any input/intermediate/output file generated on the public cloud and can design its attacking strategy based on the input as well as the intermediate and output data (*adaptive*). In addition, the adversary can view the locations to which the output of public computations are shipped. Furthermore, the adversary can *eavesdrop* on the communication channel between the public and private clouds, but cannot actively corrupt the communication. Also, the adversary is *computationally unbounded*, so he/she can launch any type of statistical/inference attacks to gain knowledge about the sensitive data stored on the private nodes. The time or storage is not a concern for him/her to learn sensitive information. Finally, the adversary is assumed to be *honest-but-curious* [55]. Thus, while the adversary correctly computes the tasks assigned to public machines, it exploits the knowledge gained to try to derive as much information as possible about the sensitive data and computation.

Under such an adversarial model, our goal is to guarantee data confidentiality, that is no sensitive data or information that can provide inferences about the sensitive data is leaked to the public cloud during data processing.

Chapter 4

Secure and Efficient MapReduce over Hybrid Clouds

4.1 Introduction

For large scale data analyses, MR-based systems [56, 57] have been widely used because of their high-scalability, fault tolerance and easy-to-use features. In the MR paradigm, users specify a *map* task that processes an input record to generate a set of intermediate key-value pairs, and a *reduce* task that merges all intermediate values with the same intermediate key. The runtime system executes the *map* and *reduce* tasks in parallel over a cluster of machines. To enable parallelization, systems such as Hadoop [57] and Google MR are supported by an underlying reliable distributed file system. Current MR systems, when deployed over hybrid clouds, do not take into account the sensitivity of information when storing/processing data. Such MR systems could thus disclose potentially sensitive information to other tenants residing on the public cloud or to privileged insiders having access to the public cloud infrastructure. As a result, organizations for whom security is indispensable are unable to

deploy MR jobs over their hybrid cloud environments.

Design Principles: Our goal, in this chapter, is to design an MR framework for hybrid clouds that follows each of the principles given next.

- *Secure*: Our new framework must be secure against honest-but-curious public cloud. We identified two types of sensitive information leakage that can occur in hybrid clouds in the context of MR jobs; *viz. direct exposure*, where the sensitive data is directly transmitted to a public machine for either storage or computation purposes; and *key-inference exposure* where an adversary can infer the association of an intermediate key, generated during MR processing, with sensitive data. Our new framework must prevent both of these exposure scenarios.

- *Enable Public Cloud Usage*: To provide security, one option would be to do the entire computation at the private cloud. However given that private clouds has a limited computational power, this would not be an efficient solution. Thereby, our new framework needs to move as much computation to the public cloud as possible.

- *Easy-to-use* The system should not put any extra burden to the user. It should minimally change the way end-user interacts with the MR framework. Also, it should support a convenient migration of existing jobs to the new execution framework.

- *Generic* This framework should not be focusing primarily on specific type of MR Jobs, rather it should provide efficiency and security for all types of MR Jobs.

- *Efficiency* The system should be more efficient then other secure MR solutions, such as running everything at the private cloud. Also, the overhead incurred by security assurance should be low compared to a native MR framework.

To address all these design principles, we propose SEMROD, the abbreviation stands for a secure and efficient MR framework for hybrid clouds. SEMROD uses a modified distributed

file system to store data and allows both, mappers and reducers, to execute on public clouds, while guaranteeing that no sensitive data is exposed. This is achieved by implementing two additional mechanisms over the original MR framework. First, SEMROD performs an additional "sensitive key analysis" phase between the map and reduce phases to determine intermediate keys with which sensitive data may be associated. Such keys must be reduced on the private side since part of the intermediate data for such keys cannot be shipped to the public side. Second, SEMROD shuffles key-value pairs generated on the public side to a public and private reducer. Such an extra shuffling enables reduce computation over non-sensitive data to be performed over public machines, without incurring any *key-inference* exposure.

The main contributions of this chapter are as follows:

- We identify the types of sensitive information leakage that could occur in the course of MR job execution in the presence of a *honest-but-curious* adversary on the public cloud.
- We propose SEMROD, a secure and efficient framework to execute single and multi-level MR jobs over hybrid clouds. SEMROD ensures full safety against the identified types of leakage while allowing execution of both, map and reduce operations, on public machines.
- We show how SEMROD can be easily integrated and implemented in one of the most popular MR implementations, namely Hadoop.
- We conduct experiments to show the efficacy of our approach *vs.* other secure alternatives. Our results show that SEMROD dramatically improves the total MR execution time of relational join, selection and aggregation queries as compared to other solutions

We note that there have been a few attempts in building secure MR frameworks for hybrid

Notation	Description
J	Given sequence of MR jobs, $J = \{j_1, j_2, \ldots, j_n\}$.
Inp^i	Set of input records for j_i.
Int^i *and* K^i	Set of intermediate key-value (KV) pairs and keys generated by the job j_i.
Out^i	Multi-set of output KV pairs produced by job j_i.
$M^i(r)$	Map (+ Combiner) function of the job j_i where $r \subseteq Inp^i$.
$R^i(k)$	Reduce function of the job j_i where $k \in K^i$.

Table 4.1: Notations

clouds in recent literature [46]. Existing techniques limit themselves by either running map or reduce tasks (but not both) on public machines and focus only on a single level MR job. As such these techniques exhibit significant limitations specially in the context of multi-level MR jobs or reduce heavy jobs. We will compare SEMROD to existing state of the art in details in the rest of the chapter organized as follows: Sections 4.2 and 4.3 provide an overview of the original MR framework and the exposure models. Next, Section 4.4 introduces our secure and efficient MR mechanism, namely SEMROD. In Section 4.5, we present the implementation details of our techniques over Hadoop, a well known open-source MR framework. Then, Section 4.6 formally and experimentally compares the performance of SEMROD with other secure solutions and the original Hadoop implementation for a variety of tasks. Finally, we conclude the chapter and discuss extensions to further improve SEMROD's performance in Section 4.7.

4.2 Preliminaries

In this section, we provide a brief overview of MR which forms the basis of our work. To better state the ideas presented in the remainder of this chapter, we use the notations described in Table 4.1.

4.2.1 MapReduce Framework

MapReduce (MR) is a framework that have been widely used for large scale data analysis, because of its high-scalability, fault tolerance and easy-to-use features. In the MR paradigm, users specify a map task that processes an input record to generate a set of intermediate key-value (KV) pairs, and a reduce task that processes all intermediate values with the same intermediate key. The runtime system executes the map and reduce tasks in parallel over a cluster of machines.

MR framework allows storing and processing large volumes of data using a Distributed File System (DFS) as its underlying storage mechanism. A DFS provides numerous advantages such as data replication, fault tolerance and reliability. In MR, a *master* node manages the entire file system by keeping track of how blocks of files are distributed over all slave nodes. A process running on every slave node manages the storage infrastructure for that node. MR jobs process data stored in DFS in a parallelized fashion over a cluster of nodes. In MR framework, the master is responsible for scheduling an MR job among the slaves while a slave node is responsible for executing the given sub-task. An MR job consists of the following operations:

• *Map* operation takes key-value pairs as input in order to generate a list of intermediate key-value pairs. Namely,

$$map : (k_1, v_1) \rightarrow list(k_2, v_2). \tag{4.1}$$

Map operations are usually distributed across a cluster by automatically partitioning the input to possibly equivalent sized splits, which can then be processed in parallel by separate machines. The assignment of a particular map task to a slave node is done by the master. Once a map operation gets completed on a slave, the intermediate key-value pairs are stored

as partitions in local disks on that slave. Furthermore, the locations of these outputs are transferred to the master, which in turn will forward them to reduce tasks.

• An intermediate *shuffle* operation, in which reducers use remote procedure calls to fetch their inputs from local disks of slaves. Note that, the reducer is provided the locations of the slaves, from which it needs to read data, by the master.

• An intermediate *sort* phase, in which reducers upon receiving their input data (as a result of *shuffle*) sorts the incoming key-value pairs such that all the key-value pairs, having the same key, are grouped together.

• A *reduce* operation, in which a reducer processes all pairs that share a common key. This operation can be defined as follows:

$$reduce : (k_2, list(v_2)) \rightarrow list(k_3, v_3). \tag{4.2}$$

Reduce tasks are distributed across the cluster by partitioning the intermediate key space into $|R|$ parts by using a partitioning function (the default is: $hash(k_i)\ mod\ |R|$), where $|R|$ denotes the number of reduce tasks. The user is allowed to specify the number of reducers, and in turn the number of partitions. Furthermore, the user can specify a partitioning function that achieves a more balanced partitioning of the intermediate key space.

Figure 4.1 illustrates how a single MR job j_i is processed by the MR framework using the notations described in Table 4.1. Note that, input data for job j_i, Inp^i, comprises the previous MR job's output data, Out^{i-1} (if there exists any) and some external data Ext^i.

In certain cases, MR framework allows users to define an optional *combiner* function that performs a partial reduction on the map output data before it is delivered to reduce tasks. This reduces the amount of data transferred over the network. The combiner function is applied to each map task's output and its result is stored in local disks of slave nodes.

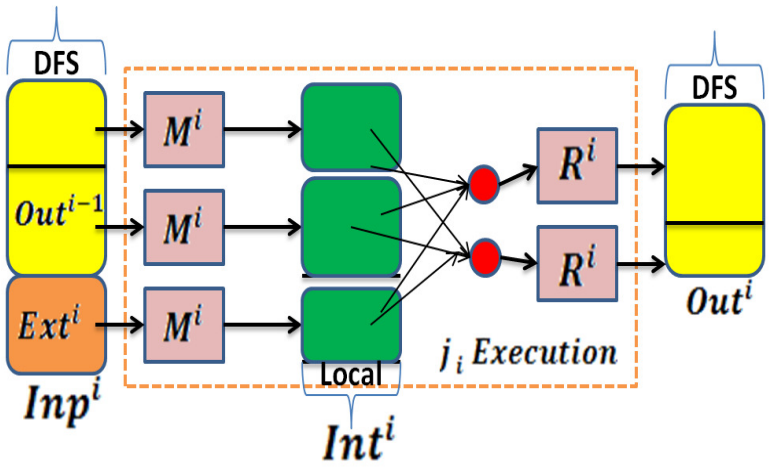

Figure 4.1: Typical MR Job Execution

Another performance enhancement technique applied in MR frameworks is to (partially) overlap the execution of map and reduce tasks. In particular, a reduce task can begin shuffling whenever some portion of map tasks have completed. This feature is called *slow-reduce start* and it enables parallel execution of map and reduce tasks rather than executing them sequentially.

4.3 Security Challenges

In this section, we identify all the security related challenges when running MR jobs over a hybrid cloud. Furthermore we provide all the security and performance motivations behind the introduction of SEMROD..

4.3.1 Exposure Scenarios

Given our adversarial model, there can mainly two types of exposures : *Direct Exposure* and *Key-inference Exposure*, that could arise while either storing the data or executing MR jobs

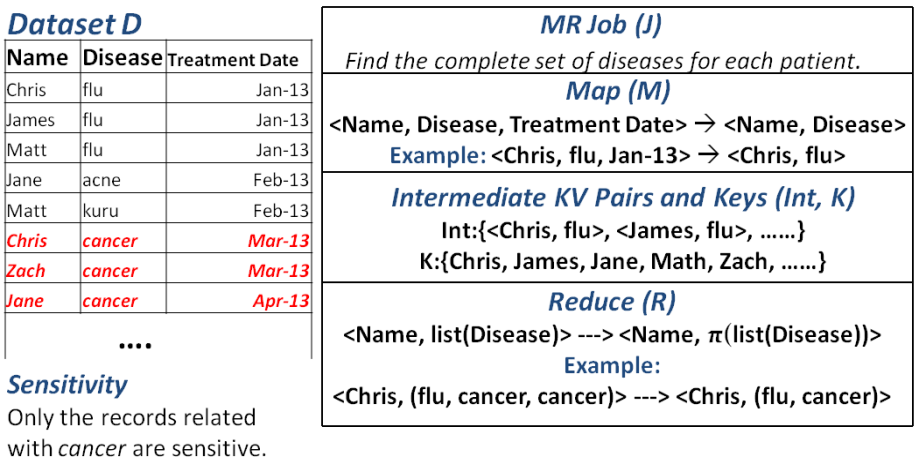

Figure 4.2: Secure MR Processing Example

on public machines. Let us illustrate these exposure scenarios in the context of an example shown in Figure 4.2, which is also used later to motivate SEMROD.

Suppose that an analyst, given dataset D about patients, wishes to compute a complete list of diseases for each patient. The analyst defines an MR job j for this purpose, where the map task emits a key-value pair consisting of the patient's name, and the disease they have. The reduce task will take all the records corresponding to a single patient to compute the complete list of diseases for the patient. Note that, this example is very similar to creating an inverted list of words given a set of documents; an example frequently used to illustrate the MR model. In the dataset, let us assume that the input records associated with the *cancer* disease are sensitive. We first note that records such as $\langle Chris,\ cancer,\ Mar-13\rangle$ cannot be stored/mapped on public machines since the attacker will have direct access to such data. Thus, the last 3 records in Figure 4.2 must be assigned to the DFS partition on private machines. Given our sensitivity model, after the map phase, intermediate $\langle Name,\ disease\rangle$ KV pairs, whose value contains *cancer* (*e.g.*, $\langle Chris,\ cancer\rangle$) are sensitive. Shuffling such intermediate KV pairs to public reduce tasks will again directly disclose the sensitive data to the attacker. In other words, **direct exposure** occurs when any public machine either

receives a sensitive record as an input for a map task, or gets an intermediate sensitive KV pair as an input for a reduce task.

To prevent the direct disclosure of sensitive data to an attacker, sensitive input records (*i.e.*, records with *cancer* disease) have to be stored and mapped over private (secure) machines. In addition, the intermediate keys associated with at least one sensitive map output KV pair such as $Chris$, $Jane$ and $Zach$ have to be reduced by private reduce tasks. Simply preventing sensitive data from being stored or shuffled to public machines does not suffice to prevent sensitive data leakage, as we show next.

Suppose that all non-sensitive records in D are stored and mapped by public machines. Also, only intermediate keys belonging to $James$ and $Matt$ are reduced by public reduce tasks. As a result, the public map outputs, $\langle Chris,\ flu\rangle$ and $\langle Jane,\ acne\rangle$, will be shuffled to a private reduce task, whereas the remaining public map outputs such as $\langle James,\ flu\rangle$ and $\langle Matt,\ acne\rangle$ will remain on the public side. In such a case, the attacker can infer that $James$ and $Matt$ have no associated *cancer* record. Furthermore, the attacker's chance of correctly guessing whether $Chris$ or $Jane$ have *cancer* will increase compared to his/her initial knowledge based on the data distribution while storing the data into DFS and during the map phase. Due to such key-inferences, the MR framework will be more vulnerable to statistical attacks that aim to reconstruct sensitive input files. We refer to such indirect loss of sensitive data as **key-inference exposure**. Key-inference exposures arise when data is selectively shuffled from public machines to reducers executing on private machines. In the example above, selectively shuffling Chris and Jane's records to private machines while reducing Matt and James's records on the public side provide insight about the sensitive data to the adversary.

4.3.2 Preventing Sensitive Data Leakage

Preventing both direct as well as key-inference exposure of data during MR execution poses interesting challenges in designing an MR framework . Since intermediate keys generated during the execution may dynamically become sensitive (since some sensitive record generates the same intermediate key), we can no longer statically partition the key space and assign the resulting reducers to machines prior to execution, as is typically done by the MR framework. Instead, such a partitioning and assignment must now be done dynamically to ensure no direct exposure of sensitive data occurs.

Preventing key-inference exposure requires that public machines must not be able to distinguish amongst intermediate keys generated by them, since a different treatment of different keys could lead the adversary to make inferences about the sensitivity of the key. Two straightforward solutions for preventing key-inference attacks can be as followsL The first is to perform map operations only on private machines, whereas, reduce tasks (for intermediate keys that do not become sensitive) could be submitted to public machines. Another alternative would be like *Sedic*, has recently been proposed in [46]. Sedic performs map operations on the public (and private) side, but shifts all the reducers to private machines[1]. Intuitively, both techniques prevent exposure – they never submit any sensitive data to the public side (thereby preventing direct exposure), and also the treatment of each intermediate key on the public side is exactly the same. In the first solution, all such keys are reduced, while in the second, all keys are shifted to the private side. In this way, Sedic prevents the key-inference attacks. Neither of the two solutions mentioned above are satisfactory, especially in the context of data management workloads, because of the following reasons: (1) A MR framework over hybrid clouds is most effective from the perspective of computational performance when both, map and reduce, can be be assigned to public machines. However,

[1]Sedic automatically generates combiners, when possible, to reduce the amount of work done during the Reduce phase so as to best exploit public machines.

neither of the two alternatives presented above executes both phases on public machines. (2) Data management workloads typically comprise of tasks that require multiple MR jobs (*i.e.*, multi-level MR jobs). However, none of the two techniques presented above provide support for executing multi-level MR jobs.

In contrast, SEMROD follows a different philosophy. Instead of limiting the entire computation to the private side in one of the phases, it allows work to be done on the public side throughout the entire execution. The main issue is that the work done at the public side should not incur any type of sensitive data exposure. SEMROD achieves this at the expense of allowing some redundant computations to occur on the private side. The details of SEMROD's MR model are provided in the next section.

4.4 SEMROD

Standing for secure and efficient MR framework for hybrid clouds, SEMROD is described separately in the context of single and multi-level MR, as they mandate a special care for efficiency. We begin by describing modifications required at the storage layer to ensure security against direct exposure.

At the DFS layer in SEMROD, input files are classified as either sensitive or non-sensitive. Record-level sensitivity can be completely captured by partitioning the sensitive and non-sensitive records into separate files. Such a partitioning can be achieved automatically by SEMROD, unless the files are already created so. However, a metadata file that identifies the sensitive records in the input files can be prepared and given along with the original file. Such metadata can be in the form of set of views (predicates) or a list of offsets that identifies the sensitive records. While uploading the files, a module embedded to master can iterate over records and split them into two sub-files based on the provided metadata.

SEMROD maintains sensitive file blocks only on trusted machines and never replicates them to the public side. On the other hand, while storing and replicating non-sensitive data, no constraints are applied and it is handled as in the original MR framework.

4.4.1 Single-level MR

4.4.1.1 Map

In SEMROD, since input files either contain sensitive records, or do not contain any sensitive records at all, the input splits created during the map phase are also pure - i.e., either all records in the split are sensitive or none of them are. SEMROD labels the split as sensitive or non-sensitive based on the input file used to create them. After labeling, each input split object intrinsically determines if the map task running on that particular split is sensitive or not. The master strictly runs on a private (secure) machine and schedules the sensitive map tasks exclusively on the private side (trusted) machines. In contrast, non-sensitive map tasks are scheduled preferably on public machines. Only when no slot is available on public nodes and no waiting map tasks remain in the task queue, non-sensitive map tasks are assigned to private machines.

SEMROD needs to deploy several enhancement techniques at the map stage *a priori* so that during the reduce stage the master is able to discern what is sensitive and non-sensitive within the map's output, and further, divert the non-sensitive key-value pairs to public reducers without resulting in any sensitive data leakage or causing a large performance overhead. Additionally, these modifications enable master to track the incorrect data after the reduce tasks.

Figure 4.3 illustrates the map phase in SEMROD with 3 map tasks, $M_1^1 - M_3^1$, which are executed over two non-sensitive and one sensitive data block. In this example, M_1^1 and M_2^1

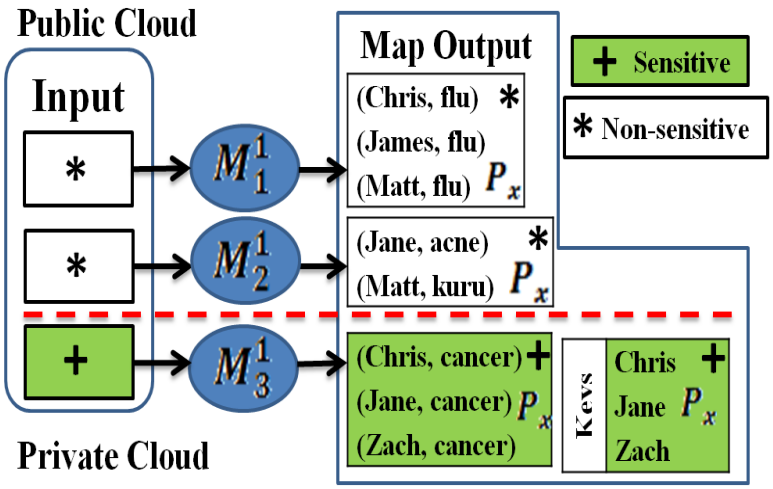

Figure 4.3: The map phase of Single-level SEMROD

runs on the public cloud, whereas M_3^1 is executed on a private machine. The figure also portrays the concept of sensitive key set collection for an arbitrary partition, P_x.

Sub-Partitioning: The sensitive and non-sensitive map (or combine) output KV pairs should be separated before storing them on local disks. Otherwise, if a map output partition contains both sensitive and non-sensitive records at the same time, then its sensitive portion would need to be sanitized before it is shipped to a public reducer. This sanitization operation is expensive, as one needs to parse and subsequently repack the output partition. Hence, SEMROD deploys the sub-partitioning concept across all map tasks.

Note that, each partition represents a different key space among the map's outputs and SEMROD divides every P_i into 2 sub-partitions (P_{i_0} and P_{i_1}) for each map task: the first of which is for sensitive key-value pairs in that key region, whereas the second one is for non-sensitive ones. Observe that P_{i_0} and P_{i_1} belong to the same key space and can have some key-value pairs that share a common key. Since we use file level sensitivity, for a given map task $M_j \in M$, either all P_{i_0}s or P_{i_1}s are empty depending upon whether the input block given to M_j belongs to a sensitive or a non-sensitive file. That is, $\forall M_j \in M$ either $P_{i_0} = \emptyset$ or $P_{i_1} = \emptyset$.

Sensitive KeySet Collection: For each partition P_x, the map tasks running on sensitive file blocks dynamically collect the set of intermediate keys that they generate for that partition. Once these map tasks finish, they store the collected sensitive key sets per partition in a separate local file. In Figure 4.3, M_3^1 is the map task processing sensitive input split, and therefore, it only generates a sensitive key set for partition P_x and maintains it locally. Later, the reduce task, which handles partition P_x, will first read sensitive key sets from each sensitive map task output and obtains the overall set of sensitive keys in P_x.

4.4.1.2 Reduce

During the reduce phase, SEMROD attempts to shift as much of the non-sensitive load (the execution of which will not reveal sensitive data) as possible to the public machines, while at the same time it will be balancing the load amongst the reducers. This requires several structural changes to how task scheduling is done in the MR model.

The reduce phase in SEMROD is depicted in Figure 4.4. Our model splits an original reduce task, $R_x^1 \in R^1$, into 2 reduce tasks: $R_{x_{priv}}^1$ and $R_{x_{pub}}^1$. Suppose that, R_x^1 is created to reduce partition P_x by the original MR framework. In SEMROD, $R_{x_{priv}}^1$ is designed to reduce only the set of sensitive keys in P_x (*e.g.* $Chris$ and $Jane$). On the other hand, $R_{x_{pub}}^1$ is created to reduce the remaining non-sensitive keys in P_x, such as $James$ and $Matt$. As should be evident from the notations, $R_{x_{priv}}^1$ and $R_{x_{pub}}^1$ are respectively assigned to a private and public node.

As shown in Figure 4.4, $R_{x_{priv}}^1$ first retrieves the sensitive key sets from only the sensitive map outputs and creates $K_{x_s}^1$, the overall set of sensitive keys within partition P_x. In our example, $K_{x_s}^1 = \{Chris, Jane, Zach\}$. Next, $R_{x_{priv}}^1$ pulls all the map output KV pairs within P_x, eliminates the ones with a key $\notin K_{x_s}^1$ such as $(James, flu)$ and applies the reduce function over the rest of them. Lastly, $R_{x_{priv}}^1$ stores its sensitive output file in DFS.

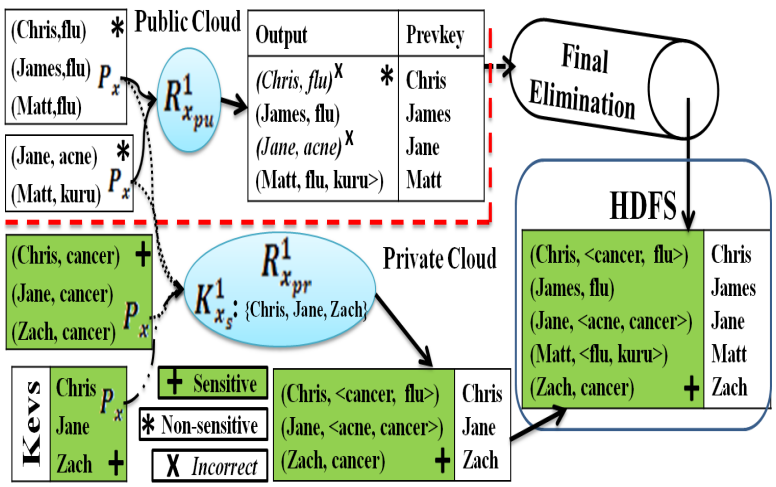

Figure 4.4: The reduce phase of Single-level SEMROD

In case of $R^1_{x_{pub}}$, it retrieves the data for partition P_x exclusively from the non-sensitive map task outputs (*e.g.* M^1_1 and M^1_2 in our example) and applies the reduce function over all incoming KV pairs without any filtering. As a result of following this protocol, $R^1_{x_{pub}}$ may generate some incorrect outputs. To illustrate, in Figure 4.4, reducer $R^1_{x_{pub}}$ receives only the pair $(Chris, flu)$ from amongst all intermediate KV pairs with key $Chris$, and in turn, outputs $(Chris, flu)$ pair again. However, our MR job aims to find the complete set of diseases for each person. Unfortunately, $(Chris, flu)$ pair does not include all of Chris's diseases such as *cancer* and thus it must be considered as incorrect. In fact, incorrectness occurred in $R^1_{x_{pub}}$'s outputs due to reducing keys such as $Chris$ and $Jane$ without certain sensitive KV pairs associated with them.

Therefore, once $R^1_{x_{pub}}$ produces its final result; a filtering step, called *final elimination*, is applied to it on the private cloud in order to remove incorrect outputs. To be able to differentiate between what is correct or incorrect in $R^1_{x_{pub}}$, the source intermediate key of each reducer output ($Prevkey$) is appended to the output itself. For instance, $(Matt, < flu, kuru >)$ is tailed with *prevkey* $Matt$ because it is generated by reducing the intermediate key $Matt$. Once all *prevkeys* have been appended to the outputs, the master can trivially

identify the incorrect KV outputs using their prevkeys. Note that if a prevkey is present in both $R^1_{x_{priv}}$ and $R^1_{x_{pub}}$ outputs, then any KV pair with that particular prevkey in $R^1_{x_{pub}}$ would be incorrect. To demonstrate this fact, $< Chris, flu >$ and $< Jane, acne >$ are the only incorrect pairs amongst $R^1_{x_{pub}}$'s outputs, and their *prevkeys* are $Chris$ and $Jane$ respectively. Note that $Chris$ and $Jane$ are the only common *prevkeys* between $R^1_{x_{priv}}$ and $R^1_{x_{pub}}$ outputs. Finally, the results of both, $R^1_{x_{priv}}$ and *final elimination*, are declared as sensitive and migrated to DFS. Remember that this reduce protocol is repeated for every reduce task R^1_x.

Another systematic difference of our reduce phase with respect to the traditional reduce stage is related with the time when reduce tasks are initiated. The master does not start shuffling and sorting in private reduce tasks ($R^1_{x_{priv}}$'s) until all sensitive input blocks are completely mapped. Suppose that, M^1_3 has not processed $(Jane, cancer)$ record yet and in turn the key $Jane$ has not been sent to the master. In this case, $K^1_{x_s}$ created at the master will not contain the key $Jane$ and $R^1_{x_{priv}}$, given in Figure 4.4, will be started with the incomplete $K^1_{x_s}$. Because of this, it will discard all the incoming KV pairs with key $Jane$. Thus, Jane's list of diseases will never emerge in the job outputs. In summary, private reduce tasks are not started until the end of the map phase.

Observe that, in the course of our reduce phase, no sensitive intermediate KV pair is shuffled to any public machine, and therefore, it is safe against *direct exposure*. In addition, all public map task outputs are steered to both, public and private reduce tasks (*i.e.*, no selective shuffling to public nodes), which makes our reduce protocol secure against *key-inference exposure* as well. A significant detail to note is that we never provide any key set to the public reducer $R^1_{x_{pub}}$. Indeed, asking the public reducer to operate over specific keys will lead to *key-inference exposure*. Therefore, $R^1_{x_{pub}}$ is designed to reduce all incoming KV pairs. Finally, the outputs of all $R^1_{x_{priv}}$s and the elimination operation are tagged as sensitive. The reason behind tagging $R^1_{x_{priv}}$'s output as sensitive is straightforward; however, labeling the

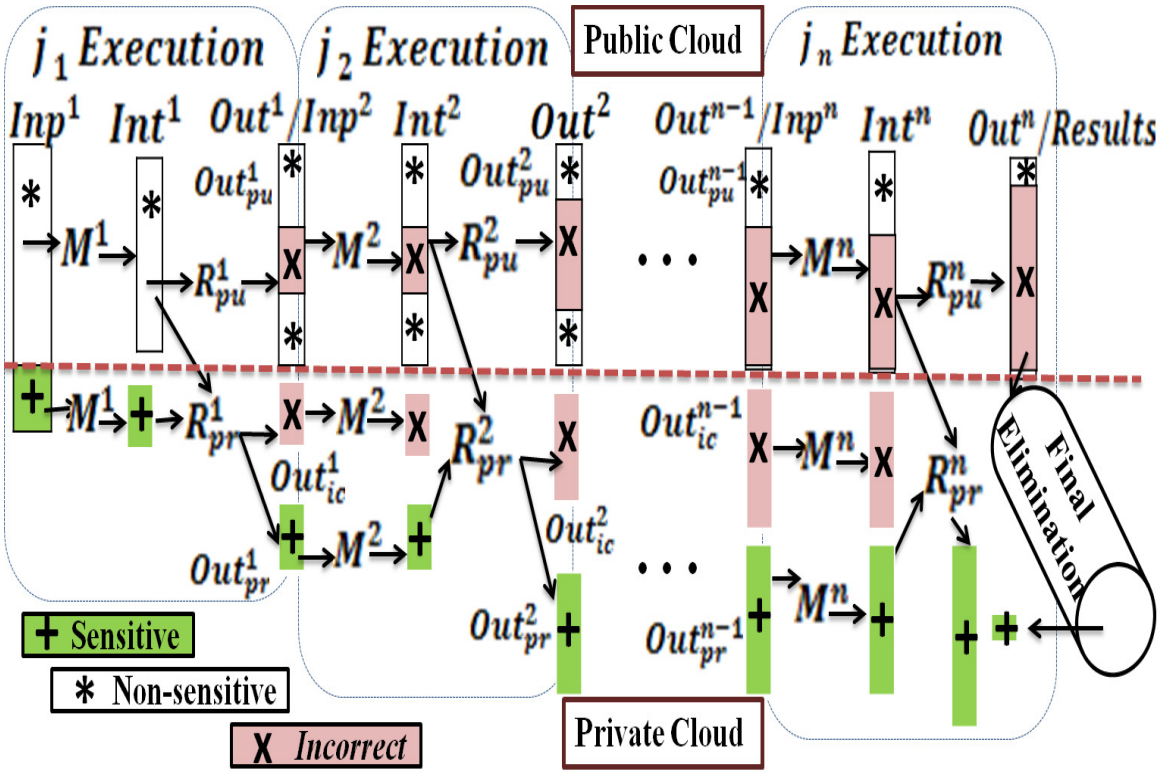

Figure 4.5: SEMROD Overview

output of the *final elimination* step as sensitive may seem unreasonable. Recall that, the adversary has full access to public resources at all times. If the *final elimination* output is somehow revealed to public machines in the future (*e.g.* to be stored or as an input to a subsequent MR job), then the adversary can identify the missing keys by comparing $R^1_{x_{pub}}$'s output and the elimination output. To eliminate such a leakage, SEMROD labels all job outputs as sensitive.

