SEMROD: Secure and Efficient MapReduce Over Hybrid Clouds

Kerim Yasin Oktay, Sharad Mehrotra
University of California, Irvine
Irvine, CA
koktay, sharad@ics.uci.edu

Vaibhav Khadilkar, Murat Kantarcioglu
The University of Texas at Dallas
Dallas, TX
vvk072000, muratk@utdallas.edu

ABSTRACT

This paper describes SEMROD, a sensitive data aware MapReduce (MR) framework for hybrid clouds. SEMROD steers data and computation through public and private machines in such a way that no knowledge about sensitive data is leaked to public machines. For this purpose, SEMROD keeps trace of intermediate keys (generated during MR execution) that become sensitive, based on which it makes dynamic task scheduling decisions. SEMROD guarantees that adversaries (*viz.* public machines) cannot gain any “additional” information about sensitive data from either the data stored on public machines or the communication between public and private machines during job execution. SEMROD extends naturally from a single MR job to multi-phase MR jobs that result, for instance, from compiling Hive queries into MR jobs. Using SEMROD, computation that may involve sensitive data can exploit public machines, thereby bringing significant performance benefits. Such computation would otherwise be restricted to only private clouds. Our experiments clearly demonstrate performance advantages to using SEMROD as compared with other secure alternatives, even when the percentage of sensitive data is as high as 50%.

Categories and Subject Descriptors: H.m [Information Systems]: Miscellaneous

Keywords: Data Processing, MapReduce, Secure, Hybrid Cloud

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 and/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. However, public clouds pose a significant challenge from the perspective of security. According to a recent survey of IT executives; security, compliance and loss of control are the top 3 concerns for enterprises adopting public clouds [1].

An alternate approach gaining traction is that of a *hybrid cloud*. A hybrid cloud enables composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability [2]. 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. Indeed, the recent Beckman Workshop on Data Management [3], identifies hybrid cloud as an opportunity to achieving secure computation. The challenge, of course, is developing systems that can automatically search the space of possible partitions and choose the one that provides optimal performance while meeting the security requirements. Even with these challenges, currently 31% of IT managers prefer to use hybrid clouds rather than public and private deployments and this number is projected to reach 74% in the near future [4]. Big cloud enterprises such as Microsoft and VMWare are therefore, increasingly investing in hybrid cloud solutions [4].

For large scale data analyses, MR-based systems [5, 6] 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 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, orga-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD’15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...\$15.00.
<http://dx.doi.org/10.1145/2723372.2723741>.

nizations for whom security is indispensable are unable to deploy MR jobs over their hybrid cloud environments.

Our goal, in this paper, is to design a secure MR framework for hybrid clouds that prevents the leakage of sensitive data, while efficiently exploiting public resources for executing a given single (or multi-level) MR job. Further, this new framework should minimally change the way an end-user interacts with the MR framework. We identify 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 transmitted to a public machine for either storage or computation; and *key-inference exposure* where an adversary can infer the association of an intermediate key, generated during MR processing, with sensitive data.

To overcome these exposure scenarios, we propose a secure and efficient MR framework for hybrid clouds, entitled SEMROD. 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. 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 the corresponding reducers at the public as well as private sides. Such an extra shuffling enables reduce computation over non-sensitive data to be performed over public machines while preventing leakage of association of keys with sensitive data.

The main contributions of this paper 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 solutions¹.

We note that there have been a few attempts to building secure MR frameworks for hybrid clouds in recent literature [7]. Existing techniques limit themselves to running either

¹This work is partially supported by the following grants: Air Force Office of Scientific Research 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.

Notation	Description
J	Given sequence of MR jobs, $J = \{j_1, j_2, \dots, j_n\}$.
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 1: Notations

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 as are typical for database workloads. We will compare SEMROD to existing state of the art in details in the rest of the paper organized as follows: Sections 2 and 3 provide an overview of the original MR framework and the deployed sensitivity, adversarial and exposure models. Next, Section 4 introduces our secure and efficient MR mechanism, namely SEMROD. In Section 5, we present the implementation details of our techniques over Hadoop, a well known open-source MR framework. Then, Section 6 compares the performance of SEMROD with other secure solutions and the original Hadoop implementation for a variety of tasks. Next, Section 7 reviews the related work in the area of secure and efficient data processing over hybrid clouds. Finally, we conclude the paper and discuss extensions to further improve SEMROD’s performance in Appendix.

