Sedic: Privacy-Aware Data Intensive Computing on Hybrid Clouds

Kehuan Zhang, Xiaoyong Zhou,
Yangyi Chen and XiaoFeng Wang
School of Informatics and Computing
Indiana University, Bloomington, IN, USA
{kehzhang, zhou, yangchen,
xw7}@indiana.edu

Yaoping Ruan
IBM T.J. Watson Research Center
Hawthorne, NY, USA
yaoping.ruan@us.ibm.com

ABSTRACT

The emergence of cost-effective cloud services offers organizations great opportunity to reduce their cost and increase productivity. This development, however, is hampered by privacy concerns: a significant amount of organizational computing workload at least partially involves sensitive data and therefore cannot be directly outsourced to the public cloud. The scale of these computing tasks also renders existing secure outsourcing techniques less applicable. A natural solution is to split a task, keeping the computation on the private data within an organization's private cloud while moving the rest to the public commercial cloud. However, this *hybrid cloud computing* is not supported by today's data-intensive computing frameworks, *MapReduce* in particular, which forces the users to manually split their computing tasks. In this paper, we present a suite of new techniques that make such privacy-aware data-intensive computing possible. Our system, called *Sedic*, leverages the special features of MapReduce to automatically partition a computing job according to the security levels of the data it works on, and arrange the computation across a hybrid cloud. Specifically, we modified MapReduce's distributed file system to strategically replicate data, moving sanitized data blocks to the public cloud. Over this data placement, map tasks are carefully scheduled to outsource as much workload to the public cloud as possible, given sensitive data always stay on the private cloud. To minimize inter-cloud communication, our approach also automatically analyzes and transforms the reduction structure of a submitted job to aggregate the map outcomes within the public cloud before sending the result back to the private cloud for the final reduction. This also allows the users to interact with our system in the same way they work with MapReduce, and directly run their legacy code in our framework. We implemented Sedic on Hadoop and evaluated it using both real and synthesized computing jobs on a large-scale cloud test-bed. The study shows that our techniques effectively protect sensitive user data, offload a large amount of computation to the public cloud and also fully preserve the scalability of MapReduce.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Unauthorized access

General Terms

Security

Keywords

Cloud security, MapReduce, data privacy, computation split, automatic program analysis

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

1. INTRODUCTION

With the rapid growth of information within organizations, ranging from hundreds of gigabytes of satellite images to terabytes of commercial transaction data, the demands for processing such data are on the rise. Meeting such demands requires an enormous amount of low-cost computing resources, which can only be supplied by today's commercial cloud-computing systems: as an example, Amazon Elastic Compute Cloud (EC2) can easily handle terabytes of data at a price as low as 0.015 dollar per hour. This newfound capability, however, cannot be fully exploited without addressing the privacy risks it brings in: on one hand, organizational data contains sensitive information (e.g., financial data, health records, etc.) and therefore cannot be shared with the cloud provider without proper protection; on the other hand, today's commercial clouds do not offer high security assurance, a concern that has been significantly aggravated by the recent incidents of Amazon outages [13] and the Sony PlayStation network data breach [10], and tend to avoid any liability [36]. As a result, attempts to outsource the computations involving sensitive data are often discouraged.

A natural solution to this problem is cryptographic techniques for secure computation outsourcing, which has been studied for a decade [38, 17, 19, 16]. However, existing approaches are still not up to the challenge posed by data-intensive computing. For example, homomorphic encryption [30, 46, 49] was found to be prohibitively expensive for a large-scale computation [25]. As another example, the secret-sharing techniques underlying most outsourcing proposals can lead to intensive data exchanges between the share holders on different clouds during a computation involving an enormous amount of data, and are therefore hard to scale.

Secure hybrid-cloud computing. Oftentimes, a data-intensive computation involves both public and sensitive data. For example, a simple `grep` across an organizational file system encounters advertising slogans as well as lines of commercial secrets. Also, many data analysis tasks, such as intrusion detection [8], targeted advertising [14], etc., need to make use of the information from public sources, sanitized network traces and social-network data [9] for example, as well as information within an organization. If the computation on the public data can be separated from that on the sensitive data, the former can be comfortably delegated to the public commercial clouds and the latter, whose scale can be much smaller than the original task, will become much easier to handle within the organization. Such a split of computation is an effective first step to securely outsource computations and can be naturally incorporated into today's cloud infrastructure, in which a public cloud typically receives the computation "overflow" from an organization's internal system when it is running out of its computing resources. This way of computing, involving both the private cloud within an organization and the public commercial cloud, is called

hybrid cloud computing [29]. The hybrid cloud has already been adopted by most organizational cloud users and is still undergoing a rapid development, with new techniques mushroomed to enable a smoother inter-cloud coordination (e.g. [18]). It also presents a new opportunity that makes practical, secure outsourcing of computation tasks possible.