4.4.2 Multi-level MR

4.4.2.1 Overview

As mentioned earlier, once the single-level SEMROD framework is applied over the input data, it could generate some incorrect results. In this situation, job j_1's public reduce task outputs should be refined on the private side to obtain the complete set of correct results, and furthermore, all outputs now need to be declared as sensitive in order to avoid any key-inference exposure. Due to this restriction, SEMROD is unable to migrate subsequent map and reduce tasks working on j_1's output to public machines, which obviously leads to a highly unbalanced execution for job j_2, especially when j_2 solely depends on j_1's output. However, by deferring the *final elimination* step over j_1's public output, j_2's execution can still be distributed to both sides of the hybrid cloud in a secure fashion, and yet all the correct outputs can be accumulated on the private side after completing j_2.

The rest of chapter deals with the case in which some incorrect input values are given to job j_i by the previous job's public reduce tasks. Incorrect input values to j_i would result in incorrect output for j_i. In general, incorrectness will spread along the execution path. To eliminate such incorrect outputs, we first need to formally identify how incorrectness spreads by a job $j_i \in J$.

Incorrectness: While executing $j_i \in J$, incorrectness could occur because of the following three reasons:

• Reducing any $k_t \in K^i$ partially results in "incorrect" reduce outputs, *i.e.*, all outputs of $R^i(k_t)$, where $\exists\ (k_t, v_1) \in Int^i\ s.t.\ (k_t, v_1)$ is not given as input to it.

• If $r \in Inp^i$ is an "incorrect" input record, all outputs of $M^i(r)$ are "incorrect".

• All outputs of $R^i(k_x)$ are "incorrect", if it takes at least 1 incorrect input.

Given that an input to a job j_i might be incorrect in SEMROD, we define a concept of sound and complete MR execution for job j_i. Intuitively, the execution of j_i is sound if all output that results from incorrect input tuples can be filtered from the results. Likewise, the system is complete if every output tuple that could result from executing j_i on correct input is present in the output results. We formally define the concept of soundness and completeness below:

Let Inp^i_c, Inp^i_{ic}, Out^i_c and Out^i_{ic} be the set of correct and incorrect input/outputs of job j_i respectively;

Soundness: MR' is "sound" for j_i if the master is able to filter out the Out^i_{ic} from Out^i.

Completeness: MR' provides a "complete" execution for j_i if $MR(Inp^i_c) = Out^i_c \subseteq MR'(Inp^i) = Out^i$. In other words, the complete set of results that generated by the original MR framework, Inp^i_c, are a subset of the results generated by the new framework MR', Out^i.

Figure 4.5 depicts the execution flow of a multi-level MR job, J, in SEMROD. For the sake of simplicity, the execution details of jobs j_1, j_2 and j_n are specifically shown. The execution of the remaining jobs, $j_3, j_4, \ldots j_{n-1}$, will be exactly similar to j_2.

Note that, the first job's input, Inp^1, does not contain any incorrect data. In fact, incorrectness is solely initiated by reducing some intermediate keys in the course of job execution. A major feature of our multi-level model is that private reduce tasks (except those of the final job j_n) are modified to create an incorrect output replica, Out^i_{ic}, on private machines. This replica will later be used to track how incorrect data is dissipated through public side computations so that the "soundness" feature is still satisfied even when numerous MR jobs are executed.

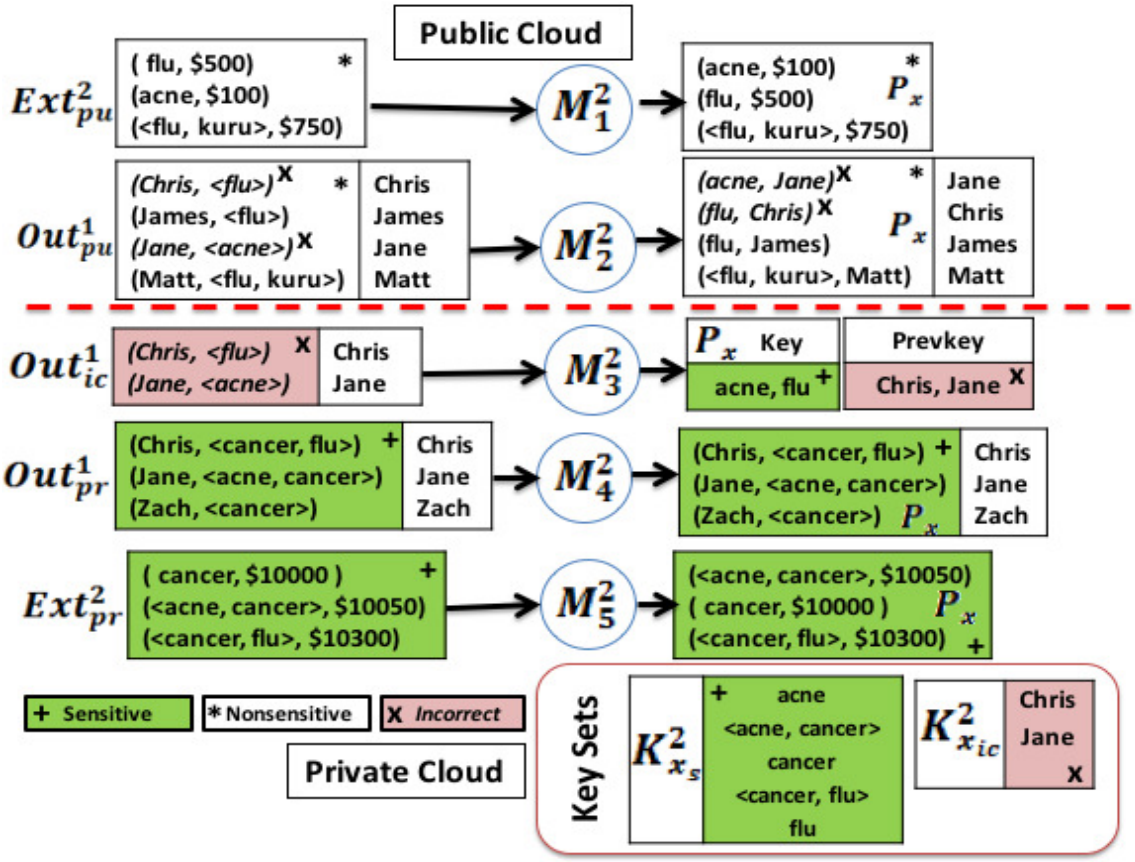

Figure 4.6: SEMROD's Multi-level MR Map Execution

4.4.2.2 Details

Suppose that, the health care provider introduced earlier wants to run another MR job, j_2, that aims to find "how much money each patient paid in total for their exact treatment combinations". For this purpose, he first runs the job j_1 in our previous example, and subsequently executes job j_2 that joins j_1's output with a table containing the exact price of all treatment combinations. These two tables are denoted as Out^1 and Ext^2 respectively in Figure 4.6. In the context of j_2, the intermediate keys will be disease combinations, such as flu or $\langle acne, cancer \rangle$, rather than the names of people.

Out^1 is split into 3 distinct sets of records: Out^1_{pub}, Out^1_{ic} and Out^1_{priv}. Out^1_{pub} refers to the output KV pairs produced by job j_1's public reduce tasks; whereas Out^1_{priv} and Out^1_{ic}

correspond to the correct and incorrect results generated by the private reduce tasks of j_1. Observe that, j_1's reduce tasks outputs, shown in Figure 4.4, only include Out^1_{pub} and Out^1_{priv}, but not Out^1_{ic}. j_1's private reduce tasks are slightly modified to generate the exact replica of j_1's incorrect outputs at the private side. To achieve this, after their initial filtering, they perform an extra reduce operation exclusively over their non-sensitive inputs. For instance, in Figure 4.4, $R^1_{x_{priv}}$ receives two KV pairs with key $Jane$: non-sensitive $(Jane, acne)$ and sensitive $(Jane, cancer)$. $R^1_{x_{priv}}$ reduces $(Jane, acne)$ without the sensitive $(Jane, cancer)$ pair and as a result obtains $(Jane, acne)$ as an incorrect output, which is an exact replica of $Jane$'s related incorrect output within Out^1_{pub}.

Map: As shown in Figure 4.6, SEMROD maps non-sensitive input blocks such as Out^1_{pub} and Ext^1_{pub} on the public cloud, whereas sensitive ones (*i.e.* Out^1_{priv}, Ext^1_{priv} and Out^1_{ic}) are mapped on secure machines. An exception to this is that map results of incorrect replicas (*i.e.* Out^1_{ic}) are never materialized, unless j_2 is a map-only MR job. Similar to single-level MR model, sensitive map tasks M^2_2, M^2_3 and M^2_4 maintain their set of sensitive keys per partition P_x locally. In our example, $K^2_{x_s}$ consists of the intermediate keys: $acne$, $\langle acne, cancer\rangle$, $cancer$, $\langle cancer, flu\rangle$ and flu.

Furthermore, incorrect map tasks such as M^2_3 create a set of incorrect prevkeys per partition P_x and stores them on the local disk. Later, these sets of prevkeys will be merged to construct $K^2_{x_{ic}}$ per P_x by the corresponding reduce task. In Figure 4.6, $K^2_{x_{ic}}$ contains $Jane$ and $Chris$.

Finally, we altered the map output structure such that if a given input record r' has a prevkey k' appended to it, the map task again tags all outputs of r' with prevkey k'. To illustrate, M^2_2 maps $(James, flu) - James$ (triplet) to $(flu, James)$, and in turn, tags this pair with prevkey $James$.

Reduce: Similar to a single-level MR's reduce phase, SEMROD splits any given reduce task R^2_x into $R^2_{x_{pub}}$'s and $R^2_{x_{priv}}$'s. Their execution flows are given in Figure 4.7. $R^2_{x_{pub}}$ reduces

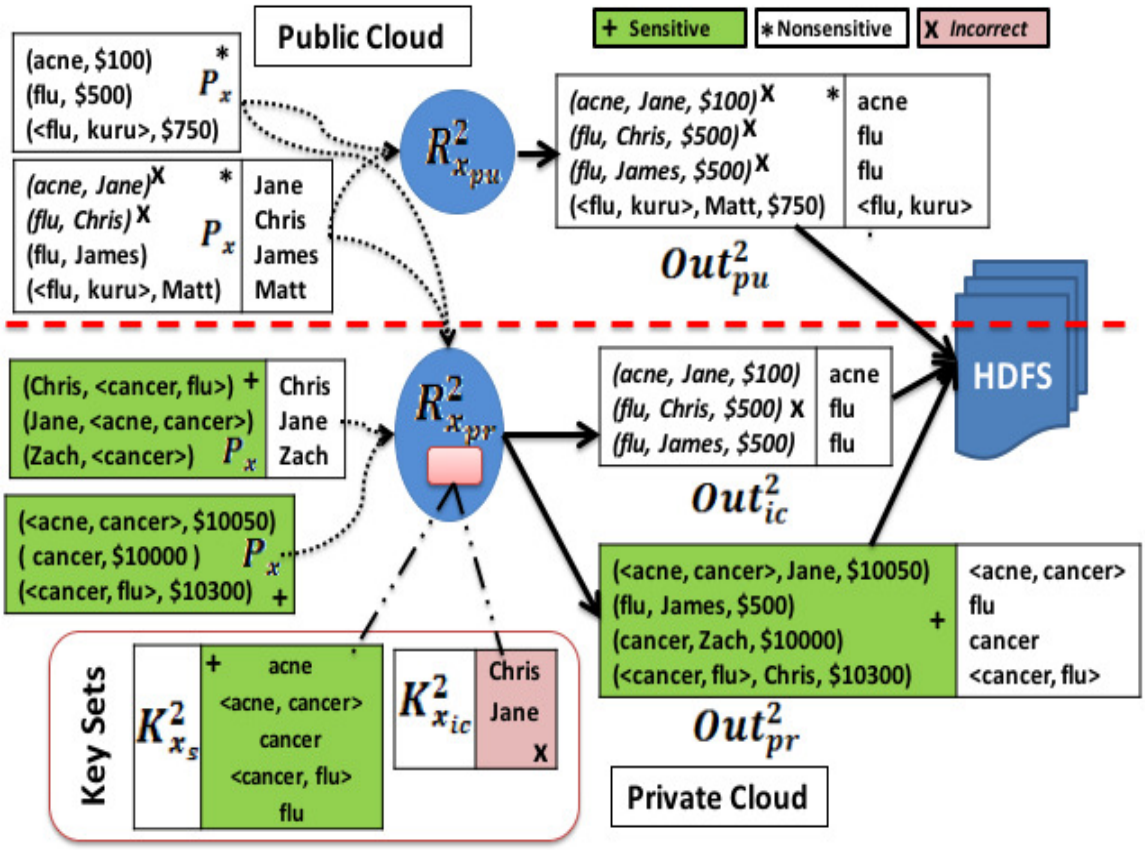

Figure 4.7: SEMROD's Multi-level MR Reduce Execution

all the non-sensitive P_xs from M_1^2 and M_2^2, without any filtering and produces $Out^2_{x_{pub}}$. As a result, the correct outputs for the intermediate P_x keys $\notin K^2_{x_s}$ such as $\langle flu, kuru \rangle$ are maintained on public machines along with some incorrect outputs.

$R^2_{x_{priv}}$ first retrieves the set of sensitive keys and incorrect prevkeys only from the sensitive map tasks P_x outputs, e.g. M_2^2, M_3^2 and M_4^2; and merges them into single set of sensitive keys ($K^2_{x_s}$) and incorrect prevkeys $K^2_{x_{ic}}$ separately. In our example, $K^2_{x_s}$ consists of the intermediate keys: $acne$, $\langle acne, cancer \rangle$, $cancer$, $\langle cancer, flu \rangle$ and flu; whereas $K^2_{x_{ic}}$ contains $Jane$ and $Chris$.

Later, $R^2_{x_{priv}}$ pulls all P_xs from map outputs and eliminates records with a key $\notin K^2_{x_s}$ (*Initial filtering*). After this, $R^2_{x_{priv}}$ is split into 2 sub-reduce tasks if a subsequent job j_3 exists: the

actual $R^2_{x_{priv}}$ and a supplementary $R^2_{x_{ic}}$. After *initial filtering*, $R^2_{x_{priv}}$ aims to create correct reduce results for keys $\in K^2_{x_s}$ (*i.e.* $\{acne, \langle acne, cancer\rangle, cancer, \langle cancer, flu\rangle, flu\}$), and therefore, applies another filtering via $K^2_{x_{ic}}$ to remove the remaining incorrect records and executes the reduce function over the remaining pairs. Note that, the second filtering step is mandatory, since both incorrect $(flu, Chris) - Chris$ and correct $(flu, James) - James$ triplets pass the initial filtering via their key flu. To obtain the correct results for key flu, incorrect triplets must be further eliminated. The reason behind the supplementary $R^2_{x_{ic}}$ is to create an exact incorrect replica of $Out^2_{x_{pub}}$ on the private side. To achieve this, after the *initial filtering*, $R^2_{x_{ic}}$ discards sensitive inputs like $(cancer, Zach) - Zach$ and applies the reduce function R^2 over the remaining non-sensitive inputs. Note that, $R^2_{x_{ic}}$ is not established unless j_3 exists.

4.4.2.3 Correctness of SEMROD

Observe that, the execution before j_2 is *sound* and *complete* in our example. The private side master has full knowledge of what is correct and incorrect within Out^1_{pub}. Thus, the master can intuitively subtract Out^1_{ic} from Out^1_{pub} and obtain the remaining correct portion (sound). Moreover, when we merge correct results from Out^1_{pub} and Out^1_{priv}, we can obtain the complete set of j_1's correct outputs (complete).

We next show that our map scheme will be sound and complete under two assumptions: (1) Jobs are record-level correspondent (defined next), and (2) None of the jobs have defined combiners (except the first one). We first show soundness and completeness under these assumptions. We then slightly adapt the protocol to ensure correctness when combiners are defined or when jobs are not record-level correspondent. Nonetheless, the assumption for record-level correspondence is expected to hold for most of the MR jobs.

Definition 1. *j_{i-1} and j_i is **record-level correspondent**, if $\forall (k, v) \in Out^{i-1}$; (k, v) is a*

single input record $r \in Inp^i$. *To denote this relation,* $j_{i-1} \Rightarrow j_i$.

To express this notion informally, the way Out^1 is interpreted by j_2 is important, j_2 could be such a job that performing a *complete* j_2 map execution would be impossible without conducting an *a priori* elimination of the incorrect portion of Out^1_{pub}. Record-level correspondency dictates that each KV pair within the previous job's input (Out^1) has to be an independent input record for the current job (j_2). If $j_1 \Rightarrow j_2$ does not hold, then the incorrect $(Chris, \angle flu\rangle)$ and the correct $(James, \angle flu\rangle)$ records in Out^1_{pub} can be merged together to create a single input record for map task M^2_2. In that case, the correct $(flu, James)$ KV pair would never appear within M^2's outputs, which is a clear violation of the *completeness* condition.

In addition to $j_1 \Rightarrow j_2$,using traditional combiners in the map phase of j_2 can violate *completeness* condition. Consider that a public map task in an arbitrary j_2 takes numerous incorrect and correct j_1 outputs as input records. Suppose that there exist incorrect record r_{ic} and correct record r_c amongst these input records such that the map outputs $M^2(r_{ic}) = (k_x, 1)$ and $M^2(r_c) = (k_x, 1)$. In other words, when r_{ic} and r_c are mapped, each generates the same KV pair, $(k_x, 1)$. Given our *incorrectness* definition, the first $(k_x, 1)$ is incorrect and the second $(k_x, 1)$ is correct. Now, if the combiner function, that sums up the values, are executed over this public map task's output; incorrect $(k_x, 1)$ and correct $(k_x, 1)$ would be combined to single incorrect $(k_x, 2)$ record in the intermediate data. Contrast this to the execution in Hadoop where all the input records are correct (and, thus, r_{ic} does not exist). In such a case, since the incorrect record $(k_x, 1)$ does not exist, the combiner will result in the intermediate data correct $(k_x, 1)$. Since incorrect $(k_x, 2)$ shows up on intermediate data instead of correct $(k_x, 1)$ in our example, completeness condition would be violated.

Recall that, the master can differentiate incorrect inputs from correct ones in Out^1_{pub} via their appended prevkeys (*Final elimination* in single-level MR) and these set of prevkeys

are already preserved in private map outputs, Out^1_{priv}. Under our assumptions, each non-sensitive map output KV pair (k, v) would be sourced by a single input record r and thereby (k, v)'s prevkey would be the same as r's prevkey. Therefore, the master can first create its own $K^2_{x_{ic}}$ exactly as $R^2_{x_pr}$ does, and later test whether its $K^2_{x_{ic}}$ contains (k, v)'s prevkey to determine its correctness. Thus, our map phase is sound. Under the assumptions that $j_1 \Rightarrow j_2$ and j_2 has no combiners, j_2's map phase is complete as the outputs of M^2_1, M^2_2, M^2_4 and M^2_5 cover all the correct map outputs that would have been generated by the original MR framework.

To track the incorrect data within Out^2_{pub}, records are appended with their prevkeys from K^2, *e.g.* $(acne, Jane, \$100)$ is tagged with the prevkey *acne*. Note that, only P_x keys $\in K^2_{x_s}$ can be partially reduced by $R^2_{x_{pub}}$ to subsequently generate incorrect reduce outputs. Thus, public reduce outputs with prevkey's $\in K^2_{x_s}$ can be identified as incorrect by master, if it creates its own $K^2_{x_s}$ exactly as $R^2_{x_pr}$ does (Soundness). During j_2's reduce phase, $K^2_{x_s}$ exactly corresponds to the set of P_x keys that have been correctly reduced by the actual $R^2_{x_{priv}}$. On the other hand, the P_x keys $\notin K^2_{x_s}$ are correctly reduced by $R^2_{x_{pub}}$, because such keys (*i.e.* $\langle flu, kuru \rangle$) have never been associated with any incorrect or sensitive data. Thus, all intermediate correct KV pairs with such keys must be within the non-sensitive map output. Since $R^2_{x_{pub}}$ retrieves all non-sensitive map outputs and applies the reduction function, its output Out^2_{pub} must contain the correct reduce results for P_x keys $\notin K^2_{x_s}$ (Completeness).

Above, we have shown how job execution is sound and complete for a 2-level MR job. Given the fact that Out^2's characteristics are exactly the same as Out^1's in terms of correctness/incorrectness, by induction, we can conclude that j_2's processing schema provides a *sound* and *complete* execution for the remaining MR jobs left in the chain as long as the following condition is satisfied:

Condition 1: $\forall\ 1 \leq i \leq n,\ j_i \Rightarrow j_{i+1}$ and j_i has no combiners.

Note that SEMROD will correctly process more complex MR workflows, e.g DAG of MR jobs where job j_{i+1} is executed on the outputs of multiple MR jobs; since the incorrect output replicas of previous jobs would already be stored in the private machines upon starting to execute j_{i+1}. As long as 'record level correspondence' is satisfied between the current job j_{i+1} and all the jobs whose outputs are given as an input to j_{i+1}; SEMROD's approach will continue to use both public and private machines during the execution of j_{i+1}.

We next slightly modify the protocol to ensure sound and complete execution when record-level correspondence between jobs does not hold. We will subsequently deal with MR jobs with combiners. In case that, $j_i \not\Rightarrow j_{i+1}$, SEMROD shifts all of j_i's public output to the private side, refines the incorrect records and continues j_{i+1}'s execution solely on private machines. If j_{i+1} has an associated combiner, then SEMROD alters it to a version where incorrect and correct inputs are never merged into a single output. In summary, as we intuitively proved in this section, the following theorem can be stated:

Theorem 1. *SEMROD always offers a sound and complete execution for all types of MR jobs.*

4.4.3 Combiners

In case of single-level MR, combiners, if they exist, can be used without changing their functionality. SEMROD's map phase simply collects the necessary key sets based on combiner outputs rather than raw map outputs. Nevertheless, the way in which combiners work has to be modified in multi-level MR jobs (except the first one, j_1). Combiners can be considered to be local reducers running over local map output data, thereby they share the same functionality as reducers; this notion can be described as: "Take KV pairs with the same key, and then reduce them". In SEMROD, this notion is replaced with: "Take KV pairs with the same key and prevkey, and then reduce them", for combiners running over map outputs.

In this way, we still ensure that all incorrect/correct public map outputs are respectively sourced by incorrect/correct public map inputs in the presence of combiners. Also, prevkey tagging for combiner outputs is feasible, since each public combiner input has to share the same prevkey. Due to these features, SEMROD's multi-level map phase will still be *sound* and *complete*.

4.4.4 Security Analysis of SEMROD

In this section, we provide a formal analysis of the security properties of SEMROD's scheduling scheme on hybrid clouds. In our analysis, we adapt the semantic security definition given in [4] to our framework. Basically, we specify what an honest-but-curious adversary learns by observing the public cloud and we prove that our MapReduce execution does not disclose any additional information.

Definition 2. *Let Sch be a MR task scheduling scheme running on a hybrid cloud infrastructure. For an adversary running algorithm A and simulator S, we define the experiments* $\mathbf{Real}_{A}^{Sch}(\lambda, Inp, M, R, side())$ *and* $\mathbf{Ideal}_{A,S}^{Sch}(\lambda, Inp, M, R, side())$*, where Inp denotes the set of all inputs and side() corresponds to the function used by SEMROD to distribute input data, as follows:*

$\mathbf{Real}_{A}^{Sch}(\lambda, Inp, M, R, side())$ *: $A(1^{\lambda})$ is given all map (M) and reduce (R) tasks, and their outputs after execution on the public side based on the scheduler Sch*[2]*. Eventually, A returns a bit that the game uses as its own output.*

$\mathbf{Ideal}_{A,S}^{Sch}(\lambda, Inp, M, R, side())$ *: $S(Inp_{ns}, M, R, side())$ is again given all M and R tasks, Inp and side() and creates a transcript of the protocol execution to $A(1^{\lambda})$. Note that S is not given any information about the sensitive data in Inp. In addition, it marks all the outputs of public M's and R's as transmitted to the private side and gives this information to $A(1^{\lambda})$.*

[2] *Sch* knows Inp_s.

Eventually, A returns a bit that the game uses as its own output.

We say that Sch is secure against attacks if for all adversaries A, for all Inp and for all M, R, there exists an algorithm S such that $Pr[\mathbf{Real}_{A}^{Sch}(\lambda, Inp, M, R, side()) = 1] - Pr[\mathbf{Ideal}_{A,S}^{Sch}(\lambda, Inp, M, R, side()) = 1] \leq neg(\lambda)$ *for security parameter* λ *and any negligible function* $neg()$*. For consistency with existing literature, we use* λ *to represent the security parameter. For protocols that involve encryption, this parameter is used to force Adversary A to run in polynomial time with respect to security parameter (e.g., encryption key length). Furthermore, the* $neg()$ *function is used to capture the fact that success probability of the attacker is negligibly small with respect to* λ*. In our setting, since we do not use any encryption, we do not need* λ*. Furthermore,* $Pr[\mathbf{Real}_{A}^{Sch}(\lambda, Inp, M, R, side()) = 1] - Pr[\mathbf{Ideal}_{A,S}^{Sch}(\lambda, Inp, M, R, side()) = 1]$ *will be equal to zero as we discuss below.*

Basically, the above definition states that, given a secure task scheduling algorithm *Sch*, any adversary running on the public side will only learn information that can be inferred by running the given MR tasks M and R on all non-sensitive input data. Given the above definition, we can easily prove that SEMROD's scheduling is secure.

Theorem 2. *SEMROD's scheduling protocol given in Section 4.4 satisfies Definition 2.*

Proof Sketch: In our case, for any adversary A, we can use a fixed simulator S. Basically, S will divide Inp_{ns} into two parts using the given $side()$ function. Recall that, in SEMROD, some non-sensitive data can be stored and mapped on the private side. For the subset of Inp_{ns} that is kept on the private side, S runs M and gives its output key-value pairs to A and marks them as transmitted from the private cloud. In addition, S runs M on the subset of Inp_{ns} that is kept on the public side, gives the resulting key-value pairs and inputs to A. Also, S marks them as being computed on the public side. Finally, all the map phase outputs are processed using R, the result of which is also given to A. The output of the reduce tasks are marked as transmitted to the private side. Please note that

the transcript created by simulator S is exactly the same as the transcript seen by A in the real execution for any $(Inp_{ns}, M, R, side())$. Therefore, all the information given to A in both, the ideal and real experiment, will be exactly the same. This in turn implies that A will output the same results in both worlds with the same probability. Therefore, $Pr[\mathbf{Real}_{A}^{Sch}(\lambda, Inp, M, R, side()) = 1] - Pr[\mathbf{Ideal}_{A,S}^{Sch}(\lambda, Inp, M, R, side()) = 1]$ will be equal to zero.

4.5 Implementing SEMROD

The design goals of SEMROD are to uphold the existing MR programming model and to not burden developers with extra programming requirements when they implement a secure MR job. Our new implementation should preserve the way a user interacts with MR and should also enable an end-user to label the sensitive portion of the data before transferring it to the DFS. Also, it should be very convenient to set up the SEMROD environment across the hybrid cloud.

We used the most popular MR implementation, namely Hadoop, to materialize SEMROD. This section describes the details of our modifications on Hadoop.

4.5.1 Environment Setup

We used Hadoop's XML-based files to configure node related properties. To declare that a node is private, a cluster admin is required to add the following properties, *mapred. tasktracker.isPrivate* and *dfs.datanode.isPrivate* with a value *true* to the private nodes' MR (*mapred-site.xml*) and HDFS [58] (*hdfs-site.xml*) configuration files respectively. If these properties are not set or are given as *false* for a particular node, then SEMROD considers that node as a public machine.

4.5.2 HDFS

Hadoop's DFS splits each file into data blocks, by default 64MB in size, and stores each block on multiple machines. All data blocks that belong to a sensitive/incorrect file are labeled as sensitive/incorrect so that the HDFS can copy or replicate these blocks only to private side nodes. In other words, file-level sensitivity is materialized by introducing block sensitivity at the HDFS level.

Partitioning the data to non-sensitive and sensitive files can be automated in SEMROD. For instance, a user could create a metadata file that identifies the sensitive records in the input files and upload it with the original files. Such metadata can be in the form of a predicate or a list of offsets that identifies the sensitive records. While uploading the files, a module embedded in the master (namenode) can iterate over records and split the files into two sub-files based on the provided metadata. Since such a module will be dependent upon the user's sensitivity model, current SEMROD's HDFS assumes that input files are either entirely sensitive or nonsensitive.

SEMROD distinguishes sensitive and incorrect files from others through their file names. If a file name contains the word "sens", then it is considered to be sensitive. Likewise, incorrect files start with the keyword "incorrect". SEMROD"s modified HDFS copies and replicates the files that contain sensitive or incorrect output records to only private nodes, whereas the remaining files are first copied to the public cloud and their replicas are distributed randomly across the hybrid cloud. All the metadata related with sensitive and incorrect files is only stored in the private machines.

4.5.3 Map

While generating the input splits, SEMROD tags each split as sensitive, incorrect or non-sensitive, based on the input file from which they are constructed. After labeling each input split, it creates multiple *TaskInProgress* objects for the map tasks by providing a single *InputSplit* object from the set of input splits. At this stage, the given InputSplit object helps us to tag the *TaskInProgress* objects of map tasks as either sensitive, incorrect or non-sensitive.

Our implementation labels map tasks running on a sensitive and incorrect data block as "private", so that during task scheduling, the JobTracker can schedule them only on secure (private) machines. On the other hand, the remaining map tasks can be scheduled on both, public and private machines. Note that, when a secure node requests a map task from the JobTracker, it gives precedence to private map tasks.