2. PRELIMINARIES

In this section, we provide a brief overview of MR which forms the basis of our work. In addition, we discuss the sensitivity model. To better state the ideas presented in the remainder of this paper, we use the notations described in Table 1.

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. A more extended information about the MR framework is given in Appendix.

2.2 Sensitivity Model

In the context of this paper, we will consider a *record level sensitivity model* wherein records are classified as either sensitive or non-sensitive². Our technique is independent of the exact method used to specify sensitive data *e.g.*, users may specify which records are sensitive by defining predicates. The only assumption we make is that non-sensitive

²While often, sensitive data in database literature is modeled at a finer grained level of columns/attributes, in MR setting, the concept of columns are not clearly identified/delineated. For instance, in an MR job over text that performs wordcount, what constitutes columns of the record is not well defined.

records by themselves do not contain any information about sensitive records. That is, even if the adversary could gain access to all non-sensitive data, it would not infer any sensitive information. Furthermore, regardless of the MR job being executed, any data generated in the course of a job execution 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 MR function F (map, reduce or combine) and any instance of it, namely $F((k_1, v_1), (k_2, v_2), \dots, (k_t, v_t)) = (k, v)$,

$$\text{sens}((k, v)) = \begin{cases} \text{true}, & \exists 1 \leq j \leq t, ((k_j, v_j)) \text{ is sensitive} \\ \text{false}, & \text{otherwise.} \end{cases}$$

In case of a map function, if key-value (KV) pair (k, v) is generated by mapping a sensitive input record, then (k, v) will automatically be sensitive or vice versa. In contrast, determining whether a reduce (or combiner) function's output KV pair is sensitive is more complicated. Sensitivity of a reduce output depends upon the sensitivity of each of the input key-value pairs. If any one of the inputs is sensitive, then the output key-value pair will be considered to be sensitive.

Note that, the sensitivity model we have employed during a job's execution is conservative, since we consider any function's output to be sensitive if it is sourced from at least one sensitive input. For many cases, such an assumption can be relaxed. In general, the output of a map/reduce function, F' , may not reveal any sensitive information about its input data. For such F' , a new 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 map/reduce functions is an interesting direction of exploration, we restrict our attention to only the conservative model for three reasons. First, the conservative model is more general and works regardless of the specific semantics of the given MR jobs. Second, it suffices to establish the feasibility of the solution. In particular, if we establish an efficient MR framework that guarantees that no sensitive data is leaked under a conservative model, we can further improve the framework to exploit specific semantics of MR jobs being executed. Finally, sensitivity models that vary based on the specifics of MR tasks additionally burden an end-user who will need to be tasked, in addition to defining job-specific operations, with reasoning and characterizing sensitivity of output data based on the input and operation. We defer exploration of the feasibility and support for such models to future work.

Note that our model of sensitivity classifies data as being sensitive or not and our solution will ensure that no sensitive data is disclosed during the MR function executions. Our model does not explicitly deal with the inference control problem wherein users may consider some inferences (derivable from data) such as association rules as sensitive while the data itself is deemed nonsensitive [8]. Inference control problem has been studied in the security literature in various contexts: statistical databases [9], multi-level relational database systems [10, 11], data publishing to name a few [12, 13].

Our goal, in this paper, 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 [14]). 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 framework 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 related to data. It may instead be associated with the computation – e.g., if the user wishes to execute a proprietary MR analysis in which map/reduce code is sensitive. For such a case, likely hybrid cloud setting is not appropriate since the adversary will gain knowledge of the corresponding map and reduce tasks creating a potential vulnerability. In our framework, we will assume that the map and reduce codes are not sensitive.

3. SECURITY CHALLENGES

In this section, we first identify the security challenges about how sensitive data can be exposed to an attacker when running MR jobs on public clouds and then provide the security and performance motivations behind SEMROD. Before we do so, we need to explain the adversarial model.

3.1 Adversary Model

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 sensitive information. In contrast, the adversary can have full control of the public cloud and can easily access any input/intermediate/output file stored or generated on the public cloud. Furthermore, an adversary can eavesdrop on the communication channel between the public and private clouds, and can launch statistical attacks to gain knowledge about the sensitive data stored on the private nodes. To be more specific, in the course of MR execution, an adversary can access the input and output of any public map and reduce task. In addition, the adversary can view the locations to which the public map/reduce task's outputs are shipped. The adversary is assumed to be honest-but-curious. 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 that no sensitive data or information that can provide inferences about the sensitive data is leaked to the public cloud during the MR execution.