However, today's cloud-based computing frameworks, such as *MapReduce* [22], are not ready for secure hybrid-cloud computing: they are designed to work on a single cloud and not aware of the presence of the data with different security levels, which forces cloud users to manually split and re-arrange each computation job across the public/private clouds. This lack of a framework-level support also hampers the reuse of existing data-processing code, and therefore significantly increases the cloud users' programming burden. Given the fact that privacy concerns have already become the major hurdle for a broader adoption of the cloud-computing paradigm [37], it is in urgent need to develop practical techniques to facilitate secure data-intensive computing over hybrid clouds.

Our work. To answer this urgent call, a new, generic secure computing framework needs to be built to support automatic splitting of a data-intensive computing job and scheduling of it across the public and private clouds in such a way that data privacy is preserved and computational and communication overheads are minimized. Also desired here is accommodation of legacy data-processing code, which is expected to run directly within the framework without the user's manual interventions. In this paper, we present a suite of new techniques that make this happen. Our system, called *Sedic*, includes a privacy-aware execution framework that automatically partitions a computing job according to the security levels of the data it involves, and distributes the computation between the public and private clouds. Sedic is based on MapReduce, which includes a "map" step and a "reduce" step: the map step divides input data into lists of key-value pairs and assigns them to a group of concurrently-running *mappers*; the reduce step receives the outputs of these mappers, which are intermediate key-value pairs, and runs a *reducer* to transform them into the final outputs. This way of computation is characterized by its simple structure, particularly the map operations that are performed independently and concurrently on different data records. This feature is leveraged by our execution framework to automatically decompose a computation on a mixture of public and sensitive data, which is actually difficult in general. More specifically, Sedic transparently processes individual data blocks, sanitizes those carrying sensitive information along the line set by the smallest data unit ("*record*") a map operation works on, and replicates these sanitized copies to the public cloud. Over those data blocks, map tasks are assigned to work solely on the public or sensitive data within the blocks. These tasks are carefully scheduled and executed to ensure the correctness of the computing outcomes and the minimum impacts on performance. In this way, the workload of map operations is distributed to the public/private clouds according to their available computing resources and the portion of sensitive data in the original dataset.

A significant technical challenge here is that reduction usually can not be done on private nodes and public nodes separately and only private nodes are suitable for such a task in order to preserve privacy. This implies that the intermediate outputs of computing nodes on the public cloud need to be sent back to the private cloud for reduction, which could bring in a significant communication overhead. To reduce such inter-cloud data transfer as well as move part of the reduce computation to the public cloud, we developed a new technique that automatically analyzes and transforms reducers to make them suitable for running on the hybrid cloud. Our approach extracts a *combiner* from the original reducer for preprocessing the intermediate key-value pairs produced by the public cloud, so as to compress the volume of the data to be delivered to the private cloud. This was achieved, again, by leveraging the special features of MapReduce: its reducer needs to perform a *folding* operation on a list, which can be automatically identified and extracted by a program analyzer embedded in Sedic. If the operation turns out to be associative or even commutative, as happens in the vast majority of cases[1] , the combiner can be built upon it and deployed to the public cloud to process the map outcomes. In our research, we implemented Sedic on *Hadoop* [28] and evaluated it over *FutureGrid* [40], a large-scale, cross-the-country cloud testbed. Our experimental results show that the techniques effectively protected confidential user data and minimized the workload of the private cloud at a small overall cost.

Contributions. The contributions of the paper are summarized as follows:

• *A new and user-transparent secure data-intensive computing framework*. We have developed the first hybrid-cloud based secure data-intensive computing framework. Our framework ensures that sensitive user data will not be exposed to the public cloud without the user's consent, while still letting the public cloud shoulder most of the computing workload when possible. Also important is the transparency our design offers, which enables cloud users to work on the framework in exactly the same way they use the original MapReduce. As a result, legacy MapReduce jobs can be directly executed within the framework. An additional benefit that comes with this transparency is the flexibility of our computing framework: not only can it outsource all non-sensitive map tasks, but our approach can also automatically move them back to the organization perimeter when necessary (e.g., when the public cloud suffers an outage). It is important to note that these properties are achieved when the scalability of MapReduce is fully preserved. This is by no means trivial given the complexity of this execution framework, which involves carefully-designed algorithms for achieving high performance, such as the replication strategies of its distributed file system, task assignment and scheduling and others.