For **Key Set Collection**, we have employed a *HashSet* data structure per partition in every private map task to represent the generated sensitive key or incorrect prevkey sets for each P_x. We use a general *set* structure in order to let the private reduce tasks conveniently compute the unions of key sets received from private map tasks. However, using a *HashSet* representation for key sets could be burdensome, especially for MR jobs emitting a very large number of unique sensitive keys. An alternative extension could be to deploy an approximate and compressed representation such as a *BloomFilter*, which would reduce the amount of memory used within private TaskTrackers as well as the network traffic overhead between the slave nodes. Note that, using a data structure with false positives is acceptable, since such a set would be already containing all the keys-prevkeys that became sensitive or incorrect. Nevertheless, a structure with false negatives would end up not capturing all sensitive or incorrect key-prevkeys within the keyset, and thus could lead to an incomplete job output.

SEMROD can also deploy a centralized approach, rather than a distributed one, for key set

collection purposes. In this centralized approach, once the sensitive map tasks finish their work in SEMROD, they send their sensitive key sets per partition to the master. Master then obtains the overall sensitive key sets per partition P_i by simply computing the union of incoming sensitive P_i key sets. Finally, SEMROD's master passes each overall sensitive key set to the corresponding private reduce task. Note that in this approach, sensitive map tasks do not need to store the key sets on local disks, and in turn may perform better than the distributed approach. However, since it is a centralized architecture, it will not scale well when the number of sensitive keys are high.

4.5.4 Reduce

For each reduce task, Hadoop creates a *TaskInProgress* object. Instead, SEMROD creates two such objects and passes partition ids and their side (public or private) to each of them.

Output Tagging for Tasks: This is done through converting Map and Reduce input/output structures from $(key, value)$ format to $(key, value, prevkey)$ format. For map inputs not in $(key, value, prevkey)$ form (M_i^1's), their $prevkey$ is assumed to be null. Typically, tasks append their output to a *Context* object, which stores the key and value parts of the currently processed KV pair. We modified this class to additionally store prevkey information. Thus, when a map task writes a KV pair to its *Context*, our TaskInputOutputContext class appends the current prevkey to it. For reduces, the class appends the current key instead of the current prevkey.

Finally, to ensure that SEMROD stores all HDFS sensitive output on secure nodes, the word "sens" or "incorrect" is appended to the output filenames accordingly. In case of multi-level MR, SEMROD tags the public reducer's output files as "public" and stores them on public machines.

4.5.5 Fault Tolerance

When a map or reduce task fails due to a bad record or TaskTracker/node failure or takes a much longer time than its siblings, Hadoop reschedules such a task on a different TaskTracker. During this rescheduling, SEMROD's JobTracker follows the same policy as it does for initial scheduling. That is, the JobTracker always reschedules private maps and reducers on private TaskTrackers; whereas public ones can be remigrated to any slave. However, the performance overheads of a private machine/task failure may be higher in SEMROD compared to Hadoop. First, in our architecture, private map and reduce tasks can only run in the private machines and it may take a while to find an available private machine to reschedule the tasks. Second, private reduce tasks shuffle and process more data in SEMROD compared to Hadoop. We will explore these overheads in the experimental section.

4.6 Evaluation

SEMROD so far achieved most of the desired design principles. First, SEMROD is *secure*, since it prevents both *direct* and *key-inference* exposure during storage and MR execution. Second, SEMROD *uses public clouds* as much as possible during the MR execution. Third, it is very *easy-to-use*, since it only requires the users to store sensitive and non-sensitive records in separate files and name them appropriately. Finally, it supports any type of MR execution, so it is *generic*. In this section, we will evaluate whether it satisfies the final design principle, *practicality*. For this purpose, we will provide a formal analysis to compare SEMROD's performance with all other secure solutions. Next, we will compare their performance by conducting extensive experiments.

4.6.1 Formal Evaluation

Before we conduct the experiments to compare SEMROD with other possibly secure solutions (viz. Sedic and *All-Private* in which all computation is handled on private side only), we provide a formal analysis of the computational and network costs of these schemes. We note that this analysis is simplified using several assumptions and its purpose is primarily to provide an intuition to interpret the experimental results discussed next.

Our first assumption is that machines on which MR jobs run are perfectly load balanced. Furthermore, we will assume that shuffling data between the machines within public or private cloud is fully overlapped. Also, we assume that when the reduce tasks shuffles data between the clouds, the entire data shuffling is done over a single link. Such an assumption is rarely true in practice, though it is conservative from the perspective of SEMROD since it favors *All-Private* (our main competitor) the most. Additionally, we assume that there is no overlap between map/shuffle/reduce phases throughout any job execution. Finally, we will assume that all the machines in the hybrid cloud are homogeneous – roughly have the same CPU (and hence computing speeds) and disks (and hence I/O bandwidth) and same LAN speeds.

Let us consider an MR job j_i which consists of a map function M^i and a reduce function R^i. Let D be the size of the input data to the function M^i and, furthermore, let D^* be the size of the intermediate data generated by function M^i and shuffled to the reduce function R^i. D and D^* is usually linearly proportional to each other, since the map functions often output a constant number of KV pairs for each input record.

Now, consider that running the map function on the input data in a single machine takes t amount of time. Then, the map speed of a single machine per unit time, β will be equal to $\frac{D}{t}$ for job j_i. Likewise, suppose that running the entire reduce function on the j_i's intermediate time takes t^* amount of time. Then, β^*, the amount of data that can be reduced by a

single machine per unit time for job j_i will be $\frac{D^*}{t^*}$. Note that slower the machine is or more complicated the given map and reduce functions (e.g. quadratic function) are, lower the β and β^* is. Now given these notations, j_i's execution time on a single machine would be

$$\frac{D}{\beta} + \frac{D^*}{\beta^*}. \tag{4.3}$$

Now let us consider running the same job j_i on our hybrid cloud, in which we have n_{priv} number of private machines and n_{pub} number of public machines. Also, suppose that the network speed between our private and public clouds would be WAN and the network speed between the machines in the same cloud would be LAN. Note that, in a typical hybrid cloud setup, $WAN << LAN$.

If all the load/data is uniformly distributed to each node and map/shuffle/reduce phases are not overlapped with each other; the amount of time that it would take to execute j_i only on the private side of our hybrid cloud (*All-Private*) will be

$$\frac{1}{n_{priv}} \left(\frac{D}{\beta} + \frac{D^*}{\beta^*} \right) + \frac{D^*}{n_{priv} \times LAN}. \tag{4.4}$$

The first term represents the map/reduce time taken by each machine and the second term represents the shuffling time between machines on the private cloud. Note that the equation above considers the shuffling between machines to be fully overlapped – that is, the total shuffling time is equal to a single private machine's shuffling time. The reason behind this is that in LAN networks, in the best case, multiple fast connections can be established at the same time, e.g. while Node A is shuffle data from Node B with LAN speeds, Node C can read data from Node D by again the same LAN speed.

Given that the ratio of sensitive input records are α and the same assumptions, load/data

is uniformly distributed and no overlap between map/shuffle/reduce phases, the same job in Sedic framework, in which the map phase is distributed across the hybrid cloud, but the reduce computation is entirely performed in the private cloud, would take

$$\begin{aligned} &\max\left(\frac{D\times\alpha}{n_{priv}},\frac{D\times(1-\alpha)}{n_{pub}}\right)\frac{1}{\beta}+\\ &\left(\frac{D^*}{n_{priv}\times\beta^*}\right)+\\ &\max\left(\frac{D^*\times(1-\alpha)}{WAN},\frac{D^*\times\alpha}{n_{priv}\times LAN}\right). \end{aligned} \tag{4.5}$$

Again, the first line in the equation above represents the map time taken by each either private or public machine and the second line represents the reduce time spent by each private machine. The third line corresponds to the shuffling time between the clouds, which is assumed to dominate the overall shuffling cost due to WAN being much slower compared to LAN. Note that in Sedic's equation, j_i's shuffling time is computed by dividing the **total** amount of public intermediate data ($D^*\times(1-\alpha)$) with speed of inter-cloud network, WAN. This is a extreme assumption (that favors *All-Private*) but, in the worst case, such a shuffling might be over a single link with WAN speed. Note that we are not trying to be unnecessarily be more conservative to Sedic – which as we explained earlier (and the results will confirm) is not a true competitor to SEMROD, but are trying to favor *All-Private* which we believe is the true alternative to beat when inter-cloud network speeds are very slow.

Now, assuming that no incorrect data will appear during j_i's execution in SEMROD and all the assumptions that we made for Sedic holds, then executing j_i in SEMROD will take

$$\begin{aligned} &\max\left(\frac{D\times\alpha}{n_{priv}},\frac{D\times(1-\alpha)}{n_{pub}}\right)\frac{1}{\beta}+\\ &\max\left(\frac{D^*\times(2\alpha-\alpha^2)}{n_{priv}},\frac{D^*\times(1-\alpha)}{n_{pub}}\right)\frac{1}{\beta^*}+\\ &\max\left(\frac{D^*\times(1-\alpha)}{WAN},\frac{D^*\times\alpha}{n_{priv}\times LAN}\right) \end{aligned} \tag{4.6}$$

amount of time. As before, the first and second line represents the map/reduce time taken by each machine and the third term represents the shuffling time between clouds as in Sedic. Notice that the shuffling time above does not include the public-to-public and private-to-private shuffling time. The reason is that, as we mentioned before, public-to-private data transfer will be the bottleneck during the shuffle phase given that WAN is slow compared to LAN. All other data shufflings (public-to-public and private-to-private), therefore, can be done simultaneously with this slow chain in the execution.

Given the above analysis of performance, the value in Equation 4.6 is always less than the value in Equation 4.5. In other words, given our assumptions, the running time of j_i in SEMROD will always be less than the one in Sedic. However, we cannot say the same thing between Equation 4.4 and Equation 4.6. So, now let us compare these two equations and find out in which cases the SEMROD will perform better than *All-private*. So the condition turns in to

$$\begin{aligned} \frac{1}{n_{priv}}\left(\frac{D}{\beta}+\frac{D^*}{\beta^*}\right) + \frac{D^*}{n_{priv}\times LAN} > \\ \max\left(\frac{D\times\alpha}{n_{priv}},\frac{D\times(1-\alpha)}{n_{pub}}\right)\frac{1}{\beta}+\max\left(\frac{D^*\times(2\alpha-\alpha^2)}{n_{priv}},\frac{D^*\times(1-\alpha)}{n_{pub}}\right)\frac{1}{\beta^*}+ \\ \max\left(\frac{D^*\times(1-\alpha)}{WAN},\frac{D^*\times\alpha}{n_{priv}\times LAN}\right). \end{aligned} \tag{4.7}$$

After bringing similar terms to the same side, replacing WAN with $\frac{LAN}{\rho}$ and n_{pub} with $\lambda \times n_{priv}$, then the same condition will look like

$$\begin{aligned} \frac{D}{\beta}\left(1-\frac{max(\lambda\alpha,1-\alpha)}{\lambda}\right)+\frac{D^*}{\beta^*}\left(1-\frac{max(\lambda(2\alpha-\alpha^2),1-\alpha)}{\lambda}\right) > \\ D^*\left(\frac{n_{priv}(1-\alpha)\rho-1}{LAN}\right). \end{aligned} \tag{4.8}$$

Now let us investigate the factors which will make the above condition easier to satisfy. First, making β or β^* lower will increase the first line of the condition. So, when the machines are slow or the given map/reduce functions are complex, then SEMROD will have more chance to beat *All-private* in terms of the performance. Second, decreasing the ρ value will reduce the value on the condition's second line. In other words, faster WAN compared to LAN is better for SEMROD. Third, increasing the number of private machines without changing the λ value will make the second line larger, so works better for *All-Private*. To investigate the impact of λ and α, we split the Equation 4.8 to three cases based on the *max* function outputs:

1st Case: If $\lambda(2\alpha - \alpha^2) > \lambda\alpha > 1 - \alpha$, (i.e. for larger α values) then the Equation 4.8 will become

$$\frac{D}{\beta} + \frac{D^*}{\beta^*}(1-\alpha) > D^* \left(\frac{n_{priv}\rho - 1}{LAN} \right). \tag{4.9}$$

So, in this case, increasing λ does not affect the comparison, whereas increasing α make the first line smaller, so favors the *All-Private*.

2nd Case: If $\lambda(2\alpha - \alpha^2) > 1 - \alpha > \lambda\alpha$, then the Equation 4.8 will turn into

$$\frac{D}{\beta}\left(1 - \frac{1-\alpha}{\lambda}\right) + \frac{D^*}{\beta^*}(1-\alpha)^2 > D^* \left(\frac{n_{priv}(1-\alpha)\rho - 1}{LAN} \right). \tag{4.10}$$

In this case, increasing the λ makes the first line larger, so favors the SEMROD. However increasing α makes the both lines smaller and in turn, it is not possible to tell whether it is useful or not for SEMROD without knowing the other variables.

3rd Case: If $1 - \alpha > \lambda(2\alpha - \alpha^2) > \lambda\alpha$, (i.e. for smaller α values) then the Equation 4.8 will become

$$\frac{D}{\beta}\left(1-\frac{1-\alpha}{\lambda}\right)+\frac{D^*}{\beta^*}\left(1-\frac{1-\alpha}{\lambda}\right)>D^*\left(\frac{n_{priv}(1-\alpha)\rho-1}{LAN}\right). \tag{4.11}$$

In the $3rd$ case, larger the α or larger the λ means larger the first line and smaller the second line in the condition above, in other words better performance for SEMROD.

In general, when the sensitive data ratio is small, increasing the number of public machines would favor SEMROD compared to *All-Private*. However, if the sensitive data ratio is high, changing λ will not impact the comparison between SEMROD and *All-Private*.

4.6.2 Experimental Evaluation

To the best of our knowledge, Sedic is the only approach that addresses secure data processing using MR in the context of hybrid clouds. By conducting extensive experiments in this section, we compared the performance of our proposed solution against Sedic and *All-Private*: in which all computation is handled on private side only. We run the same experiments on a standard Hadoop setup as well to understand the overhead of SEMROD, Sedic and All-Private bring due to additional security constraints.

4.6.3 Experimental Settings

In this section, we detail our experimental setup, Sedic implementation as well as the various datasets and MR computing jobs we have used in our experiments.

Experimental Setup: We conducted our experiments on a cluster containing 18 nodes, where each node comprises of a Dual-Core AMD Opteron processor with $\approx$ 631GB disk space and 8GB of main memory. These 18 machines are connected to each other with a 1Gbps ethernet network. The average data transfer rate that we measured between any two

machines is approximately 100MB/s. Additionally, each node was configured with at most 2 mappers and reducers. Depending on the private/public node ratio, we simulated a hybrid cloud by labeling some of the nodes as private and the remaining as public. Finally, we modified Hadoop v0.20.2's new *mapreduce* api to implement SEMROD. If a node is in the private cloud, we appropriately changed the configuration files of the SEMROD installation on that node. Similarly, we did these changes in each public node as well.

Sedic: Sedic's privacy-aware scheduling techniques is simulated as follows. In the HDFS layer, Given that each input file can be either sensitive or non-sensitive, we store the sensitive files only in private machines, while maintaining the non-sensitive files at both public and private cloud. In the MapReduce layer, we schedule the map tasks running on sensitive files as well as the reduce tasks to the private machines. We have not implemented Sedic's automatic combiner finding feature. To be fair in our comparison, we have deployed the combiner-free jobs in our experiments. Since Sedic does not provide any technique to separate sensitive job outputs from the non-sensitive outputs, all the job outputs must be declared as sensitive in order to guarantee security. Therefore, in our Sedic implementation. we labeled each job's output as sensitive.

Datasets and MR Computing jobs: In our experimental evaluation, we implemented MR jobs based on TPC-H and HiBench [59] benchmark shown in Table 4.2. Join query names are represented with the uppercase letters of their table names, such as $Part \bowtie PartSupp = P.PS$. Note that our join, median and selection jobs were executed on TPC-H data, whereas the remaining ones are operated on the data generated by HiBench benchmark. The IO characteristics of each job are given in Table 4.3.

The reason we selected these jobs is that they are mostly combiner-free MR jobs; moreover, they cover the typical type of jobs that can be run over an MR framework. Also, they enable us to comprehensively test various performance characteristics of SEMROD *vs.* Sedic *vs.* Hadoop. Therefore, we deployed all sorts of MR jobs, e.g. map/reduce-heavy or single/multi-

level MR jobs.

Single-level Jobs
Median: SELECT l_orderkey, MEDIAN(l_extendedprice) FROM lineitem GROUP BY l_orderkey
Selection: SELECT * FROM lineitem WHERE l_quantity $\geq$ 45
P.PS: SELECT * FROM part JOIN partsupp ON (p_partkey = ps_partkey)
C.O: SELECT * FROM customer JOIN orders ON (c_custkey = o_custkey)
Wordcount, Sort, Terasort
Multi-level Jobs
P.PS.S: SELECT * FROM part JOIN partsupp ON (p_partkey = ps_partkey) JOIN supplier ON (ps_suppkey = s_suppkey)
C.O.L: SELECT * FROM customer JOIN orders ON (c_custkey = o_custkey) JOIN lineitem ON (l_orderkey = o_orderkey)
C.O.L.S: SELECT * FROM customer JOIN orders ON (c_custkey = o_custkey) JOIN lineitem ON (l_orderkey = o_orderkey) JOIN supplier ON (l_suppkey = s_suppkey)
Pagerank (3 Iteration), K-Means (3 Iteration)

Table 4.2: Experimental Jobs

In the context of these experiments, we determined the number of reducers in SEMROD by multiplying the number of nodes by 2, which is in fact equivalent to the number of TaskTrackers. For example, in our hybrid cloud, since all 36 TaskTrackers across the cluster could be exploited during Reduce execution in SEMROD, so the number of Reduce tasks are set as 36 in SEMROD. Conversely in Sedic as well as *All-Private*, the number of reduce tasks are set to the number of tasktrackers in the private cloud. For instance, if there are 3 private nodes in our hybrid cloud, there will be 6 private tasktrackers. So, the number of reduce tasks are set to 6 and we ensured that those 6 reduce tasks are only placed on the private machines.

Job	Input Size	Intermediate Size	Output Size
Median	7416 MB	1118 MB	267 MB
Selection	7416 MB	981 MB	967 MB
P.PS.S	1381/2091 MB	1459/2141 MB	2077/3159 MB
C.O.L.S	1902/11419/23451	2040/12363/24144	4003/23436/31546
Wordcount	3329 MB	4596 MB	125 MB
Pagerank Iter.	4438 MB	7611 MB	85 MB
K-means Iter.	3676 MB	3863 MB	0.0084 MB
Sort	19972 MB	19624 MB	19971 MB
TeraSort	95380 MB	97230 MB	95367 MB

Table 4.3: Job Characteristics

4.6.4 Experimental Results

In this section, we outline the experiments that compare the performance (speed-up) of SEMROD and Sedic *vs. All-Private*. We identified three factors that can impact the performance of SEMROD compared to other secure solutions. Those factors are: private/public node ratio, the amount of sensitive data within the input data and finally the inter-cloud/intra-cloud network speed ratio. To demonstrate the impact of each of these parameters, we comprehensively evaluated SEMROD and Sedic by varying each one of these three criterion, while fixing the other two. Note that each line represents the average speed-up of all the jobs compared to *All-private* in a specific category, such as single or multi-level. Also in most of our experiments, the network speed between the clouds (inter-cloud network) is assumed to be equivalent the network speed within the clouds (intra-cloud network). So, unless stated otherwise, assume that they are equivalent.

Additionally, we plotted the performance results of SEMROD and Sedic *vs. All-Private* equations (given in Section 4.6.1) in single-level MR jobs for each parameter in order to give a sense of what the expected result would look like. D, D^*, β and $\beta*$ values of a single level jobs are computed separately by running that job on a single node of our cluster. In general, the eqaution results of both SEMROD and Sedic is worse compared to actual experimental

results. Speed-up of both SEMROD and Sedic in Equation results is worse than the ones in actual experimental results. Because, our assumption in equations, 'all the intermediate data is shuffled from public to private cloud through a single link', is too conservative and was violated in our experiments.

For each job in our workload, we changed the ratio between sensitive data and input data from 1% to 50%. In TPC-H becnhmark, the sensitive records are randomly selected from the first input table in order to keep the ratio of sensitive keys through out the execution more than the specified sensitivity ratio. In HiBench benchmark, we randomly selected some of the files (varying from 1% to 50%) and labeled them as sensitive.

4.6.4.1 Private/Public Node Ratio ($\frac{priv}{pub}$)

We deployed 4 distinct hybrid cloud scenarios in which $\frac{priv}{pub}$ varied between 1:1 (9 private - 9 public) and 1:17 (1 private - 17 public). We set the input sensitive data ratio to 5% for these experiments. In line with the equation result, the actual results (Figures 4.8 and 4.9) indicate that a lower private/public ratio incurs a longer job execution time in all secure settings. However, SEMROD's performance gain with respect to other approaches increases at a lower $\frac{priv}{pub}$ ratio. For instance, when $\frac{priv}{pub} = \frac{1}{17}$, SEMROD finishes all multi-level jobs on an average 3.7× faster than Sedic and 4.6× faster than *All-Private*, since SEMROD was able to exploit public machines throughout the entire MR execution. In particular, for single-level jobs these numbers become 1.9 and 5.1. Note that, when $\frac{priv}{pub} = \frac{1}{17}$, the expected speed-up of SEMROD compared to All-Private is 10, whereas in actual experiments this number is equal to 5.1. Because, we assumed that the tasks are uniformly distributed across the hybrid cloud in SEMROD equation, however in actual single-level experiments, the reduce load were not able to distributed evenly to those 17 public machines.

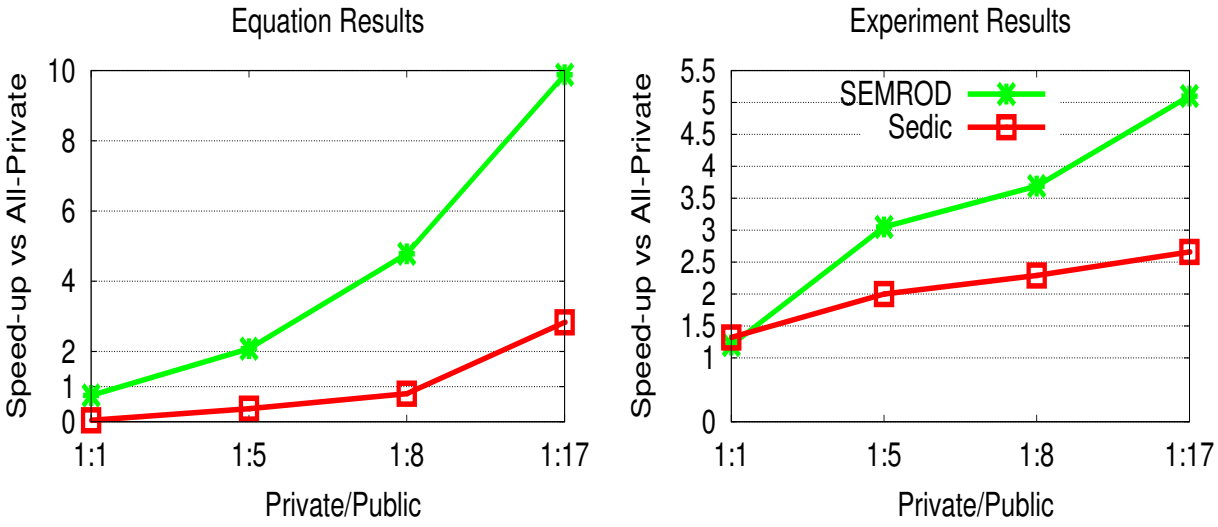

Figure 4.8: Single-level Job Results For Different $\frac{priv}{pub}$ Ratios

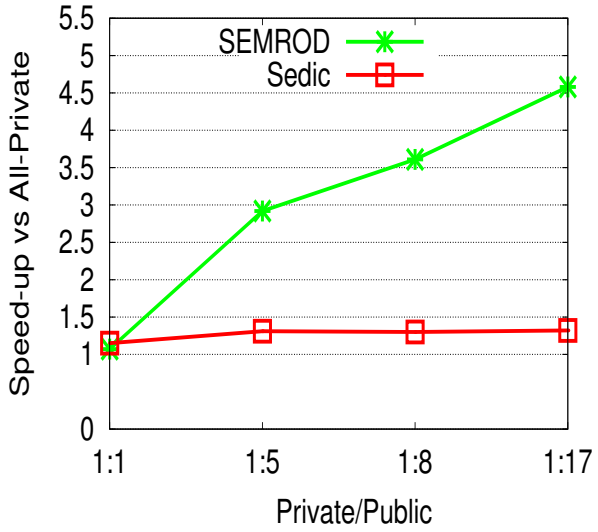

Figure 4.9: Multi-level Job Results For Different $\frac{priv}{pub}$ Ratios

4.6.4.2 Sensitive Data Ratio

For these experiments, we varied the amount of sensitive data $(1, 5, 10, 25, 40, 50\%)$ in the first table of each query. Also, we set $\frac{priv}{pub}$ to $\frac{1}{5}$. As is expected, Figures 4.10 and 4.11 show

that a larger percentage of sensitive data within the input leads to a longer job execution time in both, Sedic and SEMROD, but it does not affect *All-Private* running time. The reason behind this is that a higher ratio of sensitive data results in more computations being performed on the private side. For instance, when the sensitive data ratio is 1%, SEMROD is 2.2× faster than Sedic for multi-level jobs. This number drops to 1.6× for single level MR jobs. Nevertheless, when the ratio of sensitive data ratio is 50%, SEMROD gives the same performance as Sedic in single-level jobs. This is because, at such high ratios, SEMROD's overheads such as key set collection, over-shuffling and final elimination tend to overshadow all its efficiency advantages against Sedic.

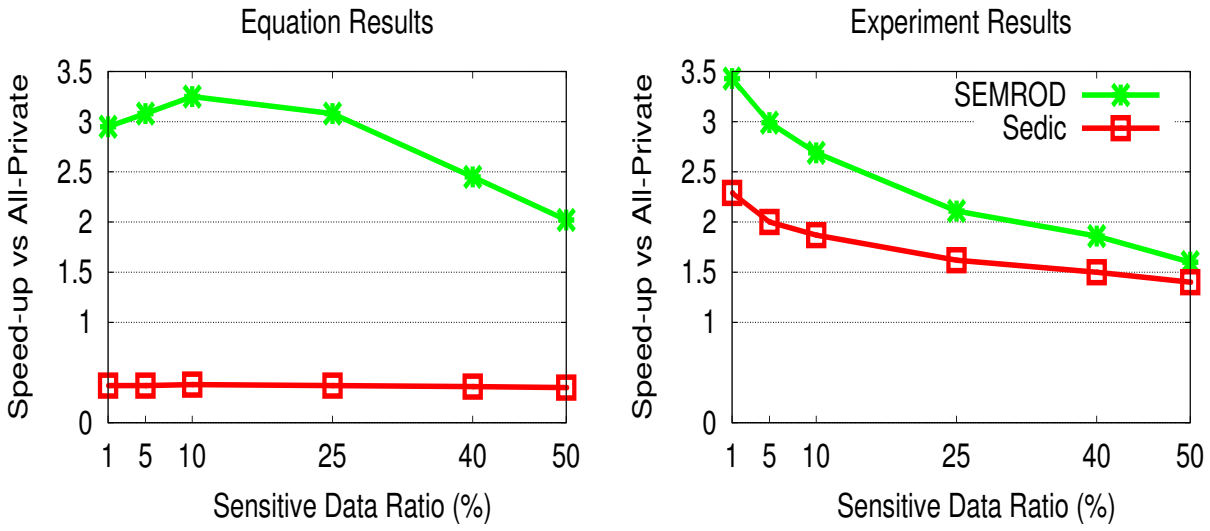

Figure 4.10: Single-level Job Results For Different Sensitivity Ratios

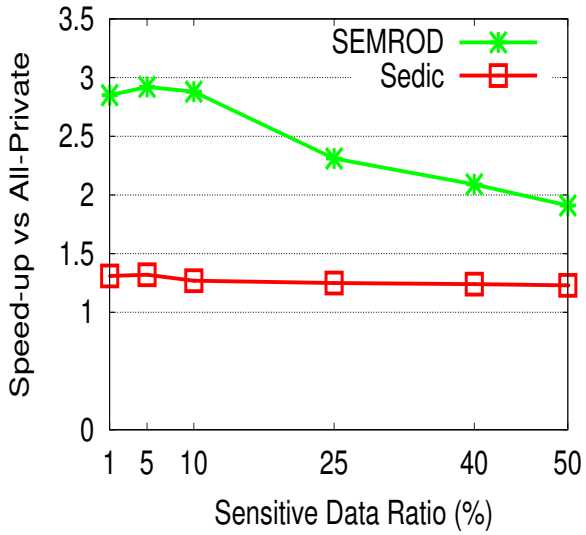

Figure 4.11: Multi-level Job Results For Different Sensitivity Ratios

4.6.4.3 Comparison per Job

In our second experiments, we fixed the $\frac{priv}{pub}$ to 1:5 and the sensitive data ratio to 5% to compare SEMROD, Sedic and Hadoop with *All-Private* for each job. The results are shown in Figure 4.12. In these experiments, SEMROD finished each job by at least 1.3x and at most 4.5x faster than *All-Private*, and at least 1.1x and at most 3x faster than Sedic. SEMROD's performance gain in reduce-heavy single-level MR jobs, such as Median, P.PS and C.O, is higher than the ones in map-heavy jobs like Selection, Sort and Wordcount. Also, note that SEMROD's speed-up ratios for multi-level jobs (except K-Means) are higher than the ones in single-level MR jobs.

The K-means implementation (borrowed from Mahout [60]) is a map heavy multi-level MR job and does not satisfy record-level correspondency condition between the iterations. Thereby, after the first iteration, SEMROD executes the remaining K-means iterations entirely on the private side. In turn, SEMROD's speed up compared to *All-Private* drops to 1.27× from 2.85×, which is the speed up in the K-means' first iteration.