3.2 Exposure Scenarios

Given the adversarial model, we identify two types of exposures, namely *Direct Exposure* and *Key-inference Exposure*, that could arise while either storing the data or executing MR jobs on public machines. Let us illustrate these exposure scenarios in the context of an example shown in Figure 1, which is also used later to motivate SEMROD.

Dataset D		
Name	Disease	Treatment Date
Chris	flu	Jan-13
James	flu	Jan-13
Matt	flu	Jan-13
Jane	acne	Feb-13
Matt	kuru	Feb-13
Chris	cancer	Mar-13
Zach	cancer	Mar-13
Jane	cancer	Apr-13
....		

Sensitivity
Only the records related with *cancer* are sensitive.

MR Job (J)	
Find the complete set of diseases for each patient.	
Map (M)	
<Name, Disease, Treatment Date> $\rightarrow$ <Name, Disease>	
Example: <Chris, flu, Jan-13> $\rightarrow$ <Chris, flu>	
Intermediate KV Pairs and Keys (I, K)	
Int: {<Chris, flu>, <James, flu>,}	
K: {Chris, James, Jane, Math, Zach,}	
Reduce (R)	
<Name, list(Disease)> $\rightarrow$ <Name, $\pi(\text{list(Disease)})$ >	
Example:	
<Chris, (flu, cancer, cancer)> $\rightarrow$ <Chris, (flu, cancer)>	

Figure 1: Secure MR Processing Example

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 $\langle \text{Chris}, \text{cancer}, \text{Mar} - 13 \rangle$ cannot be stored/mapped on public machines since the attacker will have direct access to such data. Thus, the last 4 records in the example must be assigned to the DFS partition on private machines. Based on our sensitivity model, that after the map phase, intermediate $\langle \text{Name}, \text{disease} \rangle$ KV pairs, where the value field includes *cancer* (e.g., $\langle \text{Chris}, \text{cancer} \rangle$) 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 an intermediate sensitive key-value 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, $\langle \text{Chris}, \text{flu} \rangle$ and $\langle \text{Jane}, \text{acne} \rangle$, will be shuffled to a private reduce task, whereas the remaining public map outputs such as $\langle \text{James}, \text{flu} \rangle$ and $\langle \text{Matt}, \text{acne} \rangle$ will remain on the public side. In such a case, the attacker can infer that *James* and *Matt* have no associated *cancer* record and, 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 HDFS and during the map phase. Due to such key-inferences, the MR framework will be more vul-

nerable 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.

3.3 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 that prevents such exposures. 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, since intermediate keys generated during the execution may become sensitive, 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 to preventing key-inference attacks in the context of an MR job directly emerge. 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, entitled *Sedic*, has recently been proposed in [7]. *Sedic* performs map operations on the public (and private) side, but shifts all the reducers to private machines³. 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 alternative, all such keys are reduced, while in the second, all keys are shifted to the private side. This prevents key-inference attacks. Neither of the two solutions above are satisfactory, especially in the context of data management workloads, for 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 assigned to public machines. However, neither of the two alternatives presented above executes both phases on public machines. (2) Data management workloads typically comprise 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

³Sedic automatically generates combiners, when possible, to reduce the amount of work done during the Reduce phase so as to best exploit public machines.

Figure 2: The map phase of SEMROD

on the private side. The details of SEMROD’s MR model are provided in the next section.

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.

4.1 Single-level MR

Map: In SEMROD, since input files either contain sensitive records, or do not any sensitive records, 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 re-

Figure 3: The reduce phase of SEMROD

sulting in any sensitive data leakage or causing a large performance overhead.

Figure 2 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 runs on the public cloud, whereas M_3^1 is executed on a private machine. The figure also portrays the concept of sensitive keyset collection for an arbitrary partition, P_x .

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 2, M_3^1 is the map task processing sensitive input split, and therefore, only it generates a sensitive key set for partition P_x and maintains it locally.

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) to the public machines, while at the same time 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 3. 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 3, $R_{x_{priv}}^1$ 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 HDFS.

In case of $R_{x_{pub}}^1$, it retrieves the data for partition P_x exclusively from the non-sensitive map task outputs (e.g. M_1^1 and M_2^1 in our example) and applies the reduce function over all incoming KV pairs without any filtering. As a result of following this protocol, $R_{x_{pub}}^1$ may generate some incorrect outputs. To illustrate, in Figure 3, reducer $R_{x_{pub}}^1$ 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 aimed 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 $R_{x_{pub}}^1$'s outputs due to reducing keys such as *Chris* and *Jane* without certain sensitive KV pairs associated with them.