• *Automatic reducer analysis and transformation*. We have built a new program analysis tool that automatically evaluates and transforms the reduction structure of a computing job to optimize it for hybrid-cloud computing. The tool breaks down a reducer into components that can work on the public and private clouds respectively, which not only moves part of the reduce computation away from the private cloud but also helps control the amount of the intermediate outcomes to be delivered back to the private cloud which could cause significant delay and bandwidth charges on today's cloud model. Since inter-cloud data transfers are known to be a bottleneck in cloud computing, the new techniques offer a critical support that makes secure hybrid-cloud computing practical.

• *Implementation and evaluation*. We have implemented our design and evaluated it over a large-scale cloud testbed, using both real and synthesized MapReduce jobs. Our experimental study demonstrates that the new techniques we propose are both effective and practical.

Sedic is designed to protect data privacy during map-reduce operations, when the data involved contains both public and private records. This protection is achieved by ensuring that the sensitive information within the input data, intermediate outputs and final results will never be exposed to untrusted nodes during the computation. Another important concern in data-intensive computing is integrity, i.e., whether the public cloud honestly performs a computing task and deliveries the right results back to the private cloud. We chose to address the confidentiality issue first, as it has

[1] As a prominent example, 10 out of all 11 examples coming with Hadoop distribution contain commutative and associative folding loops.

already impeded the extensive use of the computing resources offered by the public cloud [4]. By comparison, many cloud users today live with the risk that their computing jobs may not be done correctly on the public cloud. As a prominent example, the National Institutes of Health still prohibits outsourcing of the computation involving human DNA data to the commercial cloud, though the same tasks on non-human genomes has already been delegated to Amazon EC2 [35].

Roadmap. The rest of the paper is organized as follows. Section 2 outlines the high-level design of Sedic and also explicates our objectives and adversary model. Section 3 and 4 describes the details of our secure execution framework and code transformation tool, including our implementation of these techniques. Section 5 reports our experimental study that evaluates the performance of Sedic. Section 6 surveys related prior research. Section 7 discusses the limitation of our current design and possible future research, and Section 8 concludes the paper.

2. OVERVIEW

In this section, we first explain the properties expected from Sedic and then present its high-level design and the adversary model used in our research.

2.1 Background and Design Objectives

MapReduce. MapReduce is a software framework for supporting data-intensive computing, such as web searching [22], document format conversion [43], genome sequence analysis [42, 34] and others. The computation within this framework first divides the input data into lists of key-value pairs and assigns them to a group of concurrently-running problem solvers, i.e., the mappers. Each mapper is iteratively invoked by the node that performs the computation (called *tasktracker* in Hadoop terminology) to convert an input pair into one or more intermediate key-value pairs. These pairs are fed into the reducers to transfer the pairs with the same key into a list of output pairs. Although conceptually simple, this computing framework includes a set of complicated mechanisms to ensure the scalability and fault tolerance during a computation, through replicating data and scheduling the map-reduce operations dynamically to nodes capable of undertaking the workloads. It also supports a user-defined combiner structure that allows the nodes running mappers to reduce the amount of the data that needs to be delivered to the reducers, thereby limiting bandwidth consumption. MapReduce has many open-source implementations, among which the most popular one is *Hadoop* [28]. Hadoop includes a distributed file system (HDFS) that offers reliable and high-performance storage services for MapReduce applications, and a runtime framework that manages MapReduce jobs through job submission control, task scheduling and fault tolerance. Another prominent implementation is Twister [27], which supports iterative MapReduce computations. With its distributed computing nature, MapReduce has been one of the most adopted application running on cloud.

Objectives. The original MapReduce framework does not support the operation over the hybrid cloud to process the data with different security levels. We developed Sedic to enhance MapReduce to make it suitable for performing a privacy-aware data intensive computing. More specifically, our system has been designed to achieve the following objectives:

- *High privacy assurance*. Only the public data, as indicated by the user, can be handed over to the public commercial cloud.
- *Moving workload to the public cloud when possible*. When the private cloud is about to run out of its computing resources, we need to move as much computation to the public cloud as possible, given the privacy of user data is preserved.

Table 1: Steps for a Privacy-Aware MapReduce

Users	• Label sensitive data, which can be done through a data-tagging tool (Section 3.1). • Submit to Sedic labeled data and a MapReduce job.
Sedic	• Analyze and transform the reduction structure of the job (Section 4). • Partition and replicate the data according to security labels (Section 3.1). • Create and schedule mappers across the public/private clouds (Section 3.2). • Combine the results on the public cloud and complete the reduction on the private cloud (Section 3.3).