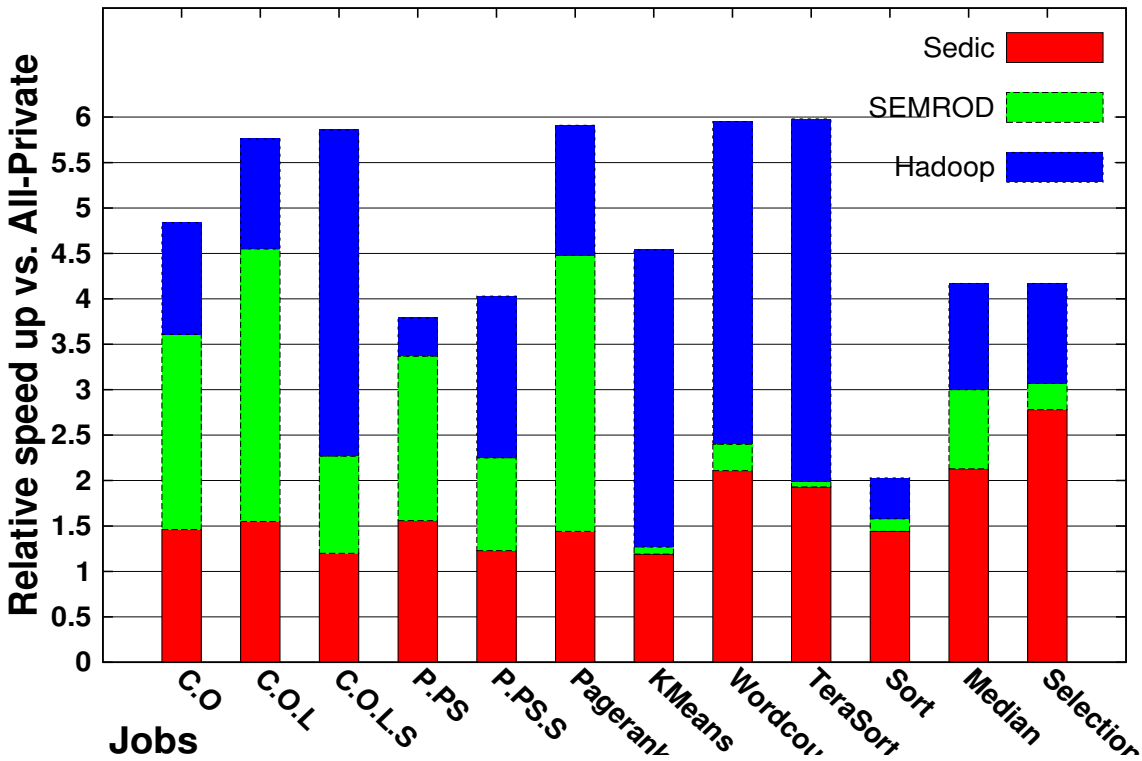

Figure 4.12: SEMROD/Sedic/Hadoop vs All-Private per Job

4.6.4.4 Distributed vs. Centralized KeySet Collection

In this set of experiments, we fixed the $\frac{priv}{pub}$ to 1:5. We first re-run Wordcount and Median jobs by varying the sensitive data ratio on the SEMROD customized with centralized key set collection. The results are provided in Figure 4.13. In most cases, distributed approach results in better execution times. In fact, centralized strategy gives better performance in Wordcount job only when the sensitive files ratio is $\leq 5\%$. Because, after 5%, most of the intermediate keys become sensitive and storing/transmitting such a huge key set becomes a bottleneck within SEMROD's master.

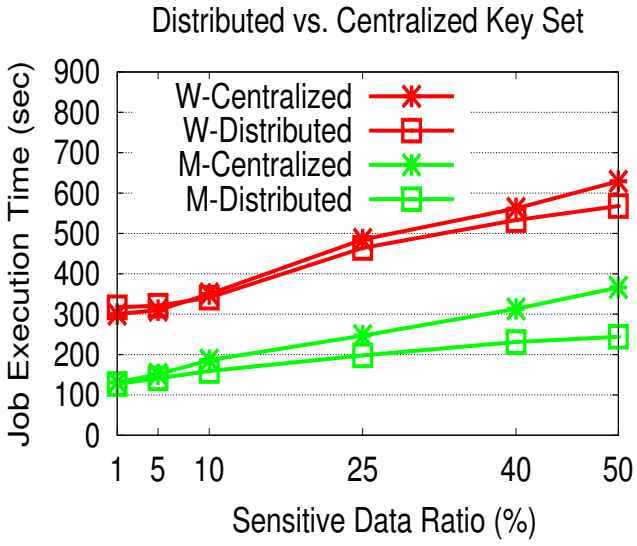

Figure 4.13: Distributed vs Centralized Key Set (M-Median, W-Wordcount)

4.6.4.5 Sensitivity Spread

In this set of experiments, we fixed the sensitive data ratio o 5% and $\frac{priv}{pub}$ to 1:5. We selected $C.O.L.S$ join query to show how the sensitive input data ratio changes in a multi-level MR execution. Figure 4.14 indicates how the ratio of sensitive input records varies along the join query execution. The ratio of sensitive records at the beginning is 5%. As we projected, it is increasing after each job and reaches to 50% before starting to execute the last MR job in the C.O.L.S sequence.

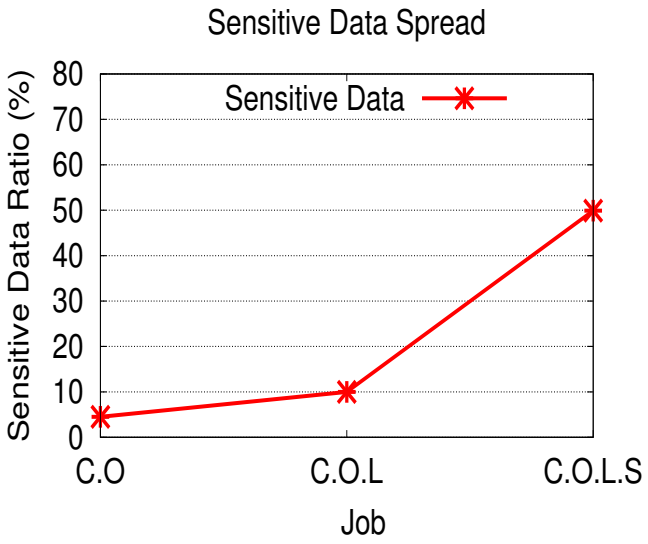

Figure 4.14: Sensitivity Spread for Multi-level Join

4.6.4.6 Non-Secure Hadoop Comparison

When $\frac{priv}{pub}$ is as low as 1:5 and the sensitive data ratio is 1%, SEMROD's overhead in terms of job running time (given at Table 4.4) for single-level queries is at most 48%. For multi-level queries, this number, however, becomes 164%. Also, Table 4.4 displays the SEMROD's overheads on shuffled intermediate and generated output data (before *final elimination*) compared to Hadoop. In most cases, SEMROD shuffles 100% or more data than Hadoop due to the over-shuffling and prevkey tagging. Another observation is that even though the output overhead is low for single level jobs, it increases up to 100% in multi-level jobs. The reason behind this sudden increase is that, in the final jobs of $P.PS.S$ and $C.O.L.S$, the ratio between sensitive keys and all keys ($\frac{K_s^i}{K^i}$) is as high as $\approx$ 90%. The more $\frac{K_s^i}{K^i}$ is, the more incorrect data is generated in both public and private reducers. Thus, shifting all the output and running the entire subsequent computation on the private side would be a more efficient option. Under this scenario, SEMROD will exactly follow this strategy, since the subsequent job's input would be violating the *Condition 2*.

	Running Times (sec)		Overheads (%)		
Job	Hadoop	SEMROD	Time	Shuffle Size	Output Size
Median	81	108	33%	140%	47%
Selection	64	83	30%	105%	8%
P.PS	48	56	17%	111%	3%
P.PS.S	97	167	72%	107%	82%
C.O	56	78	39%	109%	3%
C.O.L	259	354	37%	108%	2%
C.O.L.S	568	1499	164%	105%	100%
Wordcount	131	329	151%	177%	97%
Pagerank Iter.	285	406	42%	66%	44%
K-means Iter.	112	160	43%	114%	40%
Sort	176	258	47%	102%	98%
TeraSort	1074	1588	48%	107%	97%

Table 4.4: Hadoop vs SEMROD

4.6.4.7 Slow Inter-Cloud Network

In this experiment, we study the impact of inter-cloud network speeds on the performance of Sedic and SEMROD. Since they shuffle data between public and private machines, possibly over wide area network (WAN), their performance will be negatively impacted due to the slow inter-cloud network speed[3]. At some stage, if WAN speeds are significantly slow, the improvements due to using public resources may be mitigated by the slow networks

The inter-cloud traffic speed is slowed by a factor of ρ compared to speeds between machines within the same cluster. This is achieved by stopping a thread shuffling data from the other cloud for a period of time $(\rho - 1) * X$ where X is the actual time taken to shuffle the data. This effect simulates a slowdown of inter-cloud network speed by a factor of ρ compared to the intra-cloud network speed.

Varying ρ from 1 to 100 simulates a wide range of situation where inter-cloud network speeds

[3]Amazon offers Amazon Direct Connect, where customers can establish a dedicated network between their private data center and the closest AWS region. It offers speeds up to 10Gbps and enables customers to have multiple connections.

are the same as that between local machines to a situation where such speeds are 100 times slower compared to intra-cloud network speeds. Our experiments are conducted under the sensitive data ratio of 5% and two different $\frac{priv}{pub}$ ratios: $\frac{1}{17}$ and $\frac{1}{5}$. Finally, we measured the average speed-up ratios of SEMROD and Sedic vs *All-Private* for single-level and multi-level MR jobs separately.

The results, given in Figures 4.15 and 4.16, clearly indicate that SEMROD performs better than Sedic and *All-Private* for both single-level and multi-level MR jobs, even though the inter-cloud network is 100 times slower (1MBps) than the existing one and the $\frac{priv}{pub}$ is as high as $\frac{1}{5}$. Note that, Sedic's execution time for multi-level MR jobs is longer than *All-Private* when $\rho = 100$ and $\frac{priv}{pub} = \frac{1}{5}$. But under the same conditions, SEMROD is still 1.5x faster than both Sedic and *All-private*. Also, when $\rho >= 5$, both SEMROD's and Sedic's actual speed-up for single-level MR jobs is much more than the expected speed-ups. The reason behind this is that our conservative assumption in the Equations ('everything is shuffled through a single link between the clouds') is not applied within our slowed network simulations.

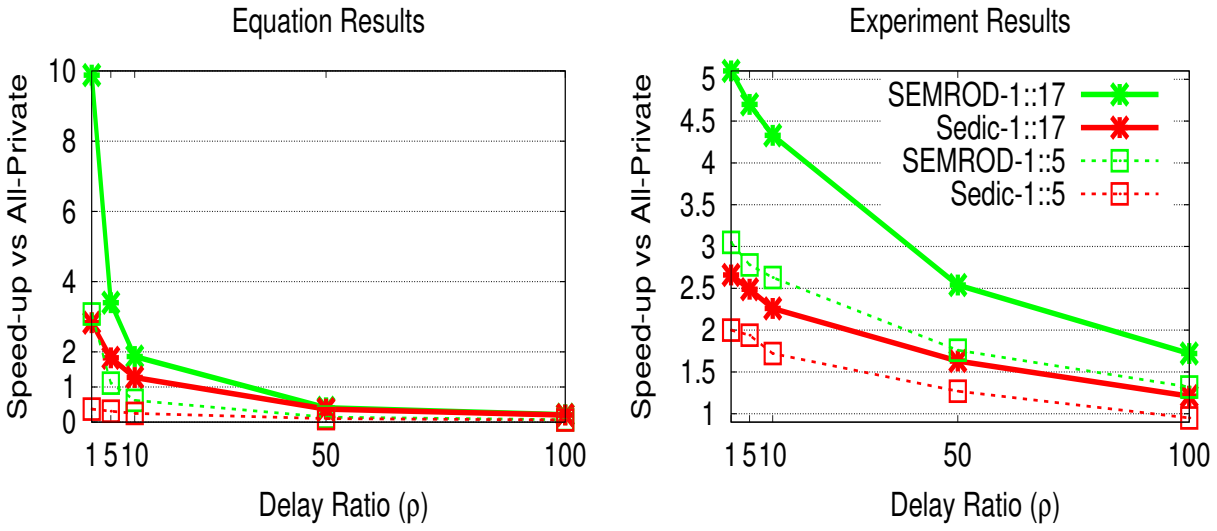

Figure 4.15: Single-level Job Results with Different ρ Values

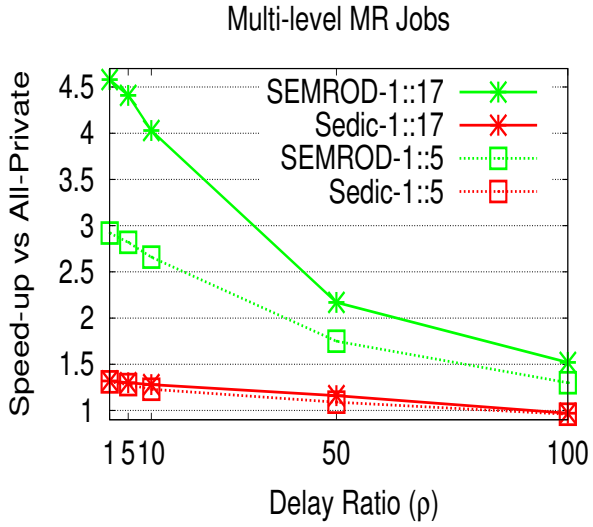

Figure 4.16: Multi-level Job Results with Different ρ Values

4.6.4.8 Experiments with Real Hybrid Cloud

For this set of experiments, we reserved 17 nodes from a remote computing center and merged it with a single machine from our cluster in order to create a geographically distributed hybrid cluster over the wide area network. Each node we reserved has Intel EM64T Xeon E5 2.6GHz 16 core cpu, 64GB Memory and 280GB disk. We installed SEMROD, Sedic and *All-Private* systems on this new 18 node hybrid cluster. We labeled these nodes as public and the single machine from our cluster as private. Note that, in this hybrid cluster, $\frac{priv}{pub}$ as $\frac{1}{17}$. We measured the average data transfer speed over the inter-cloud network as 30MBps and we ensured that the connection between the public and private machines is established over the wide-area network, not over a dedicated network.

We executed all the jobs in our testbed using SEMROD, Sedic and *All-Private* over this new hybrid cloud. The speed-up ratios of SEMROD and Sedic vs. *All-Private* per job is given at Figure 4.17. As it is seen in Figure 4.17, SEMROD performs at least 2x faster than Sedic and *All-Private* for most of the jobs. Note that Sedic's performance for MR jobs that

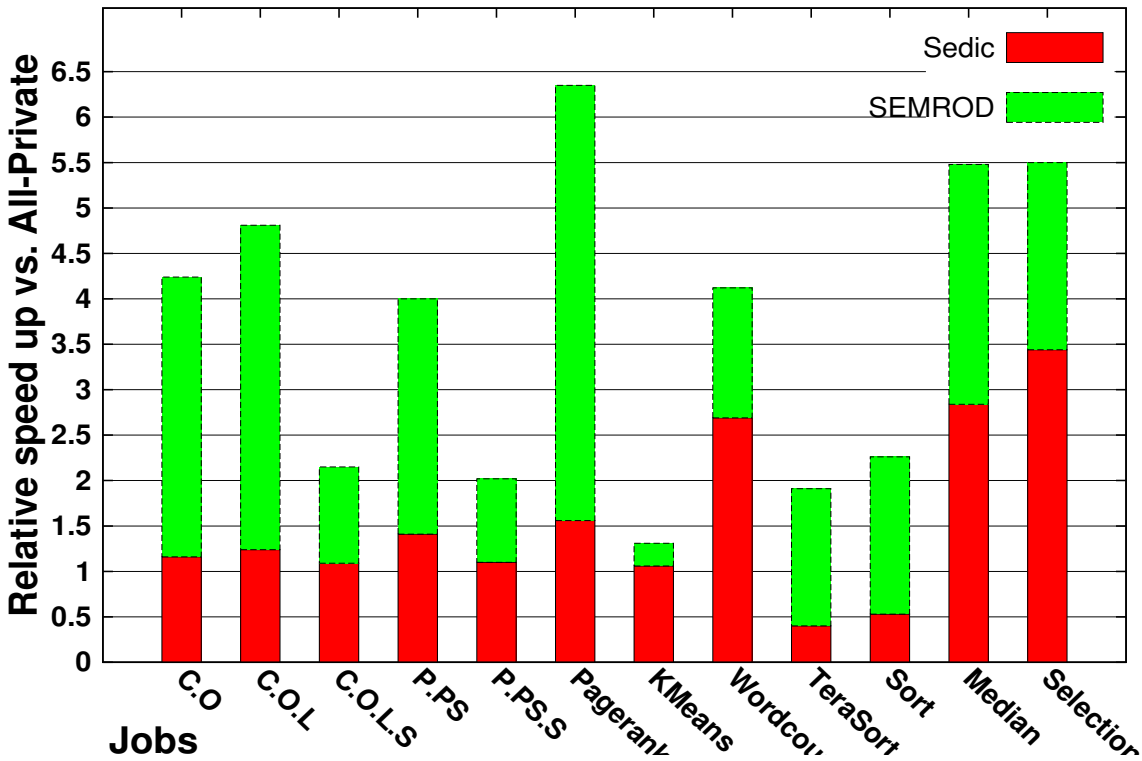

Figure 4.17: SEMROD/Sedic vs All-Private in Real Hybrid Cloud

shuffles a lot of data over the inter-cloud network (e.g. Sort and Terasort) is almost 2x worse than *All-Private*. In contrast to Sedic, SEMROD is approximately 2x faster than *All-Private* for the Sort and Terasort jobs. To sum up, the results in Figure 4.17 clearly demonstrate that SEMROD is practical to use in a hybrid cloud, even when the inter-cloud connection is established over wide area networks.

4.6.4.9 Fault-tolerance

This set of experiments are conducted to investigate the impact of a machine failure to SEMROD, Sedic and Hadoop's performance. We intentionally killed 1 private tasktracker or 3 public tasktrackers in the middle of map and reduce phases. We only executed single-level MR jobs and measured the average time overheads of these failures on job running times. In

these experiments, sensitive data ratio, $\frac{priv}{pub}$, ρ is set to 5%, $\frac{1}{5}$ and 10 respectively. The first 3 columns in Table 4.5 show the results for 1 private machine failure case, whereas the last 3 columns indicate the overheads when we killed 3 public machines. Observe that private machine failures generally harm Sedic the most and Hadoop the least in terms of performance. On the other hand, in case of public machine failures, SEMROD's and Hadoop's overheads are usually highest, and Sedic's overheads are lowest. This is because, Sedic uses the private machines heavily during an MR job execution in contrast to Hadoop and SEMROD, in which public machines are the ones doing the more computation.

	1 Private Overhead (sec)			3 Public Overhead (sec)		
Job	SEMROD	Sedic	Hadoop	SEMROD	Sedic	Hadoop
C.O	45	65	34	49	120	95
P.PS	53	72	24	96	77	94
Median	43	60	16	170	0	199
Selection	49	29	9	166	71	107
Wordcount	311	167	55	196	0	197
Sort	101	395	48	255	163	128

Table 4.5: Machine Failure Overheads

4.7 Conclusions and Extensions

In this chapter, we presented SEMROD framework to efficiently address security challenges in building an MR framework over hybrid clouds. SEMROD minimally modifies the Hadoop MR implementation by shuffling all public map outputs to both, public and private reducers. These modifications allow exploitation of public resources for both, map and reduce phases, of an MR job without compromising security. Our results show that SEMROD achieves security by paying small performance overhead as compared to Hadoop.

Furthermore, when inter-cloud networks are as fast as intra-cloud networks, SEMROD dramatically improves the average job execution time by as much as 3.5× as compared to

alternate secure MR frameworks.

While SEMROD provides an effective framework for secure MR execution that ensures no sensitive information is leaked to public clouds, several extensions are possible to make it more efficient.

An extension would be to explore a (possibly cost-based) approach to decide when the extra shuffling and filtering cost of SEMROD are compensated by the advantage of using additional computation power of the public machines. In SEMROD, the ratio of sensitive/incorrect keys increases across each phase of the multi-level MR pipeline. It is conceivable that at some stage in the execution of a SEMROD pipeline, shifting the entire computation to the private side might be a better plan compared to paying the overheads. While we did not observe such a phenomena in our experiments, at the level of sensitive data and the type of jobs we tested, such an issue can arise. This is specially the case when record-level correspondence for multi-level MR jobs does not hold as in Mahout's K-means implementation where the entire output of each iteration is input to each map task in subsequent iteration.

Finally, developing secure and effective approaches to deal with multi-level jobs when record-level correspondence may not hold would be an interesting extension.

Chapter 5

Secure and Efficient Query Processing over Hybrid Clouds

5.1 Introduction

Our goal in this chapter is to develop an efficient approach to executing SQL style queries securely by appropriately partitioning computation and data between the private and public machines. Our prior work, SEMROD, has explored secure map-reduce (MR) implementation in hybrid clouds using portioning of data and computation that supports semantic security. One approach to implementing secure SQL style queries is to transform them into corresponding MR jobs and use SEMROD. Such an approach result in significant overhead. SQL offers much higher levels of semantics that can be exploited to significantly improve performance without sacrificing semantic security.

Distributing query execution in a secure and efficient way consists of two interrelated problems:

- Data distribution problem: How should data be distributed between private and public clouds?

- Query execution problem: Given the sensitive data can only be stored on the private cloud, how do we execute the query securely and efficiently across the hybrid cloud?

The solutions to these problems will differ based on how the data is placed across the hybrid cloud at the beginning, which in turn depends upon the usage scenario of the hybrid clouds. Hybrid clouds are an emerging technology with a large amount of commercial interest, [61], and they are being used for variety of purposes. While running actual business applications in the private cloud, organization uses public clouds for archiving, testing or providing standard applications, such as mail or calendar, to their customers or employees. None of these scenarios necessitates utilizing both clouds to do heavy data processing. An example of a use case requiring data processing at both private and public cloud is cloudbursting. Recently, organizations, such as Web/e-commerce, typically have dynamic and unpredictable requirements, which can be difficult to plan for when hosting in their on-premise data center. In order to meet unpredictable demand, they rent resources from a public cloud and merge it with their existing resources (*cloudbursting*). As another use case, companies are exploring to outsource their data center functionalities in order to avoid building a new data center and all of the associated costs and compliance requirements. So, they use public clouds as a cost efficient option to meet the needs for more capacity without incurring additional capital costs (*outsourcing*) . In the cloud bursting scenario, typically the data is first flows into the private cloud through some higher level applications and then stored in there. Whenever a need arise for the public cloud, certain portion of the data is replicated to the public cloud. In such scenario, private cloud has enough resources to store the entire dataset, but is limited in terms computational power. In outsourcing scenario, usually the private cloud is limited in terms of both storage and computation. Therefore, the data is mostly initiated and stored in the public cloud and whenever a mission critical task need to be run on the

private cloud, the corresponding portion of the public cloud data is migrated to the private cloud. In the context of this chapter, we will assume that the entire data is initially stored in the private cloud as in the cloud bursting scenario, though our technical development does not fully require that the entire data must stored on the private cloud. For instance, non-sensitive data could reside on the public cloud initially and it could be shifted to private cloud whenever the need arises.

Given our initial data placement assumption, we solved the data distribution problem intuitively. To ensure security, the sensitive data is only stored on the private cloud. Non-sensitive data migrates to the public cloud, though we let a copy of it remain on the private cloud since part of non-sensitive data may be needed to efficiently partition queries between the private and public nodes. For instance, a join query may necessitate that non-sensitive data be available at the private node in case sensitive records from one relation may join with non-sensitive records in another. Since we cannot migrate sensitive data to public cloud, if we do not store non-sensitive data on the private machines, we will need to transfer such data during query execution, which would incur high transfer costs. Moreover, as will become clear, our approach is designed to support an expected class of queries (i.e., a workload) very efficiently by exploiting both public and private machines. If a query deviates from the expected class, it must run on private machines requiring all data to be locally present (or transferred dynamically during query execution which will incur significant overhead).

Given such a initial data distribution, our focus in this chapter is to minimize the running time of a given query, Q, subject to the security constraint that sensitive records / information are never leaked to the public cloud. The strategy we use is to split the given query Q running on dataset D as

$$Q(D) = Q_{merge}\Big(Q_{priv}(D_{priv}), Q_{pub}(D_{pub})\Big), \tag{5.1}$$

where Q_{priv} and Q_{pub} are private and public cloud sub-queries respectively. Q_{priv} is executed on the private subset of D, D_{priv}; whereas Q_pub is performed over the public subset of D, D_{pub}. Q_{merge} is a private cloud merge sub-query that reads the outputs of former two sub-queries as input and creates the outputs equivalent to the that of original Q. We call such an execution strategy as *split-strategy*. Two aspects of *split-strategy* are noteworthy:

- *Split-strategy* offers semantic security. To see this, note that the public machines only have access to D_{pub} that does not contain any sensitive data. Moreover, no information is exchanged between private and public machines during the execution of Q_{pub} and, as a result, the execution at the public machines is observationally equivalent to the situation where D_{priv} could be any random data.

- *Split-strategy* gains efficiency by executing Q_{priv} and Q_{pub} in parallel at the private and public cloud respectively and, furthermore, by performing inter-cloud data transfer at most once throughout the query execution. Note that the networks between private and public clouds can be significantly slower compared to the networks used within clouds. Thus, minimizing the amount of data shuffling between the clouds will have a huge performance impact.

Suppose that D_{priv} is equivalent to the original dataset D and, but D_{pub} is equal to the non-sensitive records in D. Creating an efficient *split-strategy* for queries doing linear transformations such as selection or projection only queries is straightforward. That is, Q_{priv} will be equivalent to the original Q, but will be performed only over the sensitive records in D_{priv}. Likewise, $Q_{pub} = Q$, but D_{pub} is equal to the non-sensitive records in D. Finally, $Q_{merge} = \cup$. In the context of a join query, $R \underset{C}{\bowtie} S$ where C is the join condition between relation R and S, an efficient *split-strategy* is however not trivial. Assume that only very small portions of R and S are sensitive, denoted as R_s and S_s, and remaining large fraction of them are non-sensitive, denoted as R_{ns} and S_{ns}. The naive *split-strategy* for $R \underset{C}{\bowtie} S$ would

be (1) $Q_{pub} = R_{ns} \underset{C}{\bowtie} S_{ns}$, (2) $Q_{priv} = R_s \underset{C}{\bowtie} S_s \cup R_s \underset{C}{\bowtie} S_{ns} \cup R_{ns} \underset{C}{\bowtie} S_s$. The scan/join cost of Q_{priv} in this *split-strategy* is close to the scan/join cost of the original query $R \underset{C}{\bowtie} S$.

To be able to run join queries, or in general, complex SPJ queries efficiently, a *split-strategy* requires us to identify the non-sensitive records that joins with the sensitive records from another relation. We can then limit the scan and join to those subset of records. If the size of that subset is small, one can have a much more efficient *split-strategy*.

Our primary contributions in this chapter are summarized below:

- We propose a fully secure and efficient approach to do query processing using the hybrid clouds.
- We present a set of formal split rules in order to create a secure and efficient *split-strategy* for SQL queries in the hybrid cloud model.
- We create a special attribute on the private cloud data in order to obtain an efficient *split-strategy* for queries involving join operations. Additionally, we physically partition the private cloud data based on this index to further reduce the query execution times in *split-strategy*.
- We design an efficient algorithm to construct our new attribute either based on the dataset schema or based on an expected workload, which is provided by the user.
- We carry out extensive evaluation and testing on realistic datasets using Hadoop and Spark [62] in order to validate the benefits of *split-strategy* using our new column.

The rest of the chapter is organized as follows: Section 5.2 provides an overview of our approach. In Section 5.3, we present our algebraic approach to construct an secure and efficient *split-strategy* for single block queries. Section 5.4 presents the implementation details of our techniques over Spark, Hadoop and Hive [63]. Section 5.5 provides the results of the

experimental evaluation of our strategies using the TPC-H benchmark. Section 5.6 reviews the related work in the area of distributed data processing. In Section 5.7, we discuss how to create a *split-strategy* for more complicated queries, such as cyclic or nested queries.

5.2 Overview of Our Approach

In this section, we will explain the challenges that we face in splitting queries across hybrid clouds and then provide the overview of our approach to address these challenges. For this purpose, we will use the example join query discussed in Section 6.1, i.e. $R \underset{C}{\bowtie} S$. Let R and S be as shown in Figure 5.1 and let C be equal to $R.Region = S.Region$. Recall that, the entire R and S is stored on the private cloud and only non-sensitive partition of them, R_{ns} and S_{ns} are stored on the public cloud. Given such a data distribution,the scan and join cost of running the entire $R \underset{C}{\bowtie} S$ on the private cloud is equal to 6. However, when $R \underset{C}{\bowtie} S$ is split using our naive *split-strategy*, the scan and join cost of the private side query will be equal to 8. In fact, when this naive approach is used to split a query that joins more than two tables. For instance when $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$, where C' is $S.Region = T.Region$, is split, the scan/join cost of the private side computation may be much higher compared to running the original query on private machines.

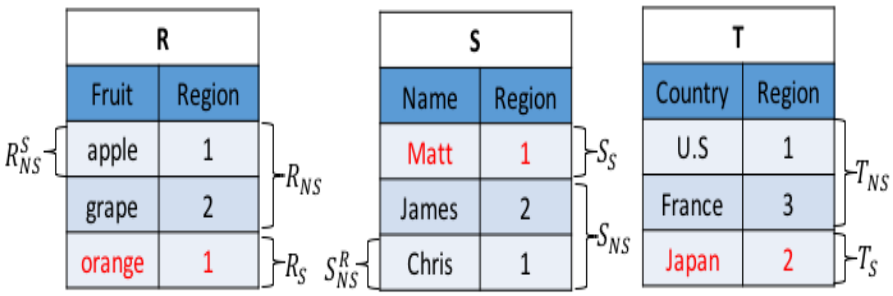

Figure 5.1: Example Relations

The cost of $R \underset{C}{\bowtie} S$ on the private machines can be significantly reduce by doing pre-filtering

relations R and S based on the sensitive records of the other relation. To perform such a pre-filtering operation, the records in R_{ns} and S_{ns} have to be co-partitioned based on whether they join with a sensitive records from the other table under condition C.

Let R^S_{ns} be the set of non-sensitive R records that joins with any sensitive record in S (i.e a record from S_s). In our case, $R^S_{ns} = (apple, 1)$. Similarly, let S^R_{ns} be the non-sensitive S records that joins with any record from R_s, i.e. (Chris, 1). In that case, the new private side computation can be rewritten as

$$(R_s \cup R^S_{ns}) \underset{C}{\bowtie} (S_s \cup S^R_{ns}). \tag{5.2}$$

Since the number of sensitive records are low (equal to 1) in both R and S, the sizes of S^R_{ns} and R^S_{ns} are expected to be low as well [1] Thus, the scan and join cost of this new private plan will be equal to 4 and will be less than the earlier version, which was 6. This strategy, nonetheless, introduces a new challenge. Since $R^S_{ns} \underset{C}{\bowtie} S^R_{ns}$ is both repeated at public and private cloud, the output of $R^S_{ns} \underset{C}{\bowtie} S^R_{ns}$, (apple, Chris, 1), will be computed on both private and public machines. To prevent this, we do a guarded join ($\bowtie'$) on the private side, which discards the output, if it is generated via joining two non-sensitive tuples. This feature can easily be implemented by adding a column to R and S that marks the sensitivity status of a tuple, whether it is sensitive or non-sensitive, and then by adding a appropriate selection after the join operation. In other words, the complete representation of private side computation for $R \underset{C}{\bowtie} S$ would be

$$\sigma_{R.sens \vee S.sens = true}((R_s \cup R^S_{ns}) \underset{C}{\bowtie} (S_s \cup S^R_{ns})) \tag{5.3}$$

[1] For instance, let R be *Customer* table and S be *Orders* table in the TPC-H dataset. If 1% of the R (Customer) records is sensitive, then the ratio between S^R_{ns} and S (Orders) will be at most 1%.

where *sens* is a boolean column (or partition id) appended to relations R and S on the private cloud. Assume that it is set to true for sensitive records and false for non-sensitive records.