Therefore, once $R_{x_{pub}}^1$ 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 $R_{x_{pub}}^1$, 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 $R_{x_{priv}}^1$ and $R_{x_{pub}}^1$ outputs, then any KV pair with that particular *prevkey* in $R_{x_{pub}}^1$ would be incorrect. To demonstrate this fact, $\langle \textit{Chris}, \textit{flu} \rangle$ and $\langle \textit{Jane}, \textit{acne} \rangle$ are the only incorrect pairs amongst $R_{x_{pub}}^1$'s outputs, and their *prevkeys* are *Chris* and *Jane* respectively. Note that *Chris* and *Jane* are the only common *prevkeys* between $R_{x_{priv}}^1$ and $R_{x_{pub}}^1$ outputs. Finally, the results of both, $R_{x_{priv}}^1$ and *final elimination*, are declared as sensitive and migrated to DFS. Remember that this reduce protocol is repeated for every reduce task R_x^1 .

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 keyset to the public reducer $R_{x_{pub}}^1$. Indeed, asking the public reducer to operate over specific keys will lead to *key-inference exposure*. Therefore, $R_{x_{pub}}^1$ is designed to reduce all incoming KV pairs. Finally, the outputs of all $R_{x_{priv}}^1$'s and the elimination operation are tagged as sensitive. The reason behind tagging $R_{x_{priv}}^1$'s output as sensitive is straightforward; however, labeling the 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 $R_{x_{pub}}^1$'s output and the elimination output. To eliminate such a leakage, SEMROD labels all job outputs as sensitive.

4.2 Multi-level MR

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

Figure 4: SEMROD Overview

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 paper 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 devise ways 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 there $\exists (k_t, v_1) \in \text{Int}^i$ s.t. (k_t, v_1) is not given as input to it.
- If $r \in \text{Inp}^i$ 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 an at least 1 incorrect input.

Given that in SEMROD input to job j_i might be incorrect and that may result in incorrect output, 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 $\text{Inp}_c^i, \text{Inp}_{ic}^i, \text{Out}_c^i$ and Out_{ic}^i 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_{ic}^i from Out^i .

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

Figure 4 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, $j_3, j_4, \dots, j_{n-1}$, 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_{ic}^i , 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.

4.2.2 Details

Let us introduce our multi-level MR framework using Figure 5. 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 5. 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_{pub}^1 , Out_{ic}^1 and Out_{priv}^1 . Out_{pub}^1 refers to the output KV pairs produced by job j_1 's public reduce tasks; whereas Out_{priv}^1 and Out_{ic}^1 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 3, only include Out_{pub}^1 and Out_{priv}^1 , but not Out_{ic}^1 . 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 3, $R_{x_{priv}}^1$ receives two KV pairs with key $Jane$: non-sensitive ($Jane, acne$) and sensitive ($Jane, cancer$). $R_{x_{priv}}^1$ 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_{pub}^1 .

Map: SEMROD maps non-sensitive input blocks such as Out_{pub}^1 and Ext_{pub}^1 on the public cloud, whereas sensitive ones (i.e. Out_{priv}^1 , Ext_{priv}^1 and Out_{ic}^1) are mapped on secure machines. An exception to this is that map results of incorrect replicas (i.e. Out_{ic}^1) 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_3^2 and M_4^2 maintain their set of sensitive keys per partition P_x locally. Furthermore, Incorrect map tasks such as M_3^2 create a set of incorrect prevkeys per partition P_x and stores them on the local disk. 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_x^2 into $R_{x_{pub}}^2$ s and $R_{x_{priv}}^2$ s. Their execution flows are given in Figure 5. $R_{x_{pub}}^2$ reduces all the non-sensitive P_x s from M_1^2 and M_2^2 , with-

out any filtering and produces $Out_{x_{pub}}^2$. As a result, the correct outputs for the intermediate P_x keys $\notin K_{x_s}^2$ such as $\langle flu, kuru \rangle$ are maintained on public machines along with some incorrect outputs.

$R_{x_{priv}}^2$ 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 ($K_{x_s}^2$) and incorrect prevkeys $K_{x_{ic}}^2$ separately. In our example, $K_{x_s}^2$ consists of the intermediate keys: $acne$, $\langle acne, cancer \rangle$, $cancer$, $\langle cancer, flu \rangle$ and flu ; whereas $K_{x_{ic}}^2$ contains $Jane$ and $Chris$.