- *Scalability*. The system should preserve the scalability of the MapReduce framework. The overhead incurred by privacy protection should be kept low.
- *Limited inter-cloud data transfer*. Inter-cloud data transfer is traditionally deemed as a very expensive operation. The bandwidth offered by today's Internet cannot sustain intensive data exchanges during the computation involving a large amount of data.
- *Ease to use*. The system should be transparent to the users, keeping the MapReduce interfaces they are familiar with, and also support a convenient migration of legacy *jobs* (i.e, MapReduce programs) to the new execution framework. These jobs can either run without any modifications, or be converted to optimize their performance through automatic program transformation.

As discussed before, the MapReduce framework is highly complicated, employing various mechanisms to improve its performance. Therefore, it is challenging to build privacy protection into the framework to meet the above requirements without undermining its scalability. Here, we present a design that makes this happen, as illustrated in Figure 1. We also prototyped the design over Hadoop. The architecture and individual components of our system are surveyed in Section 2.2 and elaborated in Section 3 and 4.

2.2 Design

The design of Sedic is meant to be generic, supporting execution of not only the MapReduce job designed for it but also legacy jobs without altering the way the user interacts with the original MapReduce platform. It includes an execution framework and auxiliary tools. The framework performs privacy-aware map-reduce operations on the data with different security labels (either *sensitive* or *public*). The auxiliary tools are used to help cloud users label their data and transform the code of their MapReduce jobs for better performance. What is expected here is that one only needs to indicate which part of the data is sensitive, and the execution framework then takes over to automatically partition the data, create and schedule map tasks on the data according to its security labels over the public/private clouds, and strategically arrange the reduce tasks to minimize inter-cloud communication. The steps for performing such a privacy-aware MapReduce are summarized in Table 1.

Specifically, before submitting a MapReduce job to the execution framework, the user can run our data-tagging tool to locate sensitive data, for example, the content involving special strings like credit-card numbers, within her dataset, and mark such data items as "sensitive". The labeled data is then uploaded to the private cloud, which is connected to the public cloud through a virtual

private network[2]. The distributed file system (DFS) of this hybrid cloud breaks the data into blocks according to their security labels and places them across the clouds strategically: the blocks containing public or sensitive data only are replicated to the public or private cloud respectively, while the others are propagated through two types of replicas, the original ones stored on the private cloud and the sanitized ones, which are cleaned of sensitive information, disseminated to the public cloud (Section 3.1). Over these data blocks, the private cloud creates map tasks, which are assigned to the nodes that host the data (Section 3.2). Such data replication and task assignment strategies, together our improvement on the execution mechanism of cloud nodes, ensure that the map tasks are computed correctly and efficiently. The results output by the nodes are delivered to the reducer to complete the computation (Section 3.3).

The reduce operation typically happens on the private cloud, as it inputs both public and sensitive data. This requires that all the mapping outcomes produced by the public cloud be transferred to the private cloud. To avoid the huge communication overhead incurred thereby and further move the computation to the public cloud, Sedic employs an automatic program analysis tool to evaluate and transform the reduction structure of a MapReduce job (Section 4). Specifically, when the user uploads her job, the tool is invoked to identify the loop within the reducer for folding the input key-value list. Once such an operation is found to be associative and commutative, which is often true for real MapReduce jobs, the loop is extracted to build a combiner and the rest of the reduce code is exported as a new reducer. The combiner is then deployed to the public cloud to pre-process mapping outcomes, which helps bring down the workload of the private cloud and reduce the volume of the data that needs to be sent to the new reducer.

2.3 Adversary Model

We consider an adversary who intends to acquire sensitive user information and has a full control of the public cloud. On the other hand, we assume that the private cloud is trustworthy: specifically, the adversary has no access to the nodes on the cloud and its underlying network, and therefore cannot launch such attacks as eavesdropping. Under this adversary model, Sedic is designed to ensure the confidentiality of a computation, though the powerful adversary can still compromise its integrity, i.e., rendering the outcomes of the computation incorrect. As discussed before, our focus on confidentiality is based on the observation that today's cloud users seem to be more willing to live with the risk of getting unreliable computing results than the threat to their private data. Finally, we assume that the absence of some records in the data blocks processed by the public cloud does not leak out information. Note that this is all that the adversary can see from a public node: Sedic ensures that the sensitive information in input/intermediate/output data never gets out of the privacy cloud.

3. THE EXECUTION FRAMEWORK

In this section, we elaborate our design of the privacy-aware execution framework within Sedic, as illustrated in Figure 1, and its implementation over Hadoop.