There exist multiple implementation challenges in this new approach. First challenge is about the cost of creating R^S_{ns} and S^R_{ns} beforehand. Extracting these partitions for a query might take as much time as executing the original query. However, if it's prepared once, whenever R relation join with S based on condition C, the new efficient private plan can be deployed.

The second challenge is about co-partitioning tables for more complex queries. For instance, in case of a query $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$, the plan would be to first compute the results of $R \underset{C}{\bowtie} S$, and then to join them with T. However, if we do the private side computation of $R \underset{C}{\bowtie} S$, based on Equation 5.2 (no duplicate filtering) and join the results with T, then we will not be able to obtain the complete set of sensitive $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$ results. To see this, consider the sensitive record (Japan, 2) in T that joins with non-sensitive (grape, 2) tuple in $R - R^S_{ns}$ or join with non-sensitive (James,2) tuple from $S - S^R_{ns}$. Thus, the non-sensitive records of R and S has to be co-partitioned based on the sensitive record of T via their join paths from T. In $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$, the join path from T to R is $T \underset{C'}{\bowtie} S \underset{C}{\bowtie} R$ and from T to S is $T \underset{C'}{\bowtie} S$. Similarly, the non-sensitive T records has to be co-partitioned based on the sensitive R and S records via join paths specified in the query.

Final challenge is about maintaining these co-partitions and feeding the right one when an arbitrary query arrives. Given a workload of queries and multiple possible join paths between any two relations, each relation R in the dataset may need to be co-partitioned multiple times. This implies that any non-sensitive record r of R might appear in more than one co-partition of R. So, maintaining each co-partition separately might be unfeasible in terms of storage. However, the identifiers of each co-partition that record r belongs to can be embedded into r as a new column. We call such a column as the *co-partition* column or

the CPT column.

Note that this column will be set to null for sensitive tuples stored in the private side, since the co-partitions are only for non-sensitive tuples. Thus, it can further be used to serve another purpose, indicating the sensitivity status of a tuple r by setting it to "sens" only for sensitive tuples. Note that *foreign key* columns in the tables does not serve our purpose, because such columns never reveals the information of whether a record joins with a sensitive or non-sensitive tuple from the other table.

To formalize the concept of co-partitioning, we first need to define the notion of join path. let R_i be a relation in our dataset D and let Q be a query containing relation R_i. We say a join path exists from relation R_j to R_i, if either R_i is joined with R_j directly based on a condition C, i.e. $R_j \underset{C}{\bowtie} R_i$, or else indirectly using other relations in Q. A join path p can be represented as a sequence of relations and conditions laying between R_j and R_i relations. Let $PathSet$ be the set of all join paths that is extracted either from the expected workload or dataset schema a given and let

$$PathSet_i = \{\forall p \in PathSet : \text{path } p \text{ ends at relation } R_i\}. \tag{5.4}$$

Let $CP(R_i, p)$ be the set of non-sensitive R_i records that will be joined with at least one sensitive record from any other relation R_j via the join path p. Note that p starts from R_j and ends at R_i and can be given as an id to $CP(R_i, p)$. Any $CP(R_i, p)$ is called as "co-partition" of R_i. Given these definitions, the CPT column of a R_i record, r can be defined as :

$$r.CPT = \begin{cases} sens & \text{if } r \text{ is sensitive} \\ \{\forall p \in PathSet_i : r \in CP(R_i, p)\} & \text{otherwise} \end{cases} \tag{5.5}$$

R		
Fruit	Region	CPT
apple	1	$S \bowtie R$
grape	2	T$\bowtie S \bowtie R$
orange	1	sens

S		
Name	Region	CPT
Matt	1	sens
James	2	$T \bowtie S$
Chris	1	$R \bowtie S$

T		
Country	Region	CPT
U.S	1	$S \bowtie T$, $R \bowtie S \bowtie T$
Japan	2	sens
France	3	null

Figure 5.2: Example Relations with CPT Column

Figure 5.2 shows our example R, S and T relations with their CPT column. For instance, the join path $R \bowtie S$ will be appended to the CPT column of all the tuples $\in S^R_{ns}$. Additionally, the CPT column of all tuples in R_s will be set to *sens*.

Next, we will provide an algebraic framework to split queries assuming such a CPT column exists. We will later present the details of CPT column creation for a given workload of queries.

5.3 Our Approach

In developing our approach, we first make an assumption that the given query Q is a single-block query, i.e., it contains a single SELECT-FROM-WHERE clause. Note that Q may contain aggregations (min, max, avg, etc.), group by, order by and having clause. A large class of SQL queries are either single block (and/or can be transformed into single block queries using rewrite rules [64, 65]. Our algorithmic development in this chapter will be limited to single block queries, though we will discuss ideas/approaches of extending the work to a more general class of SQL which may contain nested SQL queries with multiple blocks.

We will illustrate our splitting techniques using the following workload W, that consists

of three single-block queries Q_1, Q_2, Q_3 (given next) on relations $R(A, C, D)$, $S(A, B, D)$, $T(B, C, E)$ and $U(E, F)$, where only R has sensitive records.

$$
\begin{aligned}
Q_1 &: \; R \underset{R.A=S.A}{\bowtie} S \underset{S.B=T.B}{\bowtie} T \\
Q_2 &: \; R \underset{R.D=S.D}{\bowtie} S \underset{S.A=R.A}{\bowtie} R \\
Q_3 &: \; R \underset{R.C=T.C}{\bowtie} T \underset{T.E=U.E}{\bowtie} U
\end{aligned}
$$

Figure 5.3: Example Queries

We will make an assumption that the queries are *acyclic* or *tree* queries. Acyclic queries are defined based on the join graphs of queries.

Assume $graph(Q)$ be the function that returns a labeled undirected graph of a single block query q,

$$graph(Q) = G(V_Q, E_Q, L_Q) \tag{5.6}$$

where V_Q, E_Q and L_Q refers to the vertices, edges and labels in the graph. Each vertex in the $graph(Q)$ corresponds to a relation used in q. Two vertices corresponding to R_i and R_j are connected by an edge if there is a join condition C between these two relations in q. The label of such an edge will be C. The join graphs of the example queries, Q_1, Q_2 and Q_3, given at Figure 5.3, are provided at Figure 5.5.a, b and c respectively. As shown in Figure 5.5.a, R and S vertices are connected via edge $S.A = R.A$ (c_1) and R and T vertices are connected via edge $R.C = T.C$ (c_3).

If a relation R_x occurs more than once in a query q, each occurrence of R_x corresponds to a different vertex in the join graph. The first occurrence of R_x corresponds to a vertex labeled by the original relation name, R_x; whereas the vertex corresponding to the i^{th} occurrence of

R_x $(i > 1)$ is denoted as R_x^i. For instance, since relation R is used twice in Q_2, there are two vertices related with R in Figure5.5.a. The vertex that corresponds to the second occurrence of R is labeled as R^1.

A query is referred to as a *acyclic (or tree) query* iff its join graph corresponds to a tree, otherwise it is referred to as *cyclic query*. For instance, all the queries in our example are *a acyclic query*.

We focus our development for the case when queries are acyclic (or tree queries). In an acyclic query, the conditions associated with adjacent relations R_i and R_j in a join chain refer to only the attributes of R_i and R_j (not to the attributes of relations that appear either prior to or later in the query).

While restricting the algorithmic approach to only acyclic queries may seem limiting at first. We note that frequently, specially when dealing with foreign key joins, queries are often acyclic. For instance, 10 of 11 single-block TPC-H queries are acyclic. We further note that, even though our technique is developed in the context of acyclic queries, it can be used in the context of queries with cyclic conditions by simply postponing the check for some join conditions and applying them as a selection condition after the acyclic join processing. We will explain such extensions in further detail, after discussing how we deal with acyclic queries. Thus, for the remainder of this section, we will assume queries are acyclic.

The execution tree of a typical single block acyclic query Q consists of a single selection, projection, join (Q^{SPJ}) block at the bottom followed by some aggregation, sorting or duplicate elimination operators. We first explain how an SPJ block is split and then provide rules to split other possible operators in the execution tree.

Before going into the details, recall that each relation R_i in our dataset is placed across the hybrid cloud as follows:

1. All the R_i records are stored entirely on the private cloud with appropriate CPT columns,

2. All the non-sensitive R_i records, $R_{i_{ns}}$ are stored on the public cloud without any CPT column.

Our goal in this section is to develop rules to rewrite a query Q to generate split execution plan for Q. Our split rules will generate three queries – Q_{priv}, Q_{pub} and Q_{merge} such that

- Q_{pub} is executed over the data stored in public machines, that is, on non-sensitive parts of the relations

- Q_{priv} is executed over the data stored in private machines. In particular, for efficiency Q_{priv} will only access the data in relations that is either sensitive or for which the CPT column is sensitive

- The partitioning will guarantee correct execution - that is, the results produced by applying Q_{merge} to the outputs of Q_{pub} and Q_{priv} equals to the answers of Q, when applied to the full database.

We will illustrate how a single block query query can be split using a modified version of Query Q3 in TPC-H workload.

```
SELECT l_orderkey, sum(l_extendedprice) as revenue
FROM C, O, L WHERE c_custkey = o_custkey AND
l_orderkey = o_orderkey AND c_mktsegment = 'BUILDING' AND
o_orderdate < '1995-03-15' AND l_shipdate > '1995-03-15'
GROUP_BY l_orderkey ORDER BY revenue
```

where C, O, L stands for *Customer*, *Orders* and *Lineitem* tables respectively.

The query execution trees of our example query, q is given at Figure 5.4.a, where

$p_1 : c\_mktsegment = BUILDING$, $p_2 : o\_orderdate < 1995-03-15$,

$p_3 : l\_shipdate > 1995-03-15$, $c_x : c\_custkey = o\_custkey$,

$c_y : o\_orderkey = l\_orderkey$,

$(G,\ A) : (l\_orderkey,\ sum(l\_extendedprice) \rightarrow revenue)$

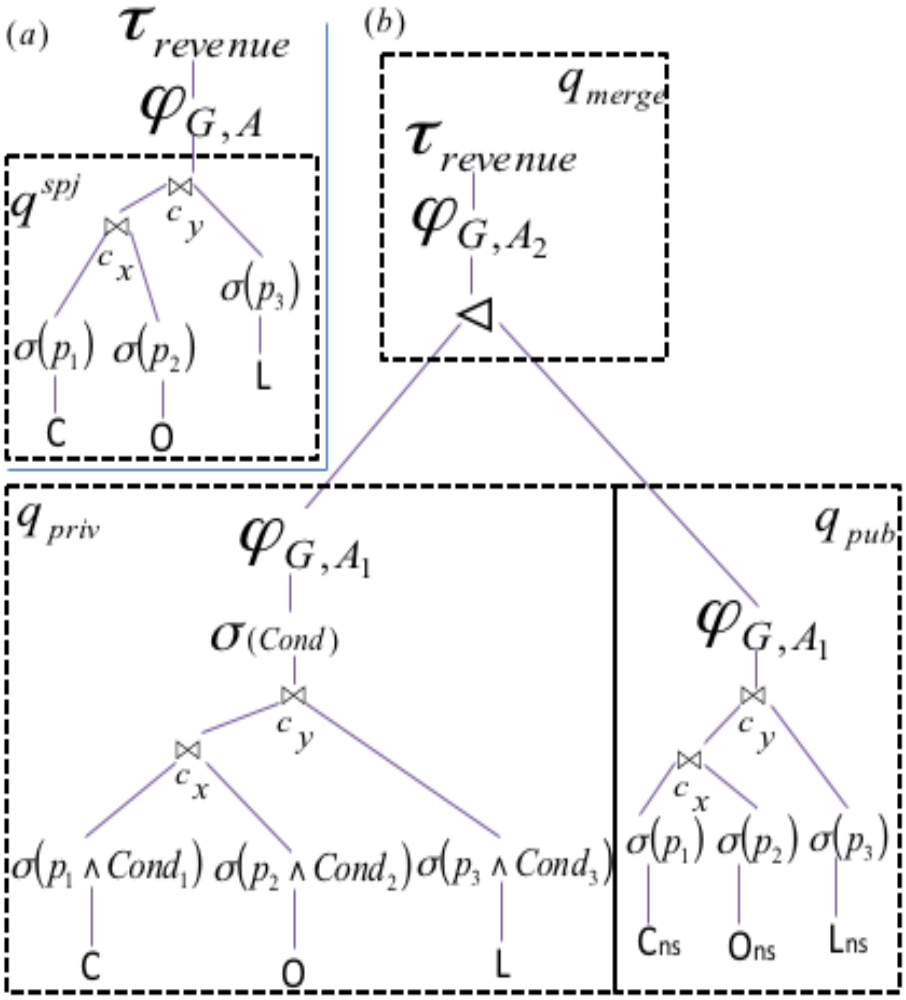

Figure 5.4: The Execution Tree for example query, q

5.3.1 Splitting SPJ block Q^{spj}

The rule to split Q^{spj} varies based on the existence of join operation. Assume that Q is a single relation query and so Q^{SPJ} does not involve any join operation, i.e. , Q^{SPJ} =

$\prod_{A}(\sigma_P(R))$. We can split such a Q^{spj} as

$$Q^{spj}_{priv} = \prod_{A}(\sigma_{P \vee Contains(CPT,sens)}(R)),$$
$$Q^{spj}_{pub} = \prod_{A}(\sigma_P(R_{ns})). \tag{5.7}$$

If Q is a multi-relational query, its Q^{spj} block can be expressed as

$$\prod_{A_1}(\sigma_{p_1}(R_1)) \underset{C_1}{\bowtie} \prod_{A_2}(\sigma_{p_2}(R_2)) \ldots \underset{C_{n-1}}{\bowtie} \prod_{A_n}(\sigma_{p_n}(R_n)) \tag{5.8}$$

, where R_i for $1 \leq i \leq n$ is a relation in the given input dataset. Each relation R_i has corresponding sensitive and non-sensitive partitions R_{i_s} and $R_{i_{ns}}$s. Rule to split Q^{spj} into public Q^{spj}_{priv} and private Q^{spj}_{pub} sub-blocks is as

$$Q^{spj}_{priv} = \underset{Cond}{\sigma}\Big(\prod_{A_1+CPT}(\sigma_{p_1 \wedge Cond_1}(R_1)) \underset{C_1}{\bowtie} \ldots \underset{C_{i-1}}{\bowtie} \prod_{A_i+CPT}(\sigma_{p_i \wedge Cond_i}(R_i)) \ldots \underset{C_{n-1}}{\bowtie} \prod_{A_n+CPT}(\sigma_{P_n \wedge Cond_n}(R_n))\Big) \tag{5.9}$$

and

$$Q^{spj}_{pub} = \prod_{A_1}(\sigma_{p_1}(R_{1_{ns}})) \underset{C_1}{\bowtie} \prod_{A_2}(\sigma_{p_2}(R_{2_{ns}})) \ldots \underset{C_{n-1}}{\bowtie} \prod_{A_n}(\sigma_{p_n}(R_{n_{ns}})) \tag{5.10}$$

, where $Cond$ is equal to

$$Contains(R_1.CPT, sens) \vee Contains(R_2.CPT, sens) \vee \ldots \vee Contains(R_n.CPT, sens) \tag{5.11}$$

and $Cond_i$ checks whether the R_i's CPT column contains any of the values required to obtain complete set of sensitive Q^{spj} answers. The set of required CPT values consists of *sens* and set of join paths P_i. *sens* value is required to fetch sensitive R_i records, whereas the P_i is required to obtain minimum set of non-sensitive R_i records to answer Q^{spj}_{priv}. P_i can be stated formally as,

$$\forall 1 \leq i \leq n,\ P_i = \Big\{ \bigcup_{j=1}^{i-1} (R_j \underset{C_j}{\bowtie} ... \underset{C_{i-1}}{\bowtie} R_i),\ \bigcup_{j=i+1}^{n} (R_i \underset{C_i}{\bowtie} ... \underset{C_{j-1}}{\bowtie} R_j) \Big\} \tag{5.12}$$

Using P_i's definition, $Cond_i$ can be defined as

$$Cond_i = Contains(R_i.CPT, sens \mid Any\ Path\ p \in P_i) \tag{5.13}$$

To illustrate, the spj block of our example query is also indicated as q^{spj} in Figure 5.4.a. In our query splitting strategy, we first split the q^{spj} using our spj block split rule. The outcome of splitting q^{spj} is depicted at Figure 5.4.b where

$Cond : Contains(C.CPT, sens) \vee Contains(O.CPT, sens) \vee Contains(L.CPT, sens)$

$Cond_1 : Contains(C.CPT, sens \mid O \underset{c_x}{\bowtie} C \mid L \underset{c_y}{\bowtie} O \underset{c_x}{\bowtie} C)$

$Cond_2 : Contains(O.CPT, sens \mid C \underset{c_x}{\bowtie} O \mid L \underset{c_y}{\bowtie} O)$

$Cond_3 : Contains(L.CPT, sens \mid O \underset{c_y}{\bowtie} L \mid C \underset{c_x}{\bowtie} O \underset{c_y}{\bowtie} L)$

5.3.2 Splitting The Higher Level Operators

The way the higher level operators are executed is common: (1) Partially execute them on both private and public cloud, (2) Transfer the partial public results to the private cloud and merge it with partial private results, (3) Do another post-processing over the merged

data at the private cloud to obtain the complete set of results. Such a post-processing will be in fact a part of Q_{merge} in our *split-strategy*.

Duplicate Elimination ($\delta(R)$): To eliminate duplicates, we rewrite δ operator as

$$\delta(R) = \delta(\delta(R_s) \lhd \delta(R_{ns})) \tag{5.14}$$

, where $\delta(R_s)$ and $\delta(R_{ns})$ are executed on private and public cloud independently. The outputs of such a split, however, may contain more tuples than the original $\delta(R)$. To remove the duplicates between R_s and R_{ns} securely, we transfer the results of $\delta(R_{ns})$ from public to private cloud ($\lhd$) and perform another duplicate elimination operation between the results of $\delta(R_s)$ and $\delta(R_{ns})$. This final elimination will be a part of Q_{merge}. At the end, again all the outputs would be present at the private cloud.

Grouping and Aggregation ($\varphi_\alpha(R)$): We denote a grouping and aggregation operator by $\varphi_\alpha(R)$, where $\alpha = \alpha_G \cup \alpha_A$, α_G corresponds to a list of grouping attributes, and α_A to a set of aggregation operations. For instance, $\varphi_{A_1, MAX(A_2) \rightarrow B}(R)$ means that R records are grouped using attribute A_1 and for each group created, maximum of A_2 attribute is computed and renamed as attribute B. In this example, $\alpha_G = A_1$ and $\alpha_A = MAX(A_2) \rightarrow B$. Note that if $\alpha_A = \emptyset$, only grouping is performed. Conversely, if $\alpha_G = \emptyset$, then the aggregation function is carried on the entire relation.

The split rule of the grouping and aggregation operator is as

$$\varphi_\alpha(R) = \varphi_{\alpha^1}(\varphi_{\alpha^2}(R_s) \lhd \varphi_{\alpha^2}(R_{ns})) \tag{5.15}$$

, where $\alpha_G = \alpha_G^1 = \alpha_G^2$. That is, the original α operator is split into three parts: 1) partial private side grouping and aggregation $O_{priv} = \varphi_{\alpha^2}(R_s)$; 2) partial public side grouping

and aggregation $O_{pub} = \varphi_{\alpha^2}(R_{ns})$; 3) Merging the results $\varphi_{\alpha^1}(O_{priv} \cup O_{pub})$. While the same grouping operation is performed in α, α^1 and α^2, their aggregation function might be different.

Let f be a function performed over attribute X and the result is reported as column Y, $f(X) \rightarrow Y$. If f is a transitive aggregation function, such as *SUM*, *COUNT*, *MAX*, *MIN* or *TOP*; then $f(X) \rightarrow Y$ will be replaced with $f(X) \rightarrow Y^{'}$ in α_A^2 and $f(Y^{'}) \rightarrow Y$ in α_A^1. If f is partially transitive, namely *AVG*; $AVG(X) \rightarrow Y$ is replaced with two statements in α_A^2: $SUM(X) \rightarrow Y_1$ and $COUNT(*) \rightarrow Y_2$; and replaced with a single statement in α_A^1, $SUM(Y_1)/SUM(Y_2) \rightarrow Y$. Finally, if f is not a transitive function, then the no operation is done over X attribute in α_A^2 and all the f computation is carried on α_A^1.

To illustrate, the example above, $\varphi_{A_1,MAX(A_2)\rightarrow B}(R)$, is split as

$$\varphi_{A_1,MAX(B')\rightarrow B}(\varphi_{A_1,MAX(A_2)\rightarrow B'}(R_s) \cup \varphi_{A_1,MAX(A_2)\rightarrow B'}(R_{ns})). \tag{5.16}$$

Note that $\alpha^1 = A_1, MAX(B^{'}) \rightarrow B$ and $\alpha^2 = A_1, MAX(A_2) \rightarrow B^{'}$ in this example.

Using the split rule above (Equation 5.15), $\varphi_{\alpha^2}(R_s)$ will be executed on the private cloud, while $\varphi_{\alpha^2}(R_{ns})$ is performed over the public cloud. All the $\varphi_{\alpha^2}(R_{ns})$ results will be transferred to the private cloud. Once the transfer is done, φ_{α^1} operation will be performed on the private cloud. φ_{α^1}'s output will be labeled as sensitive, since it mixes non-sensitive inputs with some sensitive ones.

Sorting ($\tau_L(R)$): The operation to sort relation R based on the attributes in list L can be implemented similar to the duplicate elimination operator. That is, we first sort the sensitive records of R in the private cloud and the non-sensitive ones in the public cloud. Then, we transfer the public results to the private cloud and apply a final merge-sort operation among the private sensitive results and public non-sensitive results. So, $\tau_L(R)$ operator can be split

as:

$$\tau_L(R) = \tau_L^2 \left(\tau_L^1(R_s) \lhd \tau_L^1(R_{ns})\right) \tag{5.17}$$

where τ_L^2 is a single-pass merge-sort operation. After applying this split rule, the private cloud will have all the sorted results.

The higher level operator in our example query is a 'grouping and aggregation' operator, $\varphi_{G,A}$. Thus, we split this operator in the execution tree using our grouping and aggregation split rule. The result is depicted at Figure 5.4.b where

$A_1 : sum(l\_extendedprice) \rightarrow revenue^{'}$, $A_2 : sum(revenue^{'}) \rightarrow revenue$

Once we split the spj block and next higher level operator, all the operators on the left hand side of $\lhd$ operator becomes q_{priv}, as shown at Figure 5.4.b and all the operators lying on the right hand side of $\lhd$, again as shown at Figure 5.4.b will be q_{pub} and the operators staying above $\lhd$ will be q_{merge}.

Next, we discuss how to efficiently create this CPT column on each relation for a given query workload W.

5.3.3 Creating CPT Column

Let D be a user given dataset, where $D = R_1, R_2, \ldots, R_n$ and W be the single-block acyclic query workload[2] that will be executed over D (i.e. $W = Q_1, Q_2, \ldots, Q_n$). Our objective is to create CPT column based on the given expected workload. However, it has to be done efficiently over the private cloud data, specially since CPT column is generated on the

[2]We discuss the CPT column creation for the case when the expected workload of queries is known. We will discuss the case when the workload is not available separately, later

private cloud with limited computational capability. We will illustrate our CPT creation technique using our example workload W, that consists of three single-block queries Q_1, Q_2, Q_3 (given earlier) on relations $R(A, C, D)$, $S(A, B, D)$, $T(B, C, E)$ and $U(E, F)$, where only R has sensitive records.

CPT column is mainly created through 2 steps: 1) We analyze the expected workload of queries and extract the necessary join paths from them. 2) Based on these join paths, we set the CPT columns of the records in our dataset.

Extracting Required Join Paths : We implemented a CPT creation module that analyzes each query in the workload once to extract the set of required join paths (the technique for which is described in Algorithm 1) in order to execute the queries in the workload using *split-strategy*. In developing our approach to extract required join paths, we will use the notion of join graph for a query, described at the beginning of the section.

Iterating over the each query in W, Algorithm 1 first constructs the join graph of each Q_i, V_i (Line 3). Then, we do a pass over the join graph V_i to extract the set of required join paths so as to efficiently execute Q_i in *split-strategy*. For instance, in Figure 5.5, the join graph of each query in our example workload is given.

Note that we only need to extract the join paths whose first relation have some sensitive data. For instance, in Q_1's join graph, there is a join path p between S and T, i.e $p = S \underset{c_2}{\bowtie} T$. Since S does not have any sensitive record, $CP(T, p)$ should not be computed. [3] In order to guarantee that such 'co-partitionings' are never computed in our next step, we need to eliminate such paths. Therefore in Algorithm 1, we first iterate over any pairs of vertices, R_x^m and R_y^n in the join graph (Line 4) and eliminate the ones whose first relation does not have any sensitive records (Line 5). The join paths extracted from Q_1, Q_2 and Q_3 is given below their join graphs in Figure 5.5. For instance, the query paths extracted from Q_3 are

[3] Because, there is no non-sensitive T record that can join with a sensitive record from S.

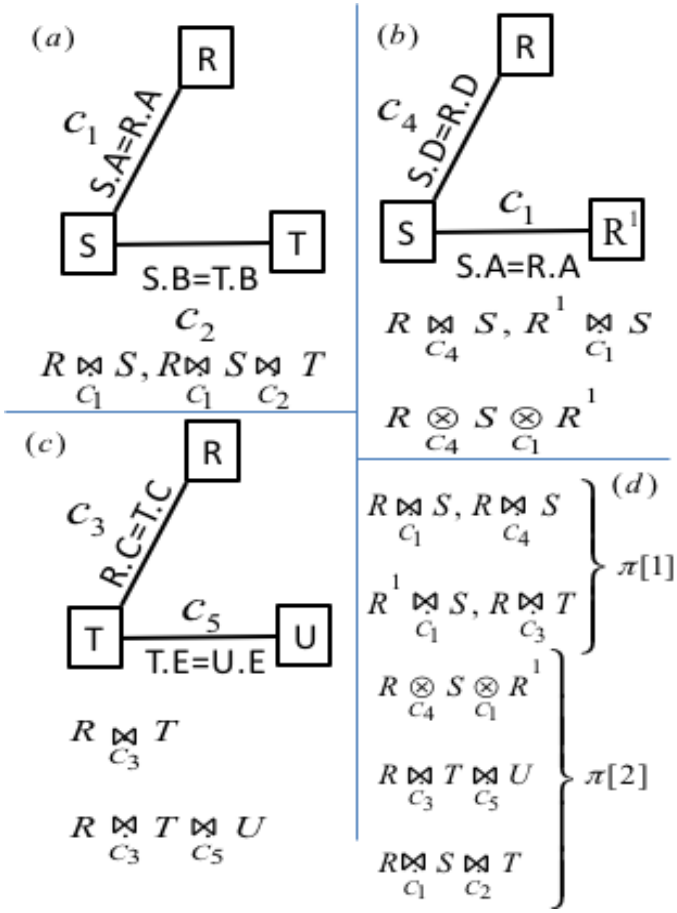

Figure 5.5: Join Graphs and Path Creation

$R \underset{c_4}{\bowtie} T$ and $R \underset{c_4}{\bowtie} T \underset{c_5}{\bowtie} U$. Due to the limited space in Figure 5.5, we represented every p and the path that is reverse of p ($p.reverse$) as a single path using $\otimes$ instead of $\bowtie$.

Finally, we store all the required join paths in a data structure, denoted as π in such a way that if the length of extracted join path p is l, then we store p in $\pi[l]$ (line 6-7). Figure 5.5.d provides the set of all required join paths extracted from our example workload and illustrates how these join paths are stored within the data structure π. Note that $R \underset{c_1}{\bowtie} S$ and $R \underset{c_2}{\bowtie} S$ are two different join paths extracted from our workload, and therefore stored separately in $\pi[1]$.

Creating CPT Column : Upon creating π, our Algorithm 2 takes it as an input and forms the CPT column in every relation.

Input: W
Output: π

```
1  π ← ∅;
2  foreach Q_i ∈ W do
3  |  V_i ← graph(Q_i);
4  |  foreach (R_x^m, R_y^n) vertex pair in V_i do
5  |  |  if R_{x_s} ≠ ∅ then
6  |  |  |  p ← V_i.Path(R_x^m, R_y^n) ;
7  |  |  |  π[p.length].add(p) ;
8  |  |  |  if π.maximumLength < p.length then
9  |  |  |  |  π.maximumLength = p.length ;
10 |  |  |  end
11 |  |  end
12 |  end
13 end
14 return π
```

Algorithm 1: Path Statistics Creation

Algorithm 2 first creates the initial private versions of each relation by appending a new column CPT (Line 2). To illustrate, based on our example dataset, we alter the tables R, S, T and U on the private cloud by appending a CPT column to them. Algorithm 2 initializes the value of this column to *sens* for sensitive records and *null* for non-sensitive records (Line 3-9). In our example dataset, CPT columns of all S, T,U records as well as all non-sensitive R records are initially set to *null*. Only, the CPT columns of sensitive R tuples is set to *sens*. Subsequently, Algorithm 2 iterates over the set of required paths based on their lengths (line 10-21), but first handles the base case, the paths with length 1, e.g. the paths : $R \underset{c_1}{\bowtie} S$, $R \underset{c_3}{\bowtie} T$ and $R \underset{c_4}{\bowtie} S$ etc. If a non-sensitive record t in R_i joins with a sensitive record from another relation R_j via existing 1-length path p, then our algorithm appends p to t's CPT column (Line 15-16). For instance, if a non-sensitive S tuple, s in our example semi-joins with any sensitive record from relation R based on condition c_1 and c_4, then the join paths $R \underset{c_1}{\bowtie} S$ and $R \underset{c_4}{\bowtie} S$ are appended to such s's CPT column.