Later, $R_{x_{priv}}^2$ pulls all P_x s from map outputs and eliminates records with a key $\notin K_{x_s}^2$ (*Initial filtering*). After this, $R_{x_{priv}}^2$ is split into 2 sub-reduce tasks if a subsequent job j_3 exists: the actual $R_{x_{priv}}^2$ and a supplementary $R_{x_{ic}}^2$. After *initial filtering*, $R_{x_{priv}}^2$ aims to create correct reduce results for keys $\in K_{x_s}^2$ (i.e. $\{acne, \langle acne, cancer \rangle, cancer, \langle cancer, flu \rangle, flu\}$), and therefore, applies another filtering via $K_{x_{ic}}^2$ 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 $R_{x_{ic}}^2$ is to create an exact incorrect replica of $Out_{x_{pub}}^2$ on the private side. To achieve this, after the *initial filtering*, $R_{x_{ic}}^2$ discards sensitive inputs like ($cancer, Zach$) – $Zach$ and applies the reduce function R^2 over the remaining non-sensitive inputs. Note that, $R_{x_{ic}}^2$ is not established unless j_3 exists.

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_{pub}^1 . Thus, the master can intuitively subtract Out_{ic}^1 from Out_{pub}^1 and obtain the remaining correct portion (sound). Moreover, when we merge correct results from Out_{pub}^1 and Out_{priv}^1 , 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. However, we note that the assumption for record-level correspondence is always expected to hold for most jobs.

DEFINITION 1. j_{i-1} and j_i is **record-level correspondent**, if $\forall (k, v) \in Out_{i-1}^1 ; (k, v)$ 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, since j_2 could be such a job that performing a *complete* map execution of j_2 would be impossible without conducting an *a priori* elimination of the incorrect portion of Out_{pub}^1 . Record-level correspondency dictates that each KV pair within the previous job's input (Out^1) has

Figure 5: SEMROD's Multi-level MR Execution

to be an independent input record for the current job (j_2). If $j_1 \Rightarrow j_2$ does not hold, then the incorrect ($Chris, \langle flu \rangle$) and the correct ($James, \langle flu \rangle$) 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_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, 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_x , 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.*

4.3 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*.

5. IMPLEMENTING SEMROD

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. We used the most popular MR implementation, namely Hadoop [6], to materialize SEMROD.

5.1 HDFS

SEMROD distinguishes sensitive and incorrect files from others through their file name. 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 sensitive / incorrect files to only private nodes.

5.2 Map

During the input splits generation, SEMROD tags each split as sensitive, incorrect or non-sensitive, based on the input file from which they are constructed. 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.

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, 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, 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.

5.3 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 $\langle key, value \rangle$ format to $\langle key, value, prevkey \rangle$ format. For map inputs not in $\langle key, value, prevkey \rangle$ form (M_i^1 s), their *prevkey* is assumed to be null. Typically, tasks append their output to a *Context* object, which is an instance of the *TaskInputOutputContext* class, 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 outputs 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 outputs as “public” and stores them on public machines.

5.4 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.

6. 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 in a standard Hadoop setup as well to understand the overhead these models bring due to additional security constraints.

6.1 Experimental Settings

Experimental Setup: We conducted our experiments on a cluster containing 18 nodes, where each node comprises a Dual-Core AMD Opteron processor with ≈ 631 GB 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.

Sedic: This model is simulated by implementing Sedic's HDFS, Map and Reduce privacy-aware scheduling ideas given that the input files are sensitive or not. As we have not implemented Sedic's automatic combiner finding feature, to be fair, we deployed combiner-free jobs in our experiments. In fact, Sedic generates partially sensitive job outputs, but it does not separate the sensitive portion from the non-sensitive one. To ensure security in our Sedic implementation, all MR job outputs are declared as sensitive.

Datasets and MR Computing jobs: In our experimental evaluation, we implemented MR jobs based on TPC-H and HiBench [16] benchmark shown in Table 2. Join query names are represented with the capital 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 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 map/reduce-heavy and single/multi-level MR jobs.