3.1 Data Labeling and Replication

Sensitive data labeling. As discussed before, to perform a privacy-aware MapReduce on her data, all a user needs to do is marking out the data she deems to be sensitive, and then submits the data and her computing job to Sedic, just as she would do when interacting with the original MapReduce platform. Data labeling can be done manually when only a very small amount of contiguous sensitive content is involved, or through a data-tagging tool that comes with Sedic. In our research, we implemented such a tool as a simple string scanner that searches a given dataset for the keywords or other text patterns that describe sensitive user information like social security numbers, credit-card numbers and others. Once the target is found, a security label is created to record the location of the information: the one built into our prototype is a tuple ($\langle$`filename`, `offset`, `length`$\rangle$). Also, in the case that a dataset contains multiple files, individual files can be automatically labeled according to their access privileges within file systems. For example, all except those accessible to the public should be marked as sensitive data. All the labels for a dataset are included in a meta-data file, which is submitted, together with the dataset, to Hadoop Distributed File System (HDFS).

Data uploading. To compute over a dataset, a Hadoop user first needs to upload it to HDFS, which further places and replicates the data to the nodes across a cloud. Specifically, HDFS has two types of nodes: *namenode* that maintains the meta information of the whole file system, particularly the *inode* for each file that documents its attributes (file name, modification time, etc), and *datanode* that keeps the actual data. The data stored on the datanode is organized as *blocks*, each containing 64 MB by default. The locations of the blocks that belong to the same file are recorded by the `BlockInfo` array within the file's inode. To upload a file, the user first uses her Hadoop client to contact a namenode, which creates an inode for the file, locates an available block within HDFS and sends its whereabout back to the client. According to the information, the client communicates with the datanode hosting that block to transfer its data. If the size of the file exceeds 64 MB, the client continues to request blocks from the namenode, until all the data has been uploaded. This process also includes data replication, which we discuss later.

To protect private user data from the public cloud, which is not trusted, we modified the Hadoop client and HDFS to ensure that a file with some sensitive content are uploaded through the namenode on the private cloud. Specifically, the client first contacts such a node to build an inode for the file, which includes the security labels associated with the file. To this end, we extended Hadoop's `INodeFile` class by adding a new private field, `secureMeta`, which is an array for accommodating locations of sensitive data. The content of `secureMeta` is built upon the security labels: for each label, the namenode puts in the array the location of the data record that carries the sensitive content indicated by the label. According to such meta-data, whenever the client asks for storage to upload the data involving sensitive records, the namenode first allocates to it a data block from a private datanode, i.e., the one on the private cloud. The positions of these records are also given to the datanode as the meta-data of the block to protect the records from being disclosed during the follow-up replication process.

Data replication. The data uploading process always comes with replication, which HDFS uses for the purposes of performance enhancement and fault tolerance. For each block of user data a Hadoop client requests space for, a namenode tries to replicate it to multiple data blocks on different datanodes. The number of replicas, called *replication factor*, can be specified in a Hadoop configuration file and is set to 3 by default. The replication process starts from the first datanode receiving data from the client. One by one, the data is streamed to the next datanode selected by the namenode from the prior one. This operation can also be triggered by a periodic checking performed by HDFS: once a block is found to be under-replicated, the namenode allocates space and directs datanodes to make copies of it.

The replication mechanism used in Sedic improves over that of

[2]This can be done, for example, through Amazon Virtual Private Cloud [3] when the EC2 is used as the public cloud.