After handling the base case, our algorithm continues to compute CPT column for all other remaining join paths in π using induction. Assume that p is a path such that its length

Input: π
Output: D_{priv}
foreach R_i *in* D **do**
  $[R_{i_{priv}}] \leftarrow [R_i]||CPT$;
  foreach *Tuple* t *in* R_i **do**
    if $t \in R_{i_s}$ **then**
      $R_{i_{priv}}.t \leftarrow t||sens$
    else
      $R_{i_{priv}}.t \leftarrow t||null$
    end
  end
  for $1 \leq length \leq \pi.maximumLength$ **do**
    foreach *Tuple* $t \in \sigma_{!Contains(CPT,sens)}(R_i)$ **do**
      foreach *Path* $p \in \pi[length]$ *where* $R_i^n = p.lastRelation$ **do**
        $p^{'} \leftarrow p - p.lastRelation$;
        $R_j \leftarrow p^{'}.lastRelation$;
        if $length = 1$ & $t \underset{c}{\ltimes} \sigma_{Contains(CPT,sens)}(R_j) \neq \emptyset$ **then**
          $t.CPT \leftarrow t.CPT + p$;
        else if $length > 1$ & $t \underset{c}{\ltimes} \sigma_{Contains(CPT,p')}(R_j) \neq \emptyset$ **then**
          $t.CPT \leftarrow t.CPT + p$;
      end
    end
  end
end
$D_{priv} \leftarrow \{R_1, R_2...R_n\}$;
return D_{priv}

Algorithm 2: CPT Column Creation

equal to k, $|p| = k$ and it is formed as $p' \underset{c}{\bowtie} R_i^n$ where $|p'| = k - 1$. To illustrate, let p be equal to the join path $R \underset{c_1}{\bowtie} S \underset{c_2}{\bowtie} T$ as in our example, then $p' = R \underset{c_1}{\bowtie} S$. Also, suppose that the last relation appeared in path p' is R_j^m. In our example, the last relation appeared in p' is S. Then, the tuples in $CP(R_j, p')$, in our example $CP(S, R \underset{c_1}{\bowtie} S)$, is already computed at the previous step. Note that $CP(R_j, p')$ can be extracted from R_j via $\sigma_{Contains(R_j.CPT,p')}$ operation. Namely, the records in $CP(S, R \underset{c_1}{\bowtie} S)$ can be obtained by running $\sigma_{Contains(S.CPT,R \underset{c_1}{\bowtie} S)}$ operation on S. So, Algorithm 2 takes each non-sensitive R_i tuple t and semi-joins with $CP(R_j, p')$ based on condition c. If the result is not an empty set, then inserts p to t's CPT column (line 17-18). To illustrate, we test for each non-sensitive T tuple, t whether it semi-joins with a tuple from $CP(S, R \underset{c_1}{\bowtie} S)$ based on condition c_2. If it does, then we append path p, $R \underset{c_1}{\bowtie} S \underset{c_2}{\bowtie} T$ to the CPT column of t. Finally, Algorithm 2 returns D_{priv}, which is the set of final versions of all relations, to be stored on the private cloud (line 23-24).

Note that we retrieve the set of required paths, π from a given workload of query templates. But, a user/organization may not know what kind of workload will be executed in the future. Under such a scenario, one can estimate the join paths by considering the primary key/ foreign key (p.k/f.k) constraints at the dataset schema as a join condition between tables. Given this schema graph where the tables are vertices and p.k/f.k constraints are edges , one can obtain all the $l - length$ join paths starting from any table that contains some sensitive data. Nevertheless, the join conditions in all these paths would be established based on p.k/f.k equality. To illustrate, Figure 5.6 depicts the schema graph for the TPC-H dataset. The tables having p.k/f.k constraints between them are connected with an edge labeled with the name of key. So, if a user wants to support the queries whose join graph contains at most $1 - length$ paths, then he/she can extract all the $1 - length$ join paths from Figure 5.6 (e.g. $C \underset{c\_custkey=o\_custkey}{\bowtie} O$ and $O \underset{c\_custkey=o\_custkey}{\bowtie} L$) and construct its CPT column based on these extracted paths. For instance, in TPC-H benchmark, all single level queries can be supported by allowing the maximum length to be as low as 3.

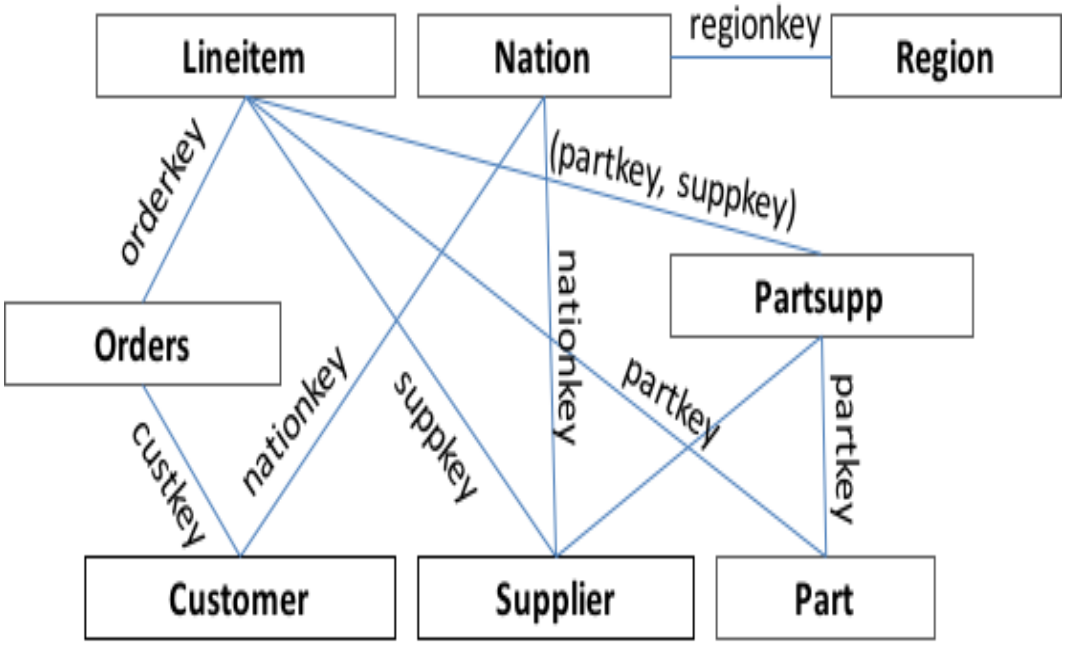

Figure 5.6: TPC-H Dataset Schema Graph

5.4 Implementation

We have so far discussed how we can derive a *split-strategy* for a given single block acyclic query Q. We furthermore described the algorithm for generating the CPT column. We have implemented the CPT computation algorithm as a MR job in Hadoop and as a Spark Job in Spark. Spark is a fast and general engine for large-scale data processing. In contrast to Hadoop's two-stage disk-based MapReduce paradigm, Spark's multi-stage in-memory primitives provides performance up to 100 times faster for certain applications [66]. Both Hadoop and Spark are designed to read and write data from and to HDFS, a distributed file system designed to run on commodity hardware. We used HDFS to store our data both at the private and public cloud. For querying relational data and storing metadata related with it, Hive has been used. Hive is an open source data warehouse software facilitates querying and managing large datasets residing in distributed storage like HDFS. It provides a mechanism to project structure onto this data and query the data using a SQL-like language, HiveQL [63]. We only discuss the details of Spark implementation, since both implementa-

tions are similar.

5.4.1 Join Path Representation

In our implementation, each join path extracted from the workload or given database schema is represented as a sequence of table ids and join condition ids in the string format. For instance, if the join path is $Customer \underset{c_x}{\bowtie} Orders \underset{c_y}{\bowtie} Lineitem$, then it is represented as "Cc_xOc_yL" and if c_x and c_y are primary key / foreign key equality checks, then the representation is abbreviated as "COL". If a non-sensitive record r is a part of multiple co-partitions, then the path ids of these co-partitions are concatenated using "," character and stored in r's CPT column in lexicographical order.

The containment check for paths on the record's CPT column is done via $RLIKE$[4] operator. For instance, assume that we want to run $R \underset{c_1}{\bowtie} S$ query and c_1 is a primary key/ foreign key equality condition. To run this query, we need non-sensitive S records that includes RS value within its CPT column. So we check this property by running $RLIKE(S.CPT, ".*, RS, .*")$ expression on table S.

5.4.2 CPT Column Creation

After extracting the set of required join paths either from the given workload or from the dataset schema, our CPT column construction algorithm is implemented by deploying a Spark job over the given dataset on the private cloud. To handle the operations between Line 3-9 in Algorithm 2, we deploy a map transformation such that its function reads the record r from any table R and outputs $\prod_A(r)$, where A is the set of all necessary R attributes. The set of necessary attributes are determined based on join conditions in all the join paths

[4]RLIKE operator performs a pattern match of a string expression against a pattern. The pattern is supplied as argument.

involving table R. If all join conditions are primary key foreign key equality checks, then A will be equal to at most all primary and foreign key attributes in table R, and thereby the size of $\prod_A(R)$ will be substantially less than the size of R. As a result of this transformation on table R, two different RDDs[5] are created R^{RDD}_{sens} and R^{RDD}_{null} based on the sensitivity of T's records.

All the operations that are done within the loop in Algorithm 2 (Line 13-22) is implemented as a sequence of complex transformations on temporarily created RDDs. We iterate over the set of extracted join paths (π) based on their lengths on ascending order. Suppose that the currently iterated path p is equivalent to $...\underset{c}{SR}$. We apply the following map transformation (M_1) on temporary $S^{RDD}_{p'}$ and R^{RDD}_{null}, where $p' = sens$ if $|p| = 1$ or $p' = ...S$ otherwise. M_1 can be formalized as

$$M_1(r) = \begin{cases} Out(r.joinAttr(c), null) & \text{if } r \in S^{RDD}_{p'} \\ Out(r.joinAttr(c), r) & \text{if } r \in R^{RDD}_{null} \end{cases} \tag{5.18}$$

where $r.joinAttr(c)$ returns the values of the r's join attributes based on condition c. Then, we join outputs of these two transformations and create a new temporary RDD, called as R^{RDD}_p.

Once all the paths are covered, we deploy another sequence of Spark transformations on the original dataset and temporary RDDs in order to obtain the final value of CPT column. For instance, to construct the table R's CPT column, we first apply the following map transformation (M_2) over R and set of temporary R^{RDD}s

[5] An RDD is a read-only, partitioned collection of records. RDDs can only be created through deterministic operations on either (1) data in stable storage or (2) other RDDs.

$$M_2(r) = \begin{cases} Out(r.primaryKey, p) & \text{if } r \in R_p^{RDD} \\ Out(r.primaryKey, r) & \text{if } r \in R \end{cases} \tag{5.19}$$

to get the outcome (K, V) pairs, where Ks are primary key values and Vs are either equivalent to the record itself (r) or a path p. At final stage , we first group by these (K, V) pairs based on their key ($groupByKey$) to return a dataset of $(K, List < V >)$ pairs. Then, we apply the map function M_3, whose code is given next, over the list of Vs for each key K (primary keys of R records) in order to construct the final version of each R tuple.

```
String tuple = ""
for (V : List<V>) {V is a path : tuple += V ? tuple = V + tuple}
return tuple
```

5.4.3 Table Partitioning

In our current approach, all the records of a table R in our dataset is stored together on the private side along with their CPT column. Since no index can be created on R's CPT column in Hadoop, Spark or Hive, when a single block query Q is executed on R, R must be scanned entirely to find appropriate tuples for Q_{priv}. Due to this, the initial scan cost of original Q would be equal to the scan cost of Q_{priv}. But, this scan cost can be significantly reduced by partitioning records based on their CPT column value and by physically storing each partition as a separate file. In this approach, we scan each table R file one more time and put the records to the corresponding file based on their CPT values. Finally, we name the output files as R_x where x stands for a possible CPT value in R. For instance, assume that in Lineitem table, the possible CPT values are $sens$,OL,COL and COL, OL; so we create a a separate $Lineitem$ file for each of these possible values (e.g.

$Lineitem\_sens, Lineitem\_OL, Lineitem\_COL$ *and* $Lineitem\_COL, OL$).

In Hive, we add these files as a separate partition of table R where partition ids are equal to the CPT values of table files. For instance, in *Lineitem* table, the partition ids would be $sens$,OL,COL and COL, OL. During the query processing, instead of running our selection operators on a CPT column, we execute them over the partition ids of tables. For instance, $Cond_1$ at Figure 5.4.b will be replace with :

$$Cond_1 : Contains(C.pid, sens \mid O \underset{c_x}{\bowtie} C \mid L \underset{c_y}{\bowtie} O \underset{c_x}{\bowtie} C)$$

in this new approach, where *pid* stands for the partition id. For any table T, the number of partitions could be as much as 2^l, where l is the number of required join paths ends at relation T. In our experiments with TPC-H benchmark, however, we have not encountered with such a partition blow-up for any table. The maximum value of l was 3 in our experiments.

5.5 Experimental Evaluation

To the best of our knowledge, our approach is the first one that partitions queries across hybrid clouds in order to achieve secure query processing. By conducting extensive experiments in this section, we compared the performance of our proposed solutions against *All-Private* and *All-Public*, in which all computation is handled on the private side or public side only. Notice that *All-Private* is a secure solution, whereas *All-Public* is not.

5.5.1 Experimental Settings

Experimental Setup: We conducted our experiments using two clusters, one at SDSC [67] and the other at UC Irvine. The first cluster comprised of 72 nodes and is used as public cloud, while the second comprised 9 machines and is used as private cloud. While a node

in our public cloud has Intel EM64T Xeon E5 2.6GHz 4 core cpu, 128GB Memory and 320GB SDD disk, a private cloud node consists of a Dual-Core AMD Opteron processor with $\approx$ 631GB disk space and 8GB of main memory. The machines on our private cloud are connected to each other with a 1Gbps ethernet network. The average data transfer rate that we measured between any two machines is approximately 100MB/s. Depending on the private/public node ratio, we excluded some of the public cloud nodes from the hybrid cloud. Finally, the clusters were set up using Spark v1.4.1 and Hadoop v2.6.0 as our distributed query execution engines. Additionally, we used Hadoop v2.6.0's HDFS as our storage engine. Finally, Hive v1.2.1 is deployed on the clusters for storing relational database metadata and querying.

Dataset and Sensitivity In our experimental evaluation, we used the TPC-H benchmark. We generated the dataset with scale factor 200 (i.e. 200GB). We declared the some of records in *customer* and *supplier* tables as sensitive. We varied the ratio of sensitive records in each table from 1% to 50%.

Workload: Original TPC-H benchmark consists of 22 queries and 11 of 22 TPC-H queries were single block queries. Of these 11 queries, 5 were doing a computation entirely on non-sensitive data, namely using neither *customer* nor *supplier* table. Therefore, we used the remaining 6 queries as of our workload for our approach: $Q3$, $Q5$, $Q7$, $Q8$, $Q9$ and $Q10$.

Query Execution : We manually split each query Q in our workload to Q_{priv}, Q_{pub} and Q_{merge} by applying our split rules. We then manually sent Q_{priv} (Q_{pub}) to the Hive server running on private (public) cloud. Once Q_{pub} is completed, we immediately transferred its results from public HDFS to the private HDFS and added the transferred results as a temporary relation to the private cloud Hive. Once this operation and Q_{priv} gets completed, we sent Q_{merge} to private cloud Hive server. Depending upon the experiment, Hive servers pushed the queries to Hadoop or Spark for execution, Since all the operations are done manually, the running time of query Q in *split-strategy* is computed as :

$$max(time(Q_{priv}), time(Q_{pub}) + transfer(output(Q_{pub}))) + time(Q_{merge}) \quad (5.20)$$

where $time(X)$ and $transfer(Y)$ denotes the running time X and transfer time of Y from public to private cloud. Note that the Q_{pub} results are brought to the private cloud, even though Q_{merge} is a void query. Thereby, the transfer time for Q_{pub}'s results are always included to calculate the running time of *split-strategy*.

5.5.2 Experimental Results

In this section, we outline the experiments that compare the performance of our two *split-strategy* techniques with *All-Private*. Recall that in our first technique (*CPT-C*) every record in a table at the private cloud contains a CPT column and they are physically stored together; whereas in our second approach, *CPT-P*, the tables are partitioned based on their record's CPT column and each partition is stored separately. Each partition file then appended to the corresponding Hive table as a separate partition, so in querying stage, Hive filters out the unnecessary partitions for that particular query.

Given that private cloud has a fixed computational power, then there are two factors that can impact the running time of queries when their execution is split across the hybrid cloud. First one is the public cloud's computational power. Smaller a public cloud implies a higher public side query execution time in the *split-plan* and thereby could increase the overall running time of the given workload in our *CPT-C* and *CPT-P* solutions. Alternatively, the amount of sensitive data within the input data directly impacts the execution time of private side queries in *split-plan*s and thereby, can again increase the overall completion time of the given workload in our solutions . To demonstrate the impact of each of these parameters, we first comprehensively evaluated our solutions by varying one of these two criterion, while

fixing the other one. Note that, figures on the left indicate the results for Hadoop, while figures on the right display the outcomes for Spark.

Public Cloud Size: We deployed 4 distinct hybrid cloud scenarios in which number of public machines is varied from 9 to 72 by doubling the amount in each scenario. We set the input sensitive data ratio to 5% for these experiments. The results presented in Figure 5.7 indicate that larger the public cloud incurs a smaller workload execution time for both of our solutions in Hadoop. But in Spark, the workload execution time does not change when more machines are added to the public cloud. Note that Spark stores the intermediate data in the memory during query processing. Since our public machines have significantly larger memory than the private cloud machines (128GB vs 16GB), the execution of public side queries Q_{pub}s takes less time than the private side queries, Q_{priv}s, even when the number of public machines are as low as 9. Therefore, adding more public machines speeds the up Q_{pub}s in Spark, but does not change the overall execution time of the queries. Finally, when number of public machines are equal to 72, our CPT-P solution is $10.2x$ faster than All-Private in Hadoop and it is $5.3x$ faster than All-Private in Spark.

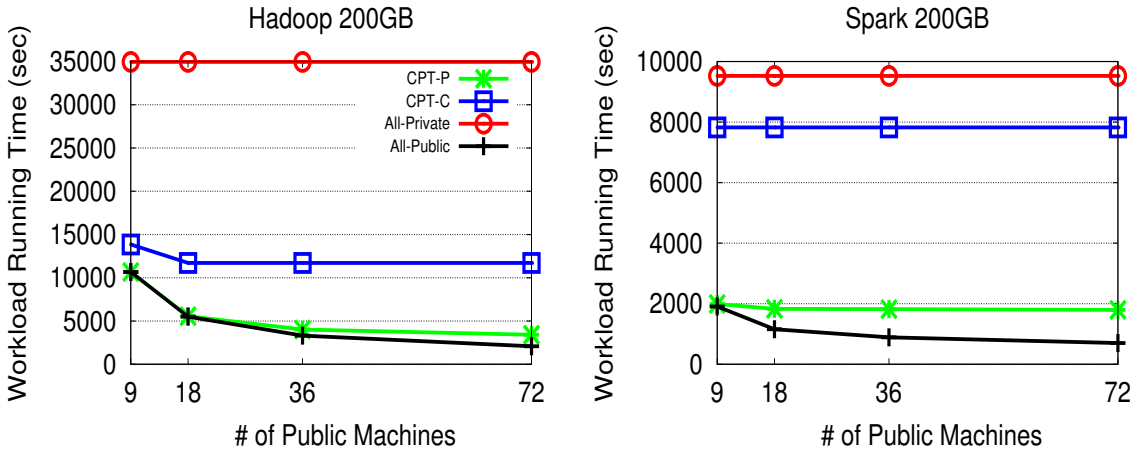

Figure 5.7: Workload Running Times For Different # of Public Machines

Sensitive Data Ratio: For these experiments, we varied the amount of sensitive records $(1, 5, 10, 25, 50\%)$ in *customer* and *supplier* tables. Also, we set number of public machines to

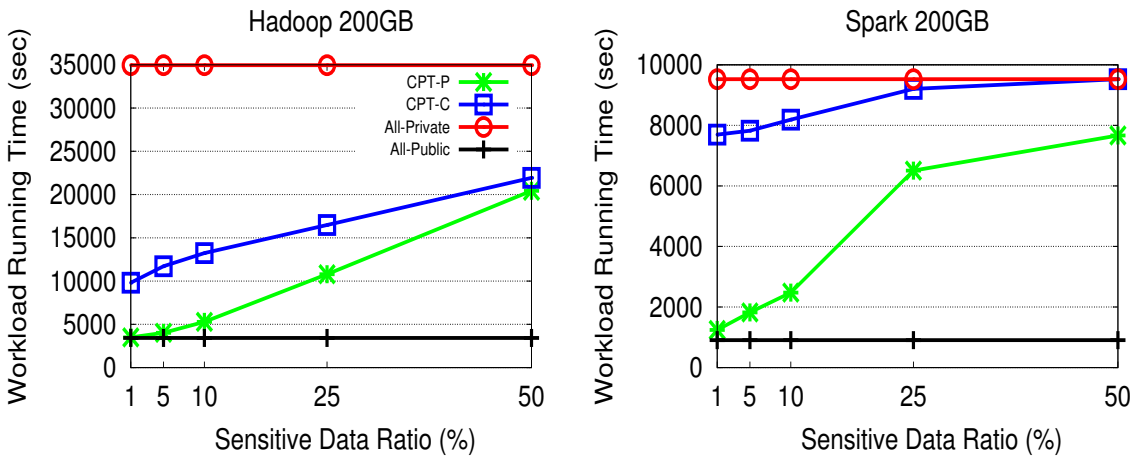

Figure 5.8: Workload Running Times For Different Sensitivity Ratios

36. As expected, Figure 5.8 shows that a larger percentage of sensitive data within the input leads to a longer workload execution time for both, CPT-C and CPT-P in Hadoop and Spark. The reason behind this is that a higher sensitive data ratio results in more computations being performed on the private side and implies a longer query execution times in *split-strategy*. When the sensitivity ratio increases, CPT-P's scan cost increases dramatically. Since the scan cost of queries is the dominant one compared to the their computation cost (join, filtering etc.), when Spark is used for query processing, CPT-C provides a very low performance gain in Spark compared to All-Private. Because, the scan cost of these two approaches are same. Overall, when sensitivity ratio is as low as 1%, our CPT-P solution provides $8.7x$ speed-up in Hadoop and $5x$ speed-up in Spark compared to All-Private.

Recall that we created the CPT column using a Spark job for CPT-C solution. We then physically partitioned tables for CPT-P solution. Figure 5.9 shows how much time we spent in preparing private cloud data for both CPT-C and CPT-P. It also indicates the gains of these approaches compared to All-Private in terms of the overall workload execution time . As indicated in Figure 5.9, until 25% sensitivity, the data preparation time is less than the performance gain of CPT-P in Hadoop; whereas in Spark, data preparation times is always higher than the performance gain for both CPT-P and CPT-C. Note that, we prepare the

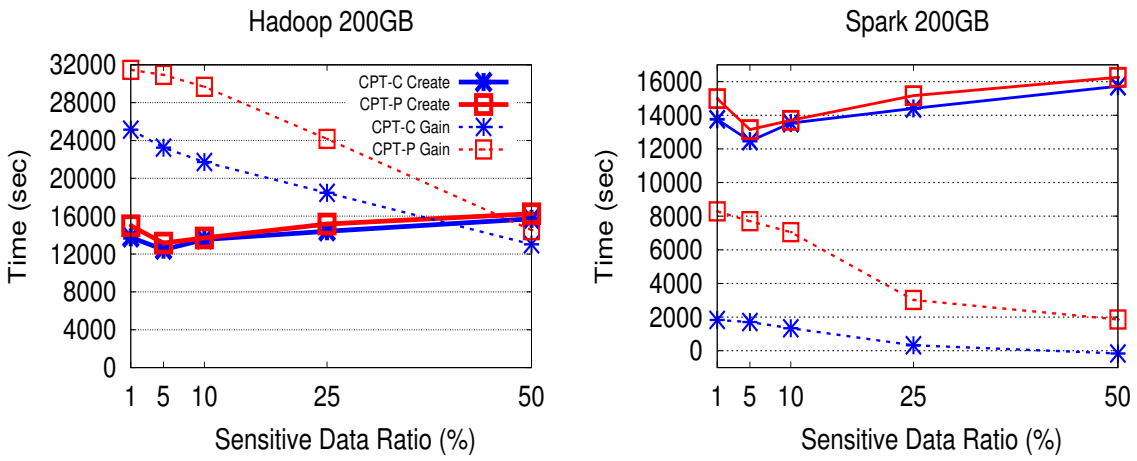

Figure 5.9: CPT Creation Times For Different Sensitivity Ratios

CPT column only once on a static data for an expected workload that will more likely be executed more than once with different selection and projection conditions. So, in Spark, if the sensitivity ratio is as high as 10%, executing the expected workload more than once will be enough for the performance gain of CPT-P solution to be higher than the overhead of data preparation time.

5.6 Related Work

Our work builds upon a significant body of prior work on data partitioning (e.g., [68–71]), distributed query processing (e.g., from systems such as SDD-1 [72] to DISCO [73] that operates on heterogeneous data sources, to internet scale systems such as Astrolabe [74], and cloud systems [75]. We limit the discussion to only a few that are most relevant.

A more recent paper [71] looks at the partitioning problem in distributed databases for supporting OLTP queries efficiently. The objective is to improve throughput by reducing the transaction time, which is dependent on whether it accesses data on a single or multiple nodes and therefore, reducing the number of multi-node transactions can significantly increase

throughput. Their notion of limiting the computation to only single site in order to gain efficiency is in spirit similar to our approach of splitting each query to three independent sub-queries, each of which runs on a single site, public or private cloud. *split-strategy*They propose a graph based data partitioning approach based on a well known class of algorithms called METIS [76], which are known to generate balanced partitions in practice. However, a graph partitioning approach may not be suitable to partition relational workloads and datasets, since graph size is proportional to the number of records in tables.

One of the many works that is related to ours is [77]. Their aim is to efficiently process data warehousing queries in a split execution environment. Their approach push the parts of the query to a higher performing database layer, and the rest of query is processed in a more generic MapReduce framework. In order to execute as many joins as possible locally in the database layer, they perform aggressive hash-partitioning. They co-partitions the tables via many steps of foreign key/primary key references in order to keep join local to each node (referential partitioning). Our idea to co-partition non-sensitive records based on the sensitive records via possible join paths is in spirit similar to the referential partitioning approach.

5.7 Extension

We presented a strategy that partitions the execution of single-block acylic queries execution over a hybrid cloud, while ensuring that the sensitive data/information is never leaked to the private cloud. The main challenge is to provide efficiency by partitioning queries compared to executing everything over the private cloud. To overcome this challenge, we create a special column over the private cloud data. Using our additional column, our query partitioning approach dramatically improved the overall execution time of the given queries by as much as 10× as compared to private cloud only solutions.

SQL enables users to write more complex queries such as cyclic queries, nested queries or queries involving more complex operators (outer joins, union, set difference etc.). Now, in this section, we discuss our solution ideas about how to create a *split-strategy* for such complex queries.

Cyclic Queries : Consider the following cyclic query Q_4, which is a modified version of Q_1 given at Figure 5.5.a.

$$Q_4 = (R \underset{c_1}{\bowtie} S) \underset{c_2 \wedge c_3}{\bowtie} T \tag{5.21}$$

where $c_3 : R.C = T.C$. Also, let us assume that T also has some sensitive data. Once we split Q_4 using our approach explained in Section 5.3, the selection condition over relation S, $Cond_2$ will be as

$Cond_2 : Contains(S.CPT, sens \mid R \underset{c_1}{\bowtie} S \mid T \underset{c_2 \wedge c_3}{\bowtie} S)$.

Note that since c_3 does not involve the relation S, $T \underset{c_2 \wedge c_3}{\bowtie} S$ is not a valid join path. Therefore, SPJ block of cyclic queries cannot be split using our SPJ block split rule.

Strategy 1 : One way to address this concern is to convert cyclic SPJ block to an acyclic one by pulling up some of the conditions from the join stage and apply them as a separate selection operator after the join processing. For instance, Q_4 can be converted into the following query,

$$Q_4^{'} = \sigma_{c_3}(R \underset{c_1}{\bowtie} S \underset{c_2}{\bowtie} T). \tag{5.22}$$

Since $Q_4^{'}$ is a single block acyclic query, we can create an efficient *split-strategy* for it using our previous split rules. Note that the execution of $Q_4^{'}$ is already less efficient than the execution of original Q_4, since, in this new approach c_3 condition check is done after the join processing, rather than during the join processing.

Strategy 2 : A new SPJ block split rule, nonetheless, can be developed to split the original Q_4^{spj} by deriving it from the join graph corresponding to Q_4. Once the SPJ block of Q_4 is split using this new SPJ block split rule, which extracts the all join paths from the join graph between any two relations, $Q_{4_{priv}}^{spj}$ would be as

$$Q_{4_{priv}}^{spj} = \underset{Cond}{\sigma}(\sigma_{Cond_1}(R) \underset{c_1}{\bowtie} \sigma_{Cond_2}(S) \underset{c_2 \wedge c_3}{\bowtie} \sigma_{Cond_3}(T)) \tag{5.23}$$

where

$Cond : Contains(R.CPT, sens) \vee Contains(S.CPT, sens) \vee Contains(T.CPT, sens)$

$Cond_1 : Contains(R.CPT, sens \mid \{S \underset{c_1}{\bowtie} R, S \underset{c_2}{\bowtie} T \underset{c_3}{\bowtie} R\} \mid \{T \underset{c_3}{\bowtie} R, T \underset{c_2}{\bowtie} S \underset{c_1}{\bowtie} R\})$

$Cond_2 : Contains(S.CPT, sens \mid \{R \underset{c_1}{\bowtie} S, R \underset{c_3}{\bowtie} T \underset{c_2}{\bowtie} S\} \mid \{T \underset{c_2}{\bowtie} S, T \underset{c_3}{\bowtie} R \underset{c_1}{\bowtie} S\})$

$Cond_3 : Contains(T.CPT, sens \mid \{R \underset{c_3}{\bowtie} T, R \underset{c_1}{\bowtie} S \underset{c_2}{\bowtie} T\} \mid \{S \underset{c_2}{\bowtie} T, S \underset{c_1}{\bowtie} R \underset{c_3}{\bowtie} T\})$

Note that this new split rule modifies the conditions on the CPT column in such a way that all the join paths from a relation R to S, such as $R \underset{c_1}{\bowtie} S$ and $R \underset{c_3}{\bowtie} T \underset{c_2}{\bowtie} S$ are checked in S's CPT column. The reason behind this is that, suppose that we have a non-sensitive S record, s_1 such that s_1 joins with a sensitive R record via only path $R \underset{c_1}{\bowtie} S$. Then, when such a cyclic query is split using *split-strategy*, s_1 will have no impact on the answer set of $Q_{4_{priv}}$. Only the non-sensitive S tuples that can join with a sensitive R tuple via all the possible join paths from S to R will impact the $Q_{4_{priv}}$ results. Therefore, this new split rule checks whether S's CPT column contains both paths, instead of one of them, i.e. $Cond_2$ checks where $S.CPT$ contains $R \underset{c_1}{\bowtie} S$ and $R \underset{c_3}{\bowtie} T \underset{c_2}{\bowtie} S$ together.