Single-level Jobs	
Median:	SELECT Lorderkey, MEDIAN(Lextendedprice) FROM lineitem GROUP BY Lorderkey
Selection:	SELECT * FROM lineitem WHERE Lquantity $\geq$ 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.P.S.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 (Lorderkey = o.orderkey)
C.O.L.S:	SELECT * FROM customer JOIN orders ON (c.custkey = o.custkey) JOIN lineitem ON (Lorderkey = o.orderkey) JOIN supplier ON (Lsuppkey = s.suppkey)
Pagerank (3 Iteration), K-Means (3 Iteration)	

Table 2: Experimental Jobs

Job	Input Size	Intermediate Size	Output Size
Median	7416 MB	1118 MB	267 MB
Selection	7416 MB	981 MB	967 MB
P.P.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 3: Job Characteristics

Figure 6: Single-level Job Results

6.2 Experimental Results

In this section, we outline the experiments that compare the performance (speed-up) of SEMROD and Sedic *vs.* *All-Private*. Sedic performs the reduce stage fully on private nodes. For this reason, a lower private/public node ratio implies a higher job execution time for Sedic. Alternatively, Sedic's experiments show that the amount of sensitive data within the input data directly impacts their overall job completion time. To demonstrate the impact of each of these parameters, we first comprehensively evaluated SEMROD and Sedic by varying one of these two criterion, while fixing the other one. The performance effects of both factors for single-level and multi-level queries are shown in Figures 6 and 7 respectively. Note that, figures on the left indicate results as the private/public ratio varies, while figures on the right display the outcomes when the sensitive data ratio changes. Furthermore, 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.

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. As expected, the results presented in Figures 6 and 7 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 by average $3.7\times$ faster than Sedic and $4.6\times$ faster than *All-Private*. In particular, for single-level queries these numbers become 1.9 and 5.1.

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 6 and 7 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 sensitive data ratio results in more computations being performed on the private side. For instance, when the sensitive data ratio is 1%, SEMROD is $2.2\times$ faster than Sedic for multi-level jobs. This number drops to $1.6\times$ for single level MR jobs.

Figure 7: Multi-level Job Results

Figure 8: SEMROD/Sedic/Hadoop vs All-Private

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.

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 8. 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.P.S 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 [17]) 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, and in turn, SEMROD's speed up compared to *All-Private* drops to 1.27x from 2.85x, the speed up in the K-means' first iteration.

Distributed vs. Centralized Key Set Collection and Sensitive Data Spread: 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 on the left subfigure in Figure 9. In most cases, distributed approach results in better execution times. In fact, centralized strategy gives better performance in Wordcount job only when the sensitive records ratio is $\leq 5\%$. Moreover, the right subfigure in Figure 9 indicates how the ratio of sensitive input records varies along the C.O.L.S execution. The ratio of sensitive records at the beginning

Figure 9: Distributed vs Centralized Key Set and Sensitive Data Spread (M-Median, W-Wordcount)

Job	Running Times (sec)		Overheads		
	Hadoop	SEMROD	Time	Shuffle	Output
Median	81	108	33%	140%	47%
Selection	64	83	30%	105%	8%
P.P.S	48	56	17%	111%	3%
P.P.S.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: Hadoop vs SEMROD

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.

Hadoop Comparison: When $\frac{priv}{pub}$ is as low as 1:5 and the sensitive data ratio is 1%, SEMROD's execution overhead (given at Table 4) for single-level queries is at most 48%. For multi-level queries, this number, however, becomes 164%. Also, Table 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.P.S.S* and *C.O.L.S*, the ratio between sensitive keys and all keys ($\frac{K_s^i}{K^i}$) is as high as $\approx 90\%$. More $\frac{K_s^i}{K^i}$ is, more the incorrect data would be generated in both public and private reducers. Thus, shifting all the output and running the entire subsequent computation to the private side would be more efficient.

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 speed⁴.

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. Varying ρ from 1 to 100 simulates a wide range of situation where inter-cloud network speeds are the same as that between local machines

⁴Amazon 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.

Figure 10: Results for Slowed Network Speeds

Job	1 Private Overhead (sec)			3 Public Overhead (sec)		
	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 5: Machine Failure Overheads

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 at Figure 10, 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*. Finally, we also conducted the same experiments in a more realistic scenario, where public and private clouds are geographically located in different locations and the connection between them is established over the wide-area network (WAN). The results in such a situation are included in the appendix and are very similar to those provided in here.

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 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.