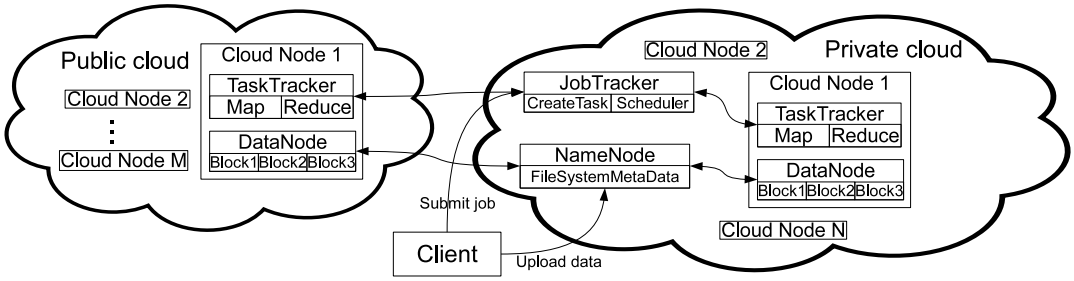

Figure 1: A Framework for Privacy-Aware MapReduce

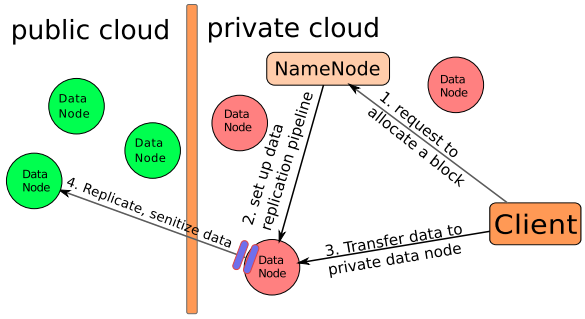

Figure 2: Data Replicate

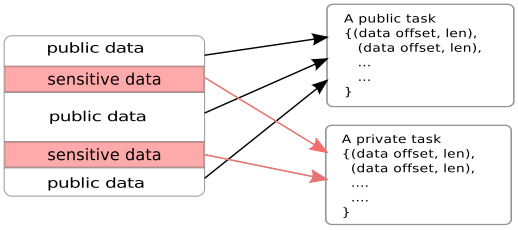

Figure 3: Task Management

Hadoop, making it suitable for operating on sensitive data in a hybrid cloud environment. Our approach ensures that the blocks with all sensitive data are only replicated to private datanodes. On the other hand, those with public data only are first sent to the public cloud, which causes the their follow-up replications more likely to happen there, due to the data locality strategy taken by Hadoop [28]. For the blocks with both public and sensitive data, the namenode first uses private datanodes to replicate them and then propagates their public data to the public nodes. This arrangement makes it possible to outsource the computation on the public data to the public cloud, while keeping the operations on the private data within the private cloud.

When replicating a data block, a private datanode copies not only the block but also its meta-data to the next node, if the recipient is also within the private cloud. Otherwise, the sender first zeros out all the sensitive records on the data block, according to the meta-data, before propagating it to a public datanode. The public node also receives the meta-data, which indicates the locations of the blank records within the block that should not be operated on. As a result of this replication, the blocks with both public and sensitive records get two types of replicas, the original versions and the sanitized ones. Figure 2 illustrates the replication process. Note that oftentimes, such an inter-cloud data transfer only happens when a new file is being uploaded, which places replicas on the public cloud. After that, the cloud user can run different jobs over this data placement. Actually, many users today already have their public data stored on the commercial cloud.

3.2 Map Task Management

Task creation and submission. After uploading data to HDFS, the client needs to submit a computing job, which includes the Java code for the mapper, reducer, other optional functions such as combiner and job configuration parameters, particularly the paths for the input and output files of the job. The node on the cloud that receives such a job is called *job submission node*. This node contains a *jobtracker* that works with the client to break the job into tasks before assigning them to the *tasktrackers* on a set of datanodes to run. More specifically, the client provides the jobtracker with a description of the map tasks over the data blocks of a file it submitted. Such a description is a list of `FileSplit` objects, with each of them associated with a map task. A `FileSplit` object carries two parameters, the offset for the start of a contiguous region within the file, a 64-MB block typically, and its length. A map task will process all the records within that block, from the beginning to the end.

To create the map tasks over the blocks with different security levels, we modified `InputSplit` to add in a security tag `sensitive`. This tag is set to sensitive when the block associated with a task carries all private content, and to public when it contains no secret at all. For the block with both types of data, a Sedic client generates two map tasks, one sensitive and one public. The offsets of these tasks point to where public or sensitive data begins and their length parameters describe the sizes of the contiguous content with the same security label within a block. To handle the block with several interleaved segments with disparate security levels, we changed `FileSplit` to specify multiple offset-length pairs. The idea is to use one map task to handle all public data of the block and the other to process the sensitive one. This treatment ensures the correctness of the keys produced by these two tasks, which often depend on the locations of individual records, as well as that of the values, since each of such segments carries integer number of records, the smallest data unit a mapper works on, as described before. The task creation is shown in Figure 3.

Task scheduling. Scheduling those map tasks to cloud nodes is done by the Hadoop jobtracker. Upon receiving a list of task descriptions, the jobtracker first creates `TaskInProgress` objects for the tasks on the list, and then assembles these objects into a `JobInProgress` queue. Whenever a "heartbeat" signal comes, indicating that a tasktracker of a node is ready to run a new task, the jobtracker looks up the queue and locates the most appropriate task for that node, the one whose data block is stored on the node, for example. This task scheduling process becomes privacy-sensitive in Sedic: we revised the jobtracker, which sits on the private cloud, to tag the `TaskInProgress` objects according to the `sensitive` fields within the `InputSplit` objects, and schedule the tasks based on their tags. Simply put, a sensitive task is always scheduled to a private datanode, which often hosts the re-

lated data block, while a public task is more likely to be handed over to a public datanode to outsource the map computation.