While *Strategy 2* guarantees that the computation done at the private side $Q_{4_{priv}}^{spj}$ is minimal – that is, only those non-sensitive tuples that can contribute to the answer set at the private side is included. Note that there could be exponential number of join paths between any two relations in a cyclic join graph. Under such a scenario, the resulting selection conditions

associated with the relations on the private side will be significantly complex. Furthermore, the algorithm to generate the corresponding CPT column for all these paths will be very inefficient.

Strategy 3 : One could simplify both the CPT column generation, as well as the SPJ block split rule significantly by considering only the fixed subset (instead of all) of join paths between any two relations in the corresponding join graph. For instance, in the example above, to identify the value of CPT column for R, we may consider only the tuples that match some sensitive records of T through one of the two possible join paths from T to R – viz, $p_1 = T \underset{c_3}{\bowtie} R$, $p_2 = T \underset{c_2}{\bowtie} S \underset{c_1}{\bowtie} R$, but not both. The CPT column for a non-sensitive tuple in R that joins with a sensitive tuple of T is thus marked with the CPT value of the chosen join path. We likewise chose a single join path from T to S to identify non-sensitive tuples of S and mark those as that potentially join with some sensitive tuple of T. Note that the set of non-sensitive R tuples whose CPT contains p_1 is larger than the set of non-sensitive R tuples whose CPT contains both p_1 and p_2.

Now given a cyclic query, one can convert it into the corresponding join graph. A strategy that only selects the minimum length path (instead of all) between any two relations during both CPT column generation and query splitting would be a viable option. For instance, when Q_4 is split using this new strategy, the $Cond_1$, $Cond_2$ and $Cond_3$ will be as

$Cond_1 : Contains(R.CPT, sens \mid S \underset{c_1}{\bowtie} R \mid T \underset{c_3}{\bowtie} R)$

$Cond_2 : Contains(S.CPT, sens \mid R \underset{c_1}{\bowtie} S \mid T \underset{c_2}{\bowtie} S)$

$Cond_3 : Contains(T.CPT, sens \mid R \underset{c_3}{\bowtie} T \mid S \underset{c_2}{\bowtie} T)$.

In *Strategy 3*, the long join paths, such as $T \underset{c_2}{\bowtie} S \underset{c_1}{\bowtie} R$, will not appear in the required set of join paths during CPT column generation, and, in turn, CPT column creation will take less time compared to the *Strategy 2*.

Note that *Strategy 3* ensures that the private side query joins all non-sensitive tuples in any relation that potentially join with at least one sensitive tuple of a different relation in the query. Of course, this strategy may scan other additional tuples as well that may be eliminated (since they do not produce a join result with other tuples from other relations where one of those tuples is sensitive). Nonetheless, *Strategy 3* is simple and does not result in a predicate blowup as is the case with *Strategy 2*.

Nested Queries : Nested queries can be considered as a nested single block queries with an arbitrary depth, where the blocks might be correlated or uncorrelated with each other.

To give an example, let t

Q_1 = `SELECT A.id FROM A WHERE A.name IN`
`( SELECT B.name FROM B)`
Q_2 = `SELECT A.id FROM A WHERE A.name IN`
`( SELECT B.name FROM B WHERE A.id = B.id)` .

The blocks in Q_1 are uncorrelated, whereas the blocks in Q_2 are correlated with condition $A.id = B.id$.

First, if the blocks are uncorrelated with each other in a nested query, one can always split the first (or inner block) in that query using our single-block *split-strategy*, explained in the previous section, and then compute the subsequent (or outer blocks) as a part of the merge query. Note that this might be a viable solution for Hive queries, since HiveQL does not support correlated nested queries as of now.

Additionally, some types of nested queries can be converted to a more efficient single-block queries using the techniques in [64, 65]. So, one can transform such nested queries in the workload to a single-block one and again apply our single block *split-strategy*.

For the nested multi-block queries which cannot be converted to a single-block query, if the

blocks are correlated with each other, then we unfortunately end up running the entire query on the private cloud.

Outer Joins : Outer joins requires a special attention, since they cannot be expressed as a regular join operator. Suppose that we have a query, $Q = R \underset{C}{\rtimes} S$. Given the notation in Section 5.2, one can split this left outer join query similar to $R \underset{C}{\bowtie} S$:

$$Q_{priv} = \sigma_{R.sens \vee S.sens = true}((R_s \cup R^S_{ns}) \underset{C}{\rtimes} (S_s \cup S^R_{ns})) \tag{5.24}$$

$$Q_{pub} = R_{ns} \underset{C}{\rtimes} S_{ns} \tag{5.25}$$

In this strategy, if a non-sensitive record $r \in R$ only joins with a sensitive record s from S, then (r, s) and $(r, null)$ tuples will appear within Q_{priv} and Q_{pub} outputs respectively. Note that, $(r, null)$ is an incorrect output and needs to be eliminated. To achieve this elimination, the database system can tag the primary key (pk) of R tuple to each Q_{priv} and Q_{pub} output. In other words, if a left-outer join output is generated by joining $r_1 \in R$ with $s_1 \in S$ tuple (or none), the output will become like $(r_1.pk, r_1, s_1)$(or $(r_1.pk, r_1, null)$). Once the primary keys of R tuples are appended to the outputs of Q_{priv} and Q_{pub}, the incorrect $(r, null)$ tuple in our example would be eliminated in Q_{merge} by matching its pk with the primary key of (r, s). Note that such a matching operation cannot be implemented using existing relational operators. So, implementing such an operator would be an interesting extension as a future work.

Union : If there exists a union operator inside the query tree, the operators remaining on the left and right hand side of this operator can be considered as independent query blocks. So, our single block splitting strategy can be applied to each of these blocks, if they are in the form of single block acyclic query. Note that each of these single blocks have to be added

to the input workload (W) as an individual query before applying Algorithm 1. Finally, the operators lying on top of union operator will be a part of Q_{merge}

Intersection : Intersection operator can be represented as a combination of join and projection. i.e

$$R \cap S = \prod_{R\ columns} (R \underset{C}{\bowtie} S) \tag{5.26}$$

where C is an equality check between corresponding R and S columns. The intersection operator in the query tree can be replaced by join and projection operator, as it is shown in Equation 5.26. Inside this new query tree, the bottom single-block can be split using again our single block split-strategy. The remaining portion will be considered as the merge query.

Set Difference : Similar to intersections, set difference operator can be replaced by the left outer join, selection and projection operator as

$$R - S = \prod_{R\ columns} (\sigma_{S.tuple=null}(R \underset{C}{\bowtie} S)). \tag{5.27}$$

So, $-$ operator can be replaced by an outer-join, selection and projection operators given in Equation 5.27. In this new query tree, the bottom single-block can be split using again our single block split-strategy. Again, the remaining operators in the query tree will be equivalent to Q_{merge}.

Chapter 6

Partitioning Workloads for Hybrid Clouds

6.1 Introduction

Our work, so far, has shown that hybrid clouds can be used to improve data processing performance while ensuring full-security (no leakage about sensitive data) in the context of MR jobs (SEMROD) and SQL-like query processing (split-strategy). Our results indicated clear performance improvements compared to all private implementations. However, our results also has shown that our frameworks had to pay certain overhead in order to provide security. One important question is that "can these overheads be further reduced?". This would likely to require a framework to push further computation to the public cloud and it is difficult to achieve without willing to sacrifice full security. Organizations might be willing to expose a bounded amount of sensitive data (risk) to the public cloud in order to further reduce their query/workload execution times. Neither our previous approaches nor the existing frameworks allow one to do such a tradeoff between the performance, risk and

cost of using public cloud resources.

In this chapter, we designed a risk and cost aware conceptual framework, called as *Hybridizer*, whose goal is to partition an application's query workload, and in turn, its data, over a hybrid cloud in order to maximize the workload performance while meeting user's cost and risk requirements. Hybridizer primarily addresses the *Workload Distribution Problem (WDP)* over a hybrid cloud based on the end-user performance, risk and cost requirements. Hybridizer is essentially designed as a proof of that risk can help to gain further performance.

Different data and workload partitionings display different trade offs in terms of the workload's running time (performance), the monetary cost of acquiring resources as a public cloud (cost) and sensitive data disclosure risk (risk) metrics. For instance, storing all the data and executing the given workload only on private machines leads to high-security, no sensitive data disclosure risks, and no outsourcing costs, but may incur low performance due to limited private resources. Alternatively, shifting the entire data and computation to the public cloud may have a high performance benefit, but at the same time incurs the maximum risk and monetary costs. Hybridizer's goal is to adjust the data and workload that will be outsourced to the public cloud based on :

- minimizing the workload execution time,
- user's risk and cost requirements.

We demonstrate the power of Hybridizer by showing how it can be used to achieve the data and workload partitioning in a scenario, wherein security-conscious organizations use private clouds to support mission-critical tasks, and use public clouds to deploy routine, analysis-oriented tasks.

We begin this chapter by formalizing the workload and data distribution problem (WDP) in Section 6.2. Then we describe the Hybridizer's architecture in Section 6.3, Next, Section 6.4

explains how the statistics computation and the metric estimation can conceptually be done in Hybridizer in order to solve WDP. In Section 6.5, we discuss our dynamic programming based approach to solve WDP[1]. Then, we evaluate our solution approach over a variety of WDP parameter settings in Section 6.6. Finally, we conclude the chapter and discuss extensions to further improve Hybridizer framework in Section 6.7.

6.2 WDP Definition

In this section, we present the formal definition of WDP. Before we formalize WDP, we introduce some notations in Table 6.1 that are used throughout this chapter. We denote by W, the given workload and by R, the dataset used by the workload.

In general, the workload distribution problem aims to find $W_{pub} \subseteq W$ and $R_{pub} \subseteq R$ such that the workload execution time, i.e. performance, is minimized while constraining the monetary cost and sensitive data disclosure risk. WDP can be reduced to finding only the optimal value for W_{pub}. Since with given W_{pub}, the corresponding subset $R_{pub} \subseteq R$ can be computed as $\bigcup_{q \in W_{pub}} data(q)$, where $data(q)$ is the minimum data needed to evaluate query q. Thus, R_{pub} corresponds to the minimum data required to execute the set of queries in W_{pub}. Additionally, shifting more data to the public cloud than needed will incur higher monetary cost as well as higher data disclosure risk if the extra part, $R_{pub} - \bigcup_{q \in W_{pub}} data(q)$, contains some sensitive data. Due to this higher service expense and data exposure risk, the solution $(\bigcup_{q \in W_{pub}} data(q), W_{pub})$ is always a better solution than (R_{pub}, W_{pub}) for WDP. To summarize, finding the optimal (R_{pub}, W_{pub}) is equivalent to finding only the optimal W_{pub}.

WDP **Definition**: Given dataset R and workload W, WDP could be formulated as:

[1]We develop a dynamic programming solution since, as discussed in Section 6.5, a solution based on mathematical programming optimizers such as CPLEX is not suitable for solving the resulting optimization problem

Notation	Description
R	Dataset to be partitioned over a hybrid cloud.
W	The set of queries (workload) to be split over a hybrid cloud. (i.e. $W = \{q_1, q_2, \ldots, q_n\}$ where q_i is a query in the workload).
$OverallTime(W, W^{'})$	Overall running time of processing W over a hybrid cloud, given that only $W^{'} \subseteq W$ is processed by a public cloud.
$Risk(R^{'})$	Risk associated with storing data items from $R^{'} \subseteq R$ on a public cloud.
$Cost(R^{'}, W^{'})$	The monetary cost of using public cloud services for storing $R^{'}$ and processing $W^{'}$ on the public cloud.
$data(q)$	The minimum set of data items for executing a query $q \in W$.
$time_x(q)$	The estimated running time of query $q \in W$ on either public ($x = pub$) or private ($x = priv$) clouds.

Table 6.1: Notations

$$\begin{aligned} &\text{minimize} \qquad OverallTime(W, W_{pub}) \\ &\text{subject to} \quad Cost(\bigcup_{t \in W_{pub}} data(t), W_{pub}) \leq \Delta_{cost} \ (1) \\ &\qquad\qquad\quad Risk(\bigcup_{t \in W_{pub}} data(t)) \leq \Delta_{risk} \ (2) \end{aligned} \tag{6.1}$$

where Δ_{cost} and Δ_{risk} denote the maximum permissible public cloud monetary cost and sensitive data disclosure risk. Additionally, constraints (1) and (2) set an upper limit for the monetary cost and disclosure risk respectively.

6.3 Hybridizer

In this section, we describe our conceptual Hybridizer framework, that, automatically distributes data and workload of queries based on the user provided cost and risk limits.

Before going into the details, we assume that the entire dataset is initially placed on the private cloud (as in previous chapter) and the dataset is given to the Hybridizer to partition.

6.3.1 Hybridizer Architecture

Figure 6.1 presents an architectural overview of Hybridizer which consists of the following core conceptual modules:

Statistics Creator accumulates two "statistics" about the given dataset R and workload W:

- $time_x(q)$**:** The estimated running times of each query $q \in W$ on either side of the cloud ($x = priv$ or $x = pub$),
- $data(q)$**:** The estimated minimum set of data items to process query $q \in W$.

These statistics must be efficiently represented and maintained by statistics creator so that **metric estimator** component can efficiently use them to be able to measure performance, data disclosure risk and service expense metrics for any given candidate WDP solution.

Metric Estimator can compute the performance, risk and cost metric values by using the created statistics for a given data and workload partitioning. The details of how metric estimation can be done using the existing statistics will be given in next section.

WDP **Solver** can automatically determine which parts of R will be stored in private and

public sides (i.e. R_{priv} and R_{pub}) and which set of queries in W will be processed over private and public sides of the hybrid cloud (i.e. W_{priv} and W_{pub}). In solving WDP, it interacts with the metrics estimator to check whether the candidate R_{pub} and W_{pub} solutions are optimal and whether they violate the user-defined constraints Δ.

Physical Distributor can distribute R and W across a hybrid cloud, based on the solution produced by WDP solver. It might push the data partitions, R_{pub} and R_{priv}, accordingly and then forwards the corresponding set of queries to the workload executor in private and public cloud. This workload executor might be any parallel database management tool.

Hybridizer architecture that we introduced in Figure 6.1 is a conceptual framework. To instantiate Hybridizer, we need to specify the techniques to create $data()$ and $time_x()$. Statistics creation directly depends upon two key factors, viz. what kind of workload is going to be executed (workload model) and what is the format of the data that is used by the workload format (data model). The way the data is represented (relational, RDF, XML or textual etc.) directly impacts the $data()$ estimation for the workload's queries. Namely, for each different data model, an appropriate strategy has to be designed to determine and maintain *data* statistics. Additionally, the workload model (such as SQL queries, MR jobs, XML queries etc.) has a direct impact on the technique that will be used to measure $time_x$ for each query. For each workload model, estimation of the query running times would be quite different. In this chapter, we will give details about how the statistics and metrics can be calculated in the context of relational data and SQL-like workload model.

6.4 Statistics Creation and Metric Estimation

Estimating performance, disclosure risk and cost metrics as accurately as possible is a fundamental challenge towards solving WDP. This section elaborates how Hybridizer can create

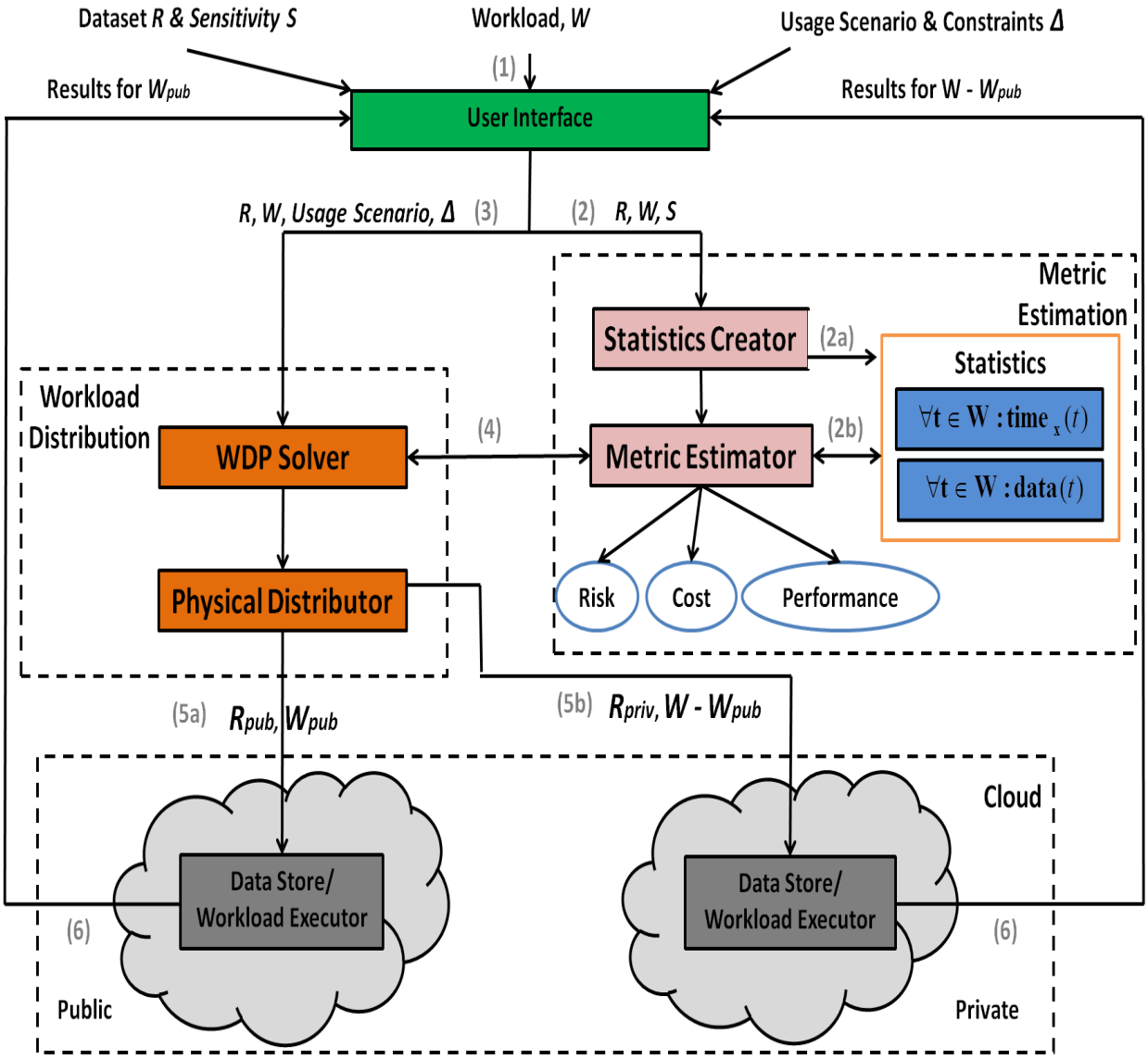

Figure 6.1: Hybridizer Architecture

its basic statistics and how the metric values can be measured based on these basic statistics.

6.4.1 Statistics Creation

As mentioned earlier, to accurately estimate the performance, risk and cost metrics, Hybridizer's statistics creator component needs to create two basic statistics for all the queries in the workload W, $data(q)$ and $time_x(q)$. However, these statistics need to be calculated differently for each data/workload model variant. Moreover, $data(q)$ has to be stored in a memory efficient format since maintaining $data(q)$ as a complete set of individual data items will not be scalable, in particular when the number of data items is large. Therefore, different dataset representation strategies must be explored to efficiently represent $data(q)$.

6.4.1.1 data(q)

As we mentioned before, identifying the minimum set of required data items and maintaining them in an efficient format is a challenging task. Moreover, the technique to achieve this task will be unique for each data/workload model.

When the data is relational and the query q is a SQL query, $data(q)$ would be a set of single table views each of which represents the minimum set of attributes and rows that is touched by query q in that particular table.

To compute these views for SQL queries, statistics creator module can create an operator tree for each query in the workload by pushing the selection and projection operators as far down in the original operators trees as possible using the relational algebra rules. After this push down, the union of projection and selection operators on top of any single table T in this new operator tree represents the smallest set of data items coming from T to answer that particular query. For instance, suppose that the query q is represented as

```
SELECT l_orderkey, o_orderdate FROM customer, orders, lineitem
WHERE c_mktsegment = '[SEGMENT]' and c_custkey = o_custkey
and l_orderkey = o_orderkey and l_shipdate > '01/01/1996'
and o_orderdate < '01/01/1996'
GROUP BY l_orderkey, o_orderdate
ORDER BY o_orderdate.
```

The data items referred by the following single relation (table) view definitions ($V1$, $V2$, $V3$) are extracted from q by pushing down the selections and projections as much as possible:

```
V1: SELECT c_custkey
FROM customer WHERE c_mktsegment = '[SEGMENT]'
V2: SELECT o_orderdate, o_custkey, o_orderkey
FROM orders WHERE o_orderdate < '01/01/1996'
V3: SELECT l_orderkey
FROM lineitem WHERE l_shipdate > '01/01/1996'
```

After extracting these view definitions from the workload queries, they can be efficiently maintained in the memory using an array in which the i-th bucket stores to the set of views generated from the i-th query in the given SQL workload. The set of views in each array bucket can be ordered by the table name and each view object has the following fields: table name, list of the projected attribute names and a predicate object. Each predicate object p can be represented as the disjunction of conjunctive predicates, i.e. $p = p_1 \vee p_2 \vee \ldots \vee p_n$, where each p_i is a conjunction of simple predicates, $p_i = p_{i_1} \wedge p_{i_2} \wedge \ldots \wedge p_{i_n}$. Finally, each simple predicate comprises of a single attribute and a simple condition on it, such as > 45 or $=' jack'$.

6.4.1.2 $\mathbf{time_x(q)}$

For a given query q, $time_x(q)$ can be calculated by estimating the amount of I/O incurred by q. For instance, such an I/O based approach has been used to evaluate the running times of queries and to assess the performance of MapReduce jobs [78]. The time required to process a query q can be estimated as:

$$time_x(q) = \frac{\sum_{\forall\ operator\ \rho \in q} inpSize(\rho) + outSize(\rho)}{w_x} \tag{6.2}$$

where $inpSize(\rho)$ and $outSize(\rho)$ are the estimated input and output sizes of an operator $\rho \in q$. Additionally, w_x denotes the processing speed of private or public cloud, namely the no. of I/O operations that can be performed per unit time at site x. For an operator ρ, $inpSize(\rho)$ and $outSize(\rho)$ can be computed using certain histograms over R.

If the data is relational and the workload is a set of SQL queries, then an operator ρ would be either join, selection, aggregation or projection, and the histograms can be constructed as equi-width statistics on each attribute of the relations.

Once these histograms are created, input/output sizes for each operator in the query tree can be estimated by applying the well-known techniques used in standard databases. For example, the output size of a simple selection operator, $\rho^{'}$ can be computed by multiplying the selectivity of the predicate in $\rho^{'}$ with its input size $inpSize(\rho)$. Note that the selectivity of a predicate can be easily obtained by iterating over overlapping equi-width histogram intervals.

6.4.2 Metric Estimation

In this subsection, we describe the metrics used in the formalization of WDP. In particular, each metric is defined in a way that captures high-level aspects of the metric relevant to hybrid clouds. This definition needs to be further refined based on a specified Data/Workload-/Sensitivity model. As an example, we present metric definitions when the Data/Workload models are based on a relational model/Hive queries.

Performance: The performance metric can be expressed as the overall running time of the entire workload in Hive across the hybrid cloud (i.e. $(OverallTime(W, W_{pub}))$). In general for any given arbitrary W_x and W_y set of Hive queries, $OverallTime$ function can be calculated as

$$OverallTime(W_x, W_y) = \max \begin{cases} \sum\limits_{q \in W_y} time_{pub}(q) \\ \sum\limits_{q \in W_x - W_y} time_{priv}(q) \end{cases} \tag{6.3}$$

. Note that, in our scenario, $W_x = W$ and $W_y = W_{pub}$.

Monetary Cost: Since public cloud services are continually used, they are included in the operational expenditure, which could be curtailed by restricting the data/processing outsourced to public clouds. In our approach, we followed an elastic cost model, viz. the user pays to the cloud provider as they use their resource. Thereby, the cost metric can be estimated as follows:

$$Cost(R_{pub}, W_{pub}) = store(R_{pub}) + \sum_{q \in W_{pub}} proc(q) \tag{6.4}$$

where $store(R_{pub})$ and $proc(q)$ denote the cost of storing $R_{pub}(i.e \bigcup\limits_{q \in W_{pub}} data(q))$ and processing query $q \in W_{pub}$ on the public cloud. Note that, $proc(q)$ will be directly proportional

to the running time of query q at public side ($time_{pub}(q)$), since the cloud provider charges its customers based on pay-as-you-use pricing model. In such case, the cost of processing a query q can be estimated as

$$proc(q) = \alpha \times time_{pub}(q) \tag{6.5}$$

where α is the average amount of money that cloud provider charges per unit time.

Sensitive Data Disclosure Risk: Disclosure risk of outsourcing $R_{pub} = \bigcup_{q \in W_{pub}} data(q)$ can be proportional to the amount of sensitive data included in R_{pub}. The risk of R_{pub} can be measured as follows:

$$Risk(R_{pub}) = \sum_{r_i \in R_{pub}} risk(r_i) \tag{6.6}$$

where r_i is an atomic data item. For instance, r_i is a tuple/record in relational dataset.

The risk of exposure associated with the data item r_i itself depends upon various factors including the degree of sensitivity associated with the data, the underlying representation of r_i on public machines (e.g. encrypted / plain text and if encrypted, the encryption mechanism used).

While Hybridizer framework can allow any risk function to be associated with the data items (table tuples), we will use a simple model in the remainder of the chapter wherein exposure of a sensitive record incurs a risk of a single unit. Thus, the risk of R_{pub} is the count of the sensitive records in R_{pub}.

6.5 WDP SOLUTION

The goal of any solution to WDP should be to find an optimum division of dataset R and workload W, (R_{pub}, W_{pub}). As we explained in Section 6.2, any WDP solver is in fact only required to find a subset $W_{pub} \subseteq W$, since the corresponding subset $R_{pub} \subseteq R$ can be directly obtained by taking the union of $data(q)$ where $q \in W_{pub}$ (i.e. $\bigcup_{q \in W_{pub}} data(q)$).

In this section, we first describe why traditional integer programming optimizers such as IBM's CPLEX optimizer are unable to model the WDP accurately. Then, we will subsequently present our efficient dynamic programming algorithm to solve WDP.

6.5.1 Solving WDP with Integer Programming

In WDP, our goal is to minimize overall running time while meeting given risk and cost requirements. In fact, associated time, risk and cost integer values can be computed for each query q in the workload, as if the query q is the only query moved to the public cloud. After obtaining these set of integer values for each query, one can formulate the WDP as an integer programming problem (a mathematical optimization program in which some or all the variables are restricted to be integer type) and, in turn, one can use efficient integer programming solutions such as IBM's CPLEX optimizer to solve WDP.

However, integer programming falls short in situations where one needs to compute exact storage costs associated with public cloud queries, *viz.* $store(\bigcup_{q \in W_{pub}} data(q))$ as well as the data disclosure risk, *viz.* $Risk(\bigcup_{q \in W_{pub}} data(q))$. The reason for this shortcoming is that integer programming is simply unable to accurately model the problem in this situation. For example, suppose that the queries q_1 and q_2 use the same subset of dataset R, that is R_1. To shift either of q_1 or q_2 to the public side, R_1 must be stored in the public cloud. Therefore, shifting both queries will not bring any extra storage cost for the same subset of R_1.

When there is a large number of queries in the workload, formulating such conditions in integer programming can be a time consuming job, especially when each data item is commonly used by many queries in the workload. One could create a conditional integer (0 or 1) for each record r in R, indicating whether any of the queries that uses this particular r is assigned to the public cloud, and multiply it with the r's storage cost. The actual storage cost after workload distribution can then be computed by taking the sum of each multiplication. However, the complexity of such a formula cannot be managed by any integer programming solution when the dataset, R, has billions of data items (or tuples). A practical demonstration of this phenomenon can be seen in the experiments given in Section 6.6.

6.5.2 Dynamic Programming Solution

Given the exponential number of workload subsets, WDP in general is NP-Hard as the 0-1 Knapsack Problem can be reduced to the WDP. Therefore, we developed a dynamic programming heuristic to find the best W_{pub} for WDP. We now present our dynamic programming algorithm that produces the W_{pub} as a solution to the WDP. To represent WDP along with its inputs and constraints, we use the following notation: $WDP(W, \Delta_{cost}, \Delta_{risk})$ where, Δ_{cost} and Δ_{risk} are the corresponding sensitive monetary cost and data disclosure risk constraints in that WDP. Each of these constraints denotes an upper limit for one of the metrics.

To make it accessible for readers, we first provide the intuition behind our dynamic programming heuristic for WDP. While creating a dynamic programming solution, usually the first thing to do is to define how the given large problem can be split into several small sub-problems. Intuitively, this is a simple query in our case. Because, to distribute W across the hybrid cloud, we have to individually decide where to send each query. Now, there are two possible assignments for the last query q_n in W. The query q_n can run on either the

private cloud (*private case*) or public cloud (*public case*). Therefore, both cases should be investigated carefully while solving each WDP. Such a dividing technique comprises the basis of our all dynamic programming solutions.

In WDP, *private case* will not bring any risk or cost to the workload distributer. Thereby, the given Δ_{cost} cost and Δ_{risk} risk limits can be completely used to distribute $W - q_n$. If q_n runs on the public side (*public case*), then there will be more than one WDP sub-problems that need to be investigated. This is due to the fact that the possible execution of q_n on the public side will cost at least $proc(q_n)$ and at most $Cost(q_n)$ amount of money, depending on how the remaining queries are distributed. In terms of disclosure risk, these numbers will be between 0 and $Risk(data(q_n))$. Again, each solution that has been generated by a WDP sub-problem has to be tested to ensure that it does satisfy all the constraints and it is the best solution in terms of the overall workload time amongst the others. Consequently, $WDP(W, \Delta_{cost}, \Delta_{risk})$ will be equal to:

$$min\_time \begin{cases} WDP(W - q_n, \Delta_{cost}, \Delta_{risk}) \textbf{ (private case)} \\ \\ WDP(W - q_n, \Delta_{cost} - x, \Delta_{risk} - y) \cup q_n \\ proc(q_n) \leq x \leq Cost(q_n) \text{ and where } 0 \leq y \leq Risk(data(q_n)) \textbf{(public case).} \end{cases}$$

6.5.2.1 Algorithm

Here we give the details of our dynamic programming algorithm.