7. RELATED WORK

Recently, research has begun to explore issues related to security in MR systems. We have extensively discussed Sedic

that offers a secure MR storage and execution framework for a hybrid cloud setting when some portion of data is sensitive [7]. Recently, [18] tries to address Sedic’s unbalanced reduce problem by tagging and tracking KV pairs with sensitivity tags. However, the proposed model, Tagged-Mapreduce, acts exactly like Sedic when the MR job is *combiner-free*, and in turn show no performance gain for most of the database queries. Also, Tagged-Mapreduce incurs *key-inference* exposure, when it partially outsources the reduce load to the public cloud. Finally, another recent effort related to security in MR is SecureMR [19] that focuses on protecting the integrity of MR data processing services.

Secure data processing in hybrid clouds (though not in the MR setting) has also been explored from a risk-aware perspective. Related work in [20] aims to solve the risk-aware workload distribution problem for the relational model in hybrid clouds in the presence of the conflicting goals of performance, data disclosure risk and resource allocation costs. Query workload partitioning over public and private machines has also been considered in papers such as Relational Cloud [21] and [22] that distributes data/workloads over multiple servers. The specific problems studied are very different – while most of the above approaches operate at distributing workloads between private and public machines at the application level, our focus is at the infrastructure level. Further, while many of the papers above explore risk-based models where partial exposure of sensitive data is acceptable, our framework is designed for preventing any leakage.

Another way of secure data processing over hybrid clouds might be to encrypt data prior to outsourcing and to perform analysis over encrypted data. Although the research community has made significant progress in developing cryptographic approaches that allow limited computation over encrypted data (*e.g.*, searchable encryption techniques [23, 24, 25], order preserving encryption [26], *etc.*), no generic and cost-efficient solution for MR use has yet emerged [27].

8. CONCLUSIONS AND EXTENSIONS

In this paper, we presented SEMROD 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. Furthermore, when inter-cloud networks is as fast as intra-cloud networks, SEMROD dramatically improves the average job execution time by as much as 3.5x as compared to alternate secure MR frameworks.

SEMROD’s sensitivity model considers the result of every computation on sensitive data to be sensitive. Mechanisms that allow users to specify when intermediate computation is sensitive or not sensitive is an interesting direction of future work. Another related extension is 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. 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. Finally, developing secure and effective approaches to deal with multi-level jobs when record-level correspondence may not hold is an interesting extension.

9. REFERENCES

- [1] Cloud Survey. <http://goo.gl/NNpgyc>.
- [2] Hybrid Cloud. The NIST Definition of Cloud Computing. *NIST, Special Publication, 800-145*, 2011.
- [3] D. Abadi et al. The beckman report on database research. The Beckman Database Research Self-Assessment Meeting, 2013.
- [4] Hybrid Cloud Usage. <http://goo.gl/8V8vt2>.
- [5] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, 2004.
- [6] Apache Hadoop. <http://hadoop.apache.org/>.
- [7] K. Zhang et al. Sedic: privacy-aware data intensive computing on hybrid clouds. In *ACM CCS*, 2011.
- [8] M. Atallah et al. Disclosure limitation of sensitive rules. In *Proceedings of the 1999 Workshop on Knowledge and Data Engineering Exchange*, 1999.
- [9] Josep Domingo-Ferrer, editor. *Inference Control in Statistical Databases, From Theory to Practice*. Springer-Verlag, 2002.
- [10] Alban Gabillon. Multilevel databases. In *Encyclopedia of Database Technologies and Applications*, pages 386–389, 2005.
- [11] Bhavani M. Thuraisingham et al. Design and implementation of a database inference controller. *Data Knowl. Eng.*, 11(3), 1993.
- [12] Frank D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. *SIGMOD*, 2009.
- [13] Gerome Miklau and Dan Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.*, 73(3):507–534, 2007.
- [14] Access Control in Oracle. <http://goo.gl/cnwQVv>.
- [15] R. Curtmola et al. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS*, 2006.
- [16] HiBench. <https://github.com/intel-hadoop/HiBench>.
- [17] Mahout. <https://mahout.apache.org/>.
- [18] C. Zhang, E. Chang, and Roland H. C. Yap. Tagged-mapreduce: A general framework for secure computing with mixed-sensitivity data on hybrid clouds. In *CCGRID*, 2014.
- [19] Wei Wei et al. Securemr: A service integrity assurance framework for mapreduce. *ACSAC '09*, pages 73–82, 2009.
- [20] K.Y. Oktay et al. Risk-Aware Workload Distribution in Hybrid Clouds. In *IEEE CLOUD*, 2012.
- [21] C. Curino et al. Relational Cloud: a Database Service for the cloud. In *CIDR*, 2011.
- [22] G. Aggarwal et al. Two Can Keep A Secret: A Distributed Architecture for Secure Database Services. In *CIDR*, 2005.
- [23] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE SP*, pages 44–55, 2000.
- [24] H. Hacigümüs et al. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.
- [25] E. Shi et al. Multi-dimensional range query over encrypted data. In *IEEE SP*, 2007.
- [26] A. Boldyreva et al. Order-preserving symmetric encryption. In *EUROCRYPT*, 2009.
- [27] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