Task execution. A tasktracker assigned a map task by the jobtracker executes the task over the data block stored on its local host or downloaded from other nodes. In Hadoop, this proceeds as follows. The mapper reads a record from the block through a `RecordReader` object, which makes a socket connection to `DataXceiverServer`, a service maintained by the datanode that hosts the block. Each time a record is processed, the mapper gets the next one by calling the method `nextKeyValue` through the connection. This happens even when the block is co-located with the tasktracker on the same node.

A problem of this data-access mechanism is that it only reads contiguous data: whenever the read stops (e.g., due to the need of moving the read pointer to skip some data), HDFS discontinues the current connection. Given a data block often contains disconnected segments of public or sensitive data, a mapper has to call `seek` to adjust the read pointer after finishing its work on one segment, which interrupts connections. As a result, new connections need to be made continuously, which incurs a huge overhead. For example, establishing a connection takes `DataXceiverServer` (on the sender side) several hundred milliseconds to create a new `BlockSender` object for delivering data to `RecordReader` (on the receiver side).

Our solution to this problem is to read all the data segments the mapper needs within one block through a single connection. To this end, we modified `RecordReader` and `DataXceiverServer` to pass the offset-length pairs to `BlockSender`, which was also enhanced to perform both read and seek operations on a block file according to these pairs. This ensures that a connection will be torn down only after all required data is given to the mapper.

3.3 Reduction Planning

The scheduling of the reduce task poses yet another technical challenge. The reducer receives the outputs from the mappers running on sensitive data, and therefore cannot be directly executed on the public cloud. A straightforward solution here is to move all the map outcomes produced by the public cloud back to the private cloud. This, however, can incur a huge amount of inter-cloud communication, which we intend to avoid. For example, finding the TCP ports connected by each host requires a large number of port information to be transferred back to private cloud for reduce operations. Alternatively, we can carefully plan the scheduling of map tasks to ensure that the total amount of the map output to be generated by the public cloud does not exceed an upper limit set by the user according to the bandwidth she is willing to use and the delay she can tolerate. More specifically, the user is supposed to specify to Sedic the amount of output data produced by mapping one record and a threshold that represents the maximum amount of data to be sent from the public cloud to the private cloud. When scheduling a map task to a node on the public cloud, the jobtracker estimates the output volume the task will produce according to the size of the data it works on. Once the aggregated volume of the outsourced tasks is found to exceed the threshold, we stop moving computation to the public cloud. The problem of this simple treatment is that it constrains the amount of computation that can be undertaken by the public cloud.

More desired here is to let the public cloud perform part of the reduce operation, which not only cuts down the volume of the data that needs to be sent back, but also moves part of the computation away from the private cloud. What we can leverage are the properties of reducers: they all contain a fold loop that works on a list and in the vast majority of cases, such an operation is associative and commutative. Examples include counting the number of occurrences of a keyword in a large database and comparing the edit distances of different alignments to find the smallest one. For these reducers, Sedic extracts their loops to build combiners, which process the data on the public cloud and deliver their outcomes to the private cloud to complete the computation. Sedic provides an automatic program analysis tool that evaluates the source code of a reducer to determine its features and perform code transformation when necessary, which is elaborated in Section 4.

4. AUTOMATIC REDUCER ANALYSIS AND TRANSFORMATION

In this section, we present a suite of new techniques that optimize the reduction structure of a MapReduce job for secure and efficient computing of the job over a hybrid cloud. These techniques perform an automatic analysis and transformation on a reducer's Java source code, as soon as the job is submitted to the jobtracker, which enables the Sedic framework to schedule the reduce tasks in a way that minimizes the inter-cloud communication as well as the workload of the private cloud. Their design and implementation are elaborated below.

4.1 Automatic Analysis

The idea. The MapReduce computing framework has its origin in functional programming [23]. The reduce operation actually comes from *fold*, a high order function that aggregates elements on a list. Although the reducer of real-world job can be more complicated than fold, which is rather straightforward, it typically contains a fold component, in the form of a loop, to combine the values of the same keys from an input list. If the fold is associative, we can run it on the public cloud to partially process map outcomes. For example, given an associative fold $f([a_1, a_2, a_3, a_4, a_5, a_6]) = f([f([a_1, a_2]), f([a_3, a_4]), f([a_5, a_6])])$, we can outsource $f([a_1, a_2])$ and $f([a_5, a_6])$ to the pubic cloud if only a_3 and a_4 are sensitive. This move also helps reduce the communication overhead caused by sending the map outputs back to the private cloud, since the fold component partially combines these outputs. We can do even better when the fold is also commutative, which allows us to compute $f([a_1, a_2, a_5, a_6])$ on the public node in the above example. Actually, real-world reducers are often associative and commutative. In the cases they are not, their fold components typically have these properties. Figure 4 presents an example: although a reducer that calculates the mean of its input values is clearly not associative, its fold loop, which does the sum, is.