Algorithm 1 uses a data structure $pubW$ and frequently calls a method labeled as $checkConstr$. The purpose of these constructs is as follows:

Input: W, Δ_{cost}, Δ_{risk}
Output: W_{pub}

```
1  Initialize pubW[][][]
2  for i = 1 → W.size do
3      procCost ← proc(q_i) ;
4      totCost ← procCost + store(data(q_i)) ;
5      disc ← sens(data(q_i)) ;
6      for j = 0 → Δ_cost do
7          for k = 0 → Δ_risk do
8              if i = 1 then
9                  if checkConstr({t_1}, j, k) & OverallTime(W^1, W^1) <
                   OverallTime(W^1, ∅) then
10                     pubW[i][j][k] ← {t_1}
11                 else
12                     pubW[i][j][k] ← ∅
13             else
14                 pubCaseTime ← ∞
15                 (j', k') ← (NaN, NaN)
16                 if checkConstr({q_i}, j, k) then
17                     foreach j − totCost ≤ iC ≤ j − procCost do
18                         foreach k − disc ≤ iD ≤ k do
19                             tmpSet ← pubW[i][iC][iD] ∪ q_i
20                             if
                               checkConst(tmpSet, iC, iD) & OverallTime(W^i, tmpSet) <
                               pubCaseTime then
21                                 pubCaseTime ← OverallTime(Q^i, tmpSet)
22                                 (j', k') ← (iC, iD)
23                         end
24                     end
25                 privCaseTime ← Overalltime(W^i, pub[i − 1][j][k])
26                 if privCaseTime ≤ pubCaseTime then
27                     pubW[i][j][k] ← pubW[i − 1][j][k]
28                 else
29                     pubW[i][j][k] ← pubW[i − 1][j'][k'] ∪ {q_i}
30         end
31     end
32 end
33 return pubW[W.size][Δ_cost][Δ_risk]
```

Algorithm 3: Dynamic Programming Algorithm for WDP

Query	$proc(q)$	$store(data(q))$	$sens(data(q))$
q_1	\$10	\$15	20
q_2	\$20	\$10	10
q_3	\$15	\$10	20

Table 6.2: Example Query Set

- **pubW[i][j][k]**: This data structure maintains the solution set for $WDP(W^i, j, k)$ where $W^i = \{q_1, t_2, \ldots, q_i\}$. Given that the maximum admissible monetary cost and the maximum disclosure risk are equal to j and k respectively, this data structure stores the ones from amongst the first i queries that are selected to be processed over the public cloud so as to minimize the overall response time of the first i queries. Notice that $pubW[i][j][k] \subseteq W^i$.

- **checkConstr($\mathbf{W}^{'}, \mathbf{j}^{'}, \mathbf{k}^{'}$)**: This method returns whether monetary cost bound $j^{'}$ and disclosure risk limit $k^{'}$ are satisfied when the queries in $W^{'}$ are executed on the public side. In particular, the method checks if $store(\bigcup_{t \in W^{'}} data(t)) + \sum_{t \in W^{'}} (freq(t) \times proc(t)) \leq j^{'}$ and $sens(\bigcup_{t \in W^{'}} data(t)) \leq k^{'}$.

To make it easily understandable for readers, let us illustrate how our algorithm works with an example. Assume that our workload W consists of 3 queries (i.e. $W = q_1, q_2, q_3$) and $WDP(W^3, j, k)$ needs to be solved. The detailed information about these 3 queries is given below.

Before investigating the two different cases in further details, we need to check whether assigning q_3 to the public side violates any constraints (line 16). If we ship q_3 to the public side, then the overall monetary cost and the overall disclosure risk will be at least \$25 and 20 sensitive records respectively (assume that $\forall 1 \leq i \leq 3\ freq(i) = 1$). If $j < 25$ or $k < 20$, then any solution considering q_3 as a public side query will not be a feasible one, and in turn $WDP(W^3, j, k) = WDP(W^2, j, k)$ (line 25-27). Note that, since executing any query on the private side does not cause a violation of any constraint, this case essentially does not

require a feasibility analysis. Now, we can go into the details of each case.

Public Case: If q_3 runs on the public side, then there will be more than 1 WDP sub-problems that need to be investigated. This is due to the fact that the possible execution of q_3 on the public side will bring at least \$15 and at most \$25 into the overall monetary cost value. In terms of disclosure risk, the numbers will be between 0 and 20 sensitive records. The reason is that a portion of (or the entire) $data(q_3)$ could already be included in the solution of $WDP(W^2, j^{'}, k^{'})$, W^2_{pub} and in turn storing $data(q_3)$ in addition to $\bigcup_{q \in W^2_{pub}} data(q)$ may not bring as much monetary cost and disclosure risk as is represented in the table above. Consequently, $WDP(W^2, j^{'}, k^{'})$ where $j - 25 \leq j^{'} \leq j - 15$ and $k - 20 \leq k^{'} \leq k$ should be investigated in order to solve $WDP(W^3, j, k)$ optimally (lines 17-24). However, every candidate set of queries formed by taking the union of q_3 with the solution of $WDP(W^2, j^{'}, k^{'})$ should be tested to ensure that it does not violate any constraint and it is the best solution in terms of performance among the all solutions obtained in *public case*(line 20-22).

Private Case: In case query q_3 runs on the private side, then $WDP(W^3, i, j) = WDP(W^2, i, j)$ (line 25).

After computing the best solution candidate for both cases, our algorithm compares the overall expected running times of both solutions and picks the minimum one as the solution to $WDP(Q^3, j, k)$ (lines 26-29).

6.6 Evaluation

This section presents the results to validate the effectiveness of our algorithmic solutions to WDP. We first create the statistics about a certain dataset and workload, and then we executed our dynamic programming approach based on our statistics. We estimated the workload execution times of the partitionings generated by our dynamic programming

solution and presented these estimated workload execution times.

6.6.1 Setup

Dataset: We identified our dataset as the TPC-H dataset with scale factor 100 ($\approx 100GB$) [79]. We gathered statistics by analyzing this 100GB TPC-H dataset and generated equi-width histograms for every attribute in TPC-H.

While creating equi-width histogram in TPC-H dataset, we used the data types, int, double and string in SQL to represent TPC-H data. We also created a data type 'date' that allows us to represent various dates from the TPC-H schema. The number of partitions used in a histogram is dependent on the data type; this number is fixed for a given data type: (i) For integers and doubles, the number of partitions = $log_2(max - min)$, where min and max represent the min and max domain values mandated by TPC-H for that particular attribute. (ii) For dates, since TPC-H only allows dates between '1992-01-01' and '1998-12-31', we created one partition for each year from 1992 through 1998. (iii) For strings, we created 95 partitions that cover alphabets ($a - z$ and $A - Z$), digits ($0 - 9$) and all special characters (!, @, #, *etc.*). Additionally, these histograms maintain a set of numbers such as the average length of an attribute value and the number of unique attribute values within each equi-width intervals.

After constructing required histograms, we estimated $data()$ and $time_x()$ statistics for each query in our workload, as we explained in Section 6.3.

Workload: We defined our workload as a workload of 40 queries containing modified versions of queries Q1, Q3, Q6 and Q11 in TPC-H benchmark. In particular, we modified these queries as no grouping and aggregate operations were performed in them, due to the high complexity of estimating I/O sizes for such operators in Hive. Recall that, query *time* statistics differs

for the private and public cloud, since each cloud can perform different amount of I/O operations per unit time, w_{pub} or w_{priv} .

w_x is calculated as the number of I/O operations performed per second on the public and private clouds. To compute w_{priv} and w_{pub}, we created a private and public cloud. Our private cloud consists of 14 nodes, while our public cloud consists of 38 nodes. A private cloud node had a Pentium IV processor with $\approx$ 290GB-320GB disk space and 4GB of main memory, while a public cloud node have an AMD Dual-Core processor with $\approx$ 631GB disk space and 8GB of main memory. We set up Hadoop v1.0.4 and Hive v0.7.1 [80] on both clouds. We estimated benchmark-specific weights w_{pub} and w_{priv} by running all the TPC-H benchmark using Hive and Hadoop both on our private and public cloud. w_{pub} (resp. w_{priv}) was computed as the average ratio of I/O operations required by public queries (resp. private) to the total time required to run all queries on the public side (resp. private side). We executed all TPC-H queries on a 300GB dataset and estimated w_{pub}/w_{priv} to be $\approx$ 40MB/sec and $\approx$ 8MB/sec. A larger w_{pub} value indicates that the public cloud has a higher I/O throughput than the private cloud.

Sensitive Data Definition: In our 100GB test dataset, the sensitive data is distributed in two different ways: (1) *Random* sensitivity implies that all the sensitive data is uniformly distributed inside the dataset (2) *Fixed* sensitivity distribution uses a view-based model [81, 82] to declare the sensitive portion of the dataset.

In *random* sensitivity distribution, if sensitivity level is 1%, then we first assume that 1% of the records in the entire dataset is sensitive. Additionally, we assume that 1% of each query's records are sensitive, since the sensitivity is uniformly distributed.

In *fixed* sensitivity distribution, we declare that all the records of the *customer* table are sensitive while fractions of tuples in the *lineitem* table are sensitive ($\approx$ 1% or 10% of the table is marked as sensitive). As a result, we used two different sensitivity levels for each

distribution models in our experiments: 1% and 10%.

Cost Limit: The cost of storing data and executing a workload was estimated using unit prices from Amazon Web Services. We used S3 pricing to determine storage ($0.140/GB + PUT) and communication ($0.120/GB + GET) costs, where the price for PUT and GET operations is $0.01/1000 requests and $0.01/10000 requests respectively. We used EC2 and EMR pricing to calculate the processing cost ($0.085 + $0.015 = $0.1/hour). Finally, we estimated the maximum public cloud cost, MAX_COST, by assuming that our entire test dataset/workload is migrated to the public side. We estimated that maximum cost that one can pay to run our workload is equal tio $\approx 25K$ .While evaluating our dynamic programming approach, we defined Δ_{cost} as a fraction of MAX_COST, *viz.* 25%, 50%, 75% and 100%. So, when the cost limit = 50% means that the actual Δ_{cost} is set to $\approx$ $12250 in our dynamic programming.

Risk Limit: We defined the risk limits as a fraction of the exposed sensitive records to overall sensitive records. To illustrate, when the sensitivity distribution is *fixed* and the sensitivity level is 1%, we estimated that there exists 63 million sensitive records within our dataset. $\Delta_{risk} = 10\%$ means that our dynamic programming approach is allowed to only expose 10% of 63 million sensitive records to public cloud. So, in that case our dynamic approach can at most expose 6.3 million sensitive records to the public cloud.

6.6.2 Evaluation of Our Solution

In this subsection, we present the estimated running time of the workload when it is split using our dynamic programming approach under various cost and risk limits and different sensitivity settings.

Since we aim to minimize overall workload execution time, we first estimated the workload

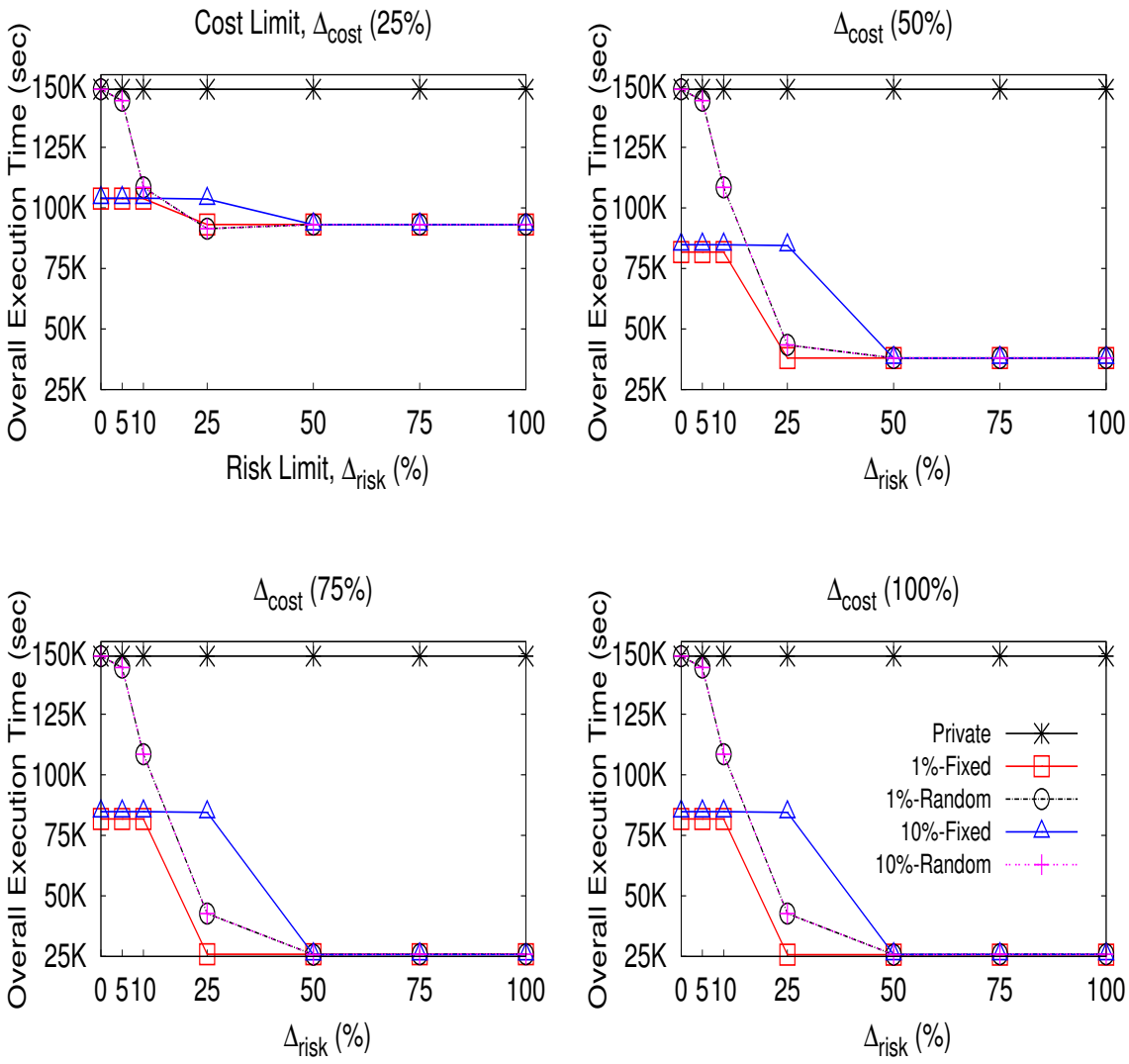

Figure 6.2: Hybridizer Results for WDP in TPC-H Workload

running time when all the computation is performed on the private cloud (*Private*). This case was marked as a baseline to compare the estimated performance of our solution in different sensitivity distributions and levels. We also varied the parameters as follows: First, the monetary expense limit (Δ_{cost}) was varied between 25-100% of MAX_COST. For each different Δ_{cost}, we set seven risk levels as 0%, 5%, 10%, 25%, 50%, 75% and 100%.

We then estimated the overall performance of the query workload for different combinations of these three parameters based on the estimated query running times, the results of which are presented in Figure 6.2. One of the first observations that can be made from Figure 6.2 is that when a user is willing to take additional risks by storing more sensitive data on the public side, they may gain a considerable speed-up in overall execution time based on our estimations (even greater than 50%). On the other hand, Figure 6.2 also shows that the monetary expenditure on public side resources is substantially low even when a user takes additional risks by storing increasing amounts of sensitive data on the public cloud (graphs for $\Delta_{cost} = 50\%$, 75% and 100% show that, even when more money is allowed to be spent on public side resources, the overall performance is relatively the same for these cases suggesting that a budget of only about 50% of the maximum possible cost is sufficient to boost the performance savings up to 50%).

Figure 6.2 also shows that when a user invests more capital towards resource allocation, a considerable gain in overall workload performance (even greater than 50%) can be achieved based on our estimations. This is expected since when more resources are allocated on the public side, we are better able to exploit the parallelism that is afforded by a hybrid cloud. Thus, the intuition that a hybrid cloud improves performance due to greater use of inherent parallelism is justified based on our estimation. Finally, from Figure 6.2, we also notice that we can achieve a considerable improvement in workload performance ($\approx 50\%$) for a relatively low risk ($\approx 40\%$) and cost ($\approx 50\%$) limits.

Moreover, our approach is estimated to finish our experimental workload in ≈ 90000 seconds

at smaller cost limits (close to 25%). On the other hand, when Δ_{cost} reaches to 75%, our solution is expected to execute our workload in almost optimal times $\approx$ 25000 second. To sum up, the intuition that a hybrid cloud improves performance due to greater use of inherent parallelism by means of increasing Δ_{cost} is justified based on our estimations.

6.7 Conclusions and Future Work

Our frameworks, SEMROD and split-strategy, have shown that hybrid clouds can be used to improve data processing performance while ensuring full-security. However, enterprises might be willing to expose a bounded amount of sensitive data (risk) to the public cloud in order to further reduce their query/workload execution times. In this chapter, we designed a risk and cost aware conceptual framework, called as *Hybridizer*, whose goal is to partition an application's query workload, and in turn, its data, over a hybrid cloud in order to maximize the workload performance while meeting user's cost and risk requirements. We proposed a dynamic programming approach to partition given data and workload across the hybrid cloud based on user's requirements. Our results indicated that allowing bounded amount of sensitive data to be leaked to the public cloud may provide further performance improvements while executing the given workload.

We are primarily exploring the following ideas for future research amongst the various areas that we outlined throughout the chapter: 1) In this chapter, we tried to solve the workload partitioning problem for the case where the entire query workload and the input dataset is given to us *a-priori*. This work can be enhanced to support distribution for dynamically changing (or arriving) workloads. 2) We proposed techniques to create statistics and estimate metrics. One can improve this work by actually implementing these modules. 3) We focused on simple risk models based on number of sensitive cells outsourced to the public cloud. Clearly, one may consider different type of risk models suitable for different scenarios. For

instance, in some cases, the sensitivity of the records might be different and organization may not allow to move some sensitive records to the public cloud, if they are not encrypted. In other cases, the association between different columns might be considered as an important dimension of the risk model. We plan to solve the same problem under a different risk models.

Chapter 7

Conclusion and Future Work

Today, large volumes of data are collected and stored by organizations for analysis purposes. Often the in-house computational capabilities of organizations cannot easily support these complex data analytical needs. While such limitations were a serious impediment in the past, emerging public cloud computing platforms (e.g., Amazon's EC2) offer a viable alternative. However, public clouds pose a significant challenge from the perspective of security. According to a survey of IT executives; security, compliance and loss of control are the top 3 concerns for enterprises adopting public clouds [3]. A possible approach to overcome the security challenge is to encrypt the data prior to outsourcing it to the cloud and to perform data analysis over encrypted the data in the cloud. Although past decades of research have made significant progress on developing cryptographic schemes that allow limited computation over encrypted data, no generic and cost-efficient solution for practical use has emerged yet.

An alternate efficient approach to secure data processing is to partition the data and computation across trusted private and untrusted public machines in such a way that the sensitive information is never leaked to the public ones. A *hybrid cloud* is a perfect instantiation of

such a mixed security computation environment. The hybrid cloud paradigm allows end-users to seamlessly integrate their in-house computing resources with public cloud services and construct potent, secure and economical data processing solutions. For instance, hybrid clouds can empower organizations to partition data and computation amongst public and private machines in such a way that they can leverage the power of the public cloud while ensuring that the sensitive data or computation never leaves private machines.

This thesis explores how partitioning data and computation based on data sensitivity can be used to support secure (or risk-aware) yet efficient data processing in hybrid clouds. Under the assumption that the data starts and initially stored at the private cloud, our frameworks steer data and computation through public and private machines in such a way that no (or bounded) knowledge about sensitive data is leaked to public machines in order to process data. For this purpose, we first presented our secure and efficient MapReduce framework for hybrid clouds, SEMROD. Next, we presented a formal strategy, *split-strategy* to process SQL-like structured queries across hybrid clouds in a secure and efficient way. Finally, we focused on data and computation partitioning problem in the level of workloads. We proposed a conceptual framework, Hybridizer, that outsource the data and workload to the public cloud in order to maximize the workload performance without violating user's risk and cost constraints. Instead of only supporting fully secure computation, Hybridizer gives the users a chance to receive a better performance at the expense of exposing some sensitive data to the public cloud. Using our strategies and frameworks, computation that may involve sensitive data can exploit public machines, thereby bringing significant performance benefits, which otherwise would be restricted to only private clouds. Our experiments demonstrate performance advantages of using our approaches as compared to other secure alternatives, even when the percentage of sensitive data is as high as 50%. We envision extending our research in this thesis in following directions:

To our knowledge, none of the existing works including ours tried to leverage encryption as

a part of the hybrid cloud. For cases where we cannot push any sensitive data to public, encryption could be an option. Distributing computation across the hybrid cloud based on both clear text and encrypted data would be an interesting direction of future work.

Additionally, we took the conservative assumption that output of a function on sensitive data is itself sensitive. In many scenarios, application aware sensitivity model may decide that output of a function on sensitive data is non-sensitive. Incorporating such application level sensitivity models into our frameworks and strategies would be an exciting extension.

In this thesis, we focused on hybrid cloud as a secure cloud-computing platform for scalable data processing using MR framework and higher level languages such as Hive. In our scenario, the data originated at the private machines and is selectively replicated to the public cloud. Another common cloud computing use case for organizations is to offload their dynamic transactional workload to the cloud. The usage context where organization use cloud resources to offload their dynamic workloads provide new challenges from the security perspective. Because under such scenarios, it is infeasible to require data to first land on the private cloud. Such a requirement may effectively defeat the purpose of using the cloud to scale to a large number of transactions. Designing new frameworks in hybrid clouds that provide secure and efficient processing for dynamic transactional workloads is a promising extension.

Bibliography

[1] Kerim Yasin Oktay, Sharad Mehrotra, Vaibhav Khadilkar, and Murat Kantarcioglu. Semrod: Secure and efficient mapreduce over hybrid clouds. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 153–166, New York, NY, USA, 2015. ACM.

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

[3] Cloud Survey. `http://cloudtweaks.com/2012/12/cloud-infographic-security-and-the-cloud-2012/`.

[4] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *13th ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. Association for Computing Machinery, Inc., October 2006.

[5] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.

[6] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 965–976, New York, NY, USA, 2012. ACM.

[7] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 44–, Washington, DC, USA, 2000. IEEE Computer Society.

[8] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. *IACR Cryptology ePrint Archive*, 2013:832, 2013.

[9] Peter Van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. Computationally efficient searchable symmetric encryption. In *Proceedings of the 7th VLDB Conference on Secure Data Management*, SDM'10, pages 87–100, Berlin, Heidelberg, 2010. Springer-Verlag.

[10] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 350–364, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 563–574, New York, NY, USA, 2004. ACM.

[12] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, pages 578–595, Berlin, Heidelberg, 2011. Springer-Verlag.

[13] Charalampos Mavroforakis, Nathan Chenette, Adam O'Neill, George Kollios, and Ran Canetti. Modular order-preserving encryption, revisited. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 763–777, New York, NY, USA, 2015. ACM.

[14] Alexandra Boldyreva, Serge Fehr, and Adam ONeill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In David Wagner, editor, *Advances in Cryptology CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 335–359. Springer Berlin Heidelberg, 2008.

[15] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[16] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In GeorgeRobert Blakley and David Chaum, editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer Berlin Heidelberg, 1985.

[17] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *IN ADVANCES IN CRYPTOLOGY EUROCRYPT 1999*, pages 223–238. Springer-Verlag, 1999.

[18] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[19] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer Berlin Heidelberg, 2005.

[20] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM*

SIGMOD International Conference on Management of Data, SIGMOD '02, pages 216–227, New York, NY, USA, 2002. ACM.

[21] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramaratnam Venkatesan. Orthogonal security with cipherbase. In *6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*, January 2013.

[22] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, September 2012.

[23] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 289–300. VLDB Endowment, 2013.

[24] Google BigQuery. `https://cloud.google.com/bigquery/`.

[25] Jeremy Kepner, Vijay Gadepally, Peter Michaleas, Nabil Schear, Mayank Varia, Arkady Yerukhimovich, and Robert K. Cunningham. Computing on masked data: a high performance method for improving big data veracity. *CoRR*, abs/1406.5751, 2014.

[26] MS researchers claim to crack encrypted database with old simple trick. `http://arstechnica.com/security/2015/09/ms-researchers-claim-to-crack-encrypted-database-with-old-simple-trick/`.

[27] Muhammed Naveed, Seny Kamara, and Charles Wright. Inference attacks on property-preserving encrypted databases. CCS '15. ACM, 2015.

[28] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[29] Arvind Arasu, Ken Eguro, Raghav Kaushik, and Ravi Ramamurthy. In *29th International Conference on Data Engineering (ICDE)*, April. Tutorial presentation.

[30] Arvind Arasu, Ken Eguro, Raghav Kaushik, and Ravi Ramamurthy. When is an encrypted database secure? Technical Report MSR-TR-2014-133, September 2014.

[31] Hybrid Cloud. The NIST Definition of Cloud Computing. *NIST, Special Publication, 800-145*, 2011.

[32] *The Beckman Report on Database Research*, The Beckman Database Research Self-Assessment Meeting, 2013.

[33] Hybrid Cloud Usage. `http://venturebeat.com/2013/10/09/hybrid-cloud-year/`.

[34] Hive query language, hiveql,. `https://cwiki.apache.org/confluence/display/Hive/LanguageManual`.

[35] Pig query language, pig latin. http://pig.apache.org/docs/r0.7.0/piglatin_ref1.html.

[36] Bijit Hore, Sharad Mehrotra, and Hakan Hacigumus. Managing and querying encrypted data. In Michael Gertz and Sushil Jajodia, editors, *Handbook of Database Security*, pages 163–190. Springer US, 2008.

[37] C. Gentry. *Building Practical Systems That Compute on Encrypted Data*. PhD thesis, Massachusetts Institute of Technology, 2014. http://www.eecs.berkeley.edu/~raluca/Thesis.pdf.

[38] Christoph Bosch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Comput. Surv.*, 47(2):18:1–18:51, August 2014.

[39] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1998.

[40] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 720–731. VLDB Endowment, 2004.

[41] Alberto Ceselli, Ernesto Damiani, Sabrina De Capitani Di Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM Trans. Inf. Syst. Secur.*, 8(1):119–152, February 2005.

[42] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption*, pages 191–204. Springer, 1994.

[43] C. Curino, E. P. C. 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 *CIDR*, pages 235–241, 2011.

[44] 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 *In Proc. CIDR*, 2005.

[45] Felix Schuster, Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. Vc3: Trustworthy data analytics in the cloud. Technical Report MSR-TR-2014-39, February 2014.

[46] Kehuan Zhang, Xiaoyong Zhou, Yangyi Chen, XiaoFeng Wang, and Yaoping Ruan. Sedic: Privacy-aware data intensive computing on hybrid clouds. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 515–526, New York, NY, USA, 2011. ACM.

[47] Chunwang Zhang, Ee-Chien Chang, and R.H.C. Yap. Tagged-mapreduce: A general framework for secure computing with mixed-sensitivity data on hybrid clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 31–40, May 2014.

[48] M. Atallah, A. Elmagarmid, M. Ibrahim, E. Bertino, and V. Verykios. Disclosure limitation of sensitive rules. In *Proceedings of the 1999 Workshop on Knowledge and Data Engineering Exchange*, KDEX '99, pages 45–, Washington, DC, USA, 1999. IEEE Computer Society.

[49] Josep Domingo-Ferrer, editor. *Inference Control in Statistical Databases, From Theory to Practice*. Springer-Verlag, 2002.

[50] Alban Gabillon. Multilevel databases. In *Encyclopedia of Database Technologies and Applications*, pages 386–389. 2005.

[51] Bhavani Thuraisingham, William Ford, Marie Collins, and Jonathan O'Keeffe. Design and implementation of a database inference controller. *Data Knowl. Eng.*, 11(3):271–297, December 1993.

[52] *Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis*, SIGMOD, 2009.

[53] Gerome Miklau and Dan Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.*, 73(3):507–534, 2007.

[54] Access Control in Oracle. `http://docs.oracle.com/cd/B19306_01/network.102/b14266/accessre.htm#CHDDGEJG`.

[55] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[56] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, 2004.

[57] Apache Hadoop. `http://hadoop.apache.org/`.

[58] HDFS. `http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html`.

[59] HiBench. `https://github.com/intel-hadoop/HiBench`.

[60] Mahout. `https://mahout.apache.org/`.

[61] Top 5 Use Cases for Moving to a Hybrid Cloud Solution. `http://www.infoq.com/zones/assets/downloads/WhitePaper_Top_5_Use_Cases_for_Moving_to_a_Hybrid_Cloud.pdf`.

[62] Apache Spark. `http://spark.apache.org/`.

[63] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using Hadoop. In *ICDE*, pages 996–1005, 2010.

[64] Won Kim. On optimizing an sql-like nested query. *ACM Trans. Database Syst.*, 7(3):443–469, September 1982.

[65] Chittaranjan Pradhan, Sushree Sangita Jena, and Prasanta Kumar Mahapatra. Optimized query plan algorithm for the nested query. *International Journal on Computer Science and Engineering*, 2:726, 2010.

[66] Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: Sql and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 13–24, New York, NY, USA, 2013. ACM.

[67] San Diego Supercomputer Center. `http://www.sdsc.edu/`.

[68] S. Agrawal, V. R. Narasayya, and B. Yang. Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. In *SIGMOD Conference*, pages 359–370, 2004.

[69] J. Rao, C. Zhang, N. Megiddo, and G. M. Lohman. Automating physical database design in a parallel database. In *SIGMOD Conference*, pages 558–569, 2002.

[70] S. Ghandeharizadeh and D. J. DeWitt. Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines. In *VLDB*, pages 481–492, 1990.

[71] C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: A Workload-Driven Approach to Database Replication and Partitioning. In *VLDB*, 2010.

[72] J. B. Rothnie, Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong. Introduction to a system for distributed databases (sdd-1). *ACM Trans. Database Syst.*, 5(1):1–17, March 1980.

[73] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Heterogeneous Databases and the Design of Disco. In *ICDCS*, 1996.

[74] R. van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[75] D. Logothetis and K. Yocum. Ad-hoc data processing in the cloud. *PVLDB*, 1(2):1472–1475, 2008.

[76] G. Karypis and V. Kumar. Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, University of Minnesota, 1995.

[77] Kamil Bajda-Pawlikowski, Daniel J. Abadi, Avi Silberschatz, and Erik Paulson. Efficient processing of data warehousing queries in a split execution environment. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1165–1176, New York, NY, USA, 2011. ACM.

[78] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi. Query Optimization for Massively Parallel Data Processing. In *SoCC*, pages 12:1–12:13, 2011.

[79] TPC BENCHMARK H. `http://www.tpc.org/tpch/spec/tpch2.11.0.pdf`.

[80] Vaibhav Khadilkar, Murat Kantarcioglu, Bhavani M. Thuraisingham, and Paolo Castagna. Jena-hbase: A distributed, scalable and effcient RDF triple store. In *Proceedings of the ISWC 2012 Posters & Demonstrations Track, Boston, USA, November 11-15, 2012*, 2012.

[81] S. Chaudhuri, T. Dutta, and S. Sudarshan. Fine Grained Authorization Through Predicated Grants. In *ICDE*, 2007.

[82] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.*, 73(3):507–534, 2007.