APPENDIX

A. MAPREDUCE FRAMEWORK

MR framework allows storing and processing large volumes of data using a Distributed File System (DFS) as its underlying storage mechanism. 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. 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: 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

Figure 11: Typical MR Job Execution

completes 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* and *sort* operation, in which a reducer first uses remote procedure calls to fetch its input from local disks of slaves and then sorts incoming key-value pairs such that the ones that share the same key are grouped together.

- A *reduce* operation in which a reducer processes all pairs that share a common key. Reduce tasks are distributed across the cluster as well by partitioning the intermediate key space into $|R|$ parts using a partitioning function (the default is: $hash(k_i) \bmod |R|$), where $|R|$ denotes the number of reduce tasks. Figure 11 illustrates how a single MR job j_i is processed by the MR framework using the notations described in Table 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.

B. SECURITY ANALYSIS

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 [15] to our framework. Basically, we specify what an honest-but-curious adversary observing the public side learns 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 $\mathbf{Real}_A^{Sch}(\lambda, Inp_{ns}, M, R, side())$ and $\mathbf{Ideal}_{A,S}^{Sch}(\lambda, Inp_{ns}, M, R, side())$, where Inp_{ns} and Inp_s denotes “all the non-sensitive and sensitive inputs” and $side()$ corresponds to the function used by SEMROD to distribute input data, as follows:

$\mathbf{Real}_A^{Sch}(\lambda, Inp_{ns}, M, R, side()) : A(1^\lambda)$ is given all map (M) and reduce (R) tasks, and their outputs after execution on the public side based on the scheduler Sch ⁵. Eventually, A returns a bit that the game uses as its own output.

⁵ Sch knows Inp_s .

$\text{Ideal}_{A,S}^{\text{Sch}}(\lambda, \text{Inp}_{ns}, M, R, \text{side}()) : S(\text{Inp}_{ns}, M, R, \text{side}())$ is again given all M and R tasks, Inp_{ns} and $\text{side}()$ 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 addition, it marks all the outputs of public M 's and R 's as transmitted to the private side and gives this information to $A(1^\lambda)$. 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_{ns} and for all M, R , there exists an algorithm S such that $\Pr[\text{Real}_A^{\text{Sch}}(\lambda, \text{Inp}_{ns}, M, R, \text{side}()) = 1] - \Pr[\text{Ideal}_{A,S}^{\text{Sch}}(\lambda, \text{Inp}_{ns}, M, R, \text{side}()) = 1] \leq \text{neg}(\lambda)$ for security parameter λ and any negligible function $\text{neg}()$. 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 $\text{neg}()$ 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, $\Pr[\text{Real}_A^{\text{Sch}}(\lambda, \text{Inp}_{ns}, M, R, \text{side}()) = 1] - \Pr[\text{Ideal}_{A,S}^{\text{Sch}}(\lambda, \text{Inp}_{ns}, M, R, \text{side}()) = 1]$ 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 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 $\text{side}()$ 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 $(\text{Inp}_{ns}, M, R, \text{side}())$. 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, $\Pr[\text{Real}_A^{\text{Sch}}(\lambda, \text{Inp}_{ns}, M, R, \text{side}()) = 1] - \Pr[\text{Ideal}_{A,S}^{\text{Sch}}(\lambda, \text{Inp}_{ns}, M, R, \text{side}()) = 1]$ will be equal to zero.

Figure 12: SEMROD/Sedic vs All-Private in Real Hybrid Cloud

C. 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 have installed SEMROD, Sedic and *All-Private* systems on this new 18 node hybrid cluster. We labeled the nodes that we reserved as public and the single machine from our cluster as private. Note that in this hybrid cluster, $\frac{\text{priv}}{\text{pub}}$ 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 12. As it is seen in Figure 12, 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 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 12 clearly demonstrate that SEMROD is practical to use in a hybrid cloud, even when the inter-cloud connection is established over wide area networks.