Before a fold operation can be outsourced, it needs to be analyzed to find out whether these desired properties are there. To see how to do this, let us first take a close look at the operation. Consider a list $[a_1, \cdots, a_n]$. A fold on the list can be described as $f([a_1, \cdots, a_n]) = g([g([\cdots g([g([a_1, a_0]), a_2]) \cdots]), a_n])$, where g is a function that works only on a two-element list and a_0 the initial value. Essentially, g describes the operation performed on a list member and an intermediate aggregation outcome at every iteration of the fold loop. If the operation is associative and commutative, so will be the whole fold loop. In other words, all we need to study here is what happens in a single iteration. In the rest of the section, we describe our analysis techniques, which first check the loop dependence of the fold and then evaluate the operation between a list member and the intermediate value in an iteration. We also prototyped our approach using Soot [11], a Java optimization framework.

Reducer analysis. The first step to analyzing a reducer is to locate its fold component. As discussed before, fold is typically performed through a loop in Java or other imperative programming languages. Actually, the loop is mandated by the reduce interface

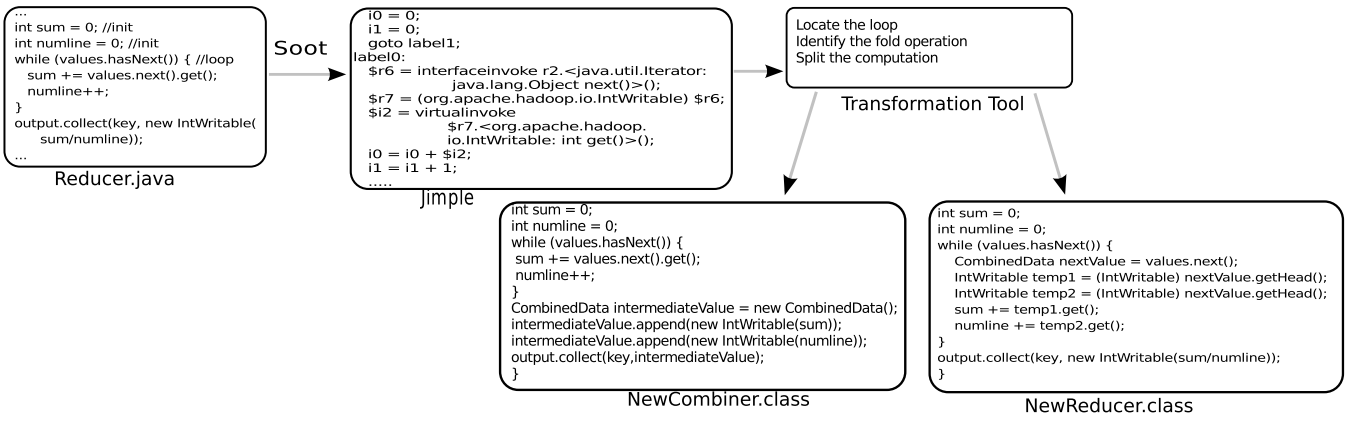

Figure 4: Reduce Structure

provided by Hadoop, which keeps the input values in an iterator. Our approach utilizes Soot to convert the Java source code of a reducer to *Jimple* intermediate representation and then runs the API `LoopFinder()` to identify this fold loop, which is characterized by its operations on these input values. All the follow-up analysis mainly happens to that loop. Specifically, we first perform a loop dependence analysis to check whether there exists any variable that is first defined within one iteration and later used in another iteration. If there is no such a variable, we conclude that the loop dependence does not exist and the loop can be used to build a combiner (Section 4.2), as it is both associative and commutative. This happens in the cases such as filtering, which drops the values above or below a certain threshold, `Grep`, `Sort`, etc.

If the dependence relations are found, our approach moves on to perform a liveness analysis on their causal variables. Specifically, the analysis utilizes a Soot API `SimpleLiveLocals` to find out those still alive posterior to the loop and put them in a set D. These variables not only carry the dependence but also produce the outcomes of the fold operation, and therefore are the keys for understanding the fold's properties. The follow-up analysis backtracks the define-use chain of each variable $v \in D$ within an iteration, starting from the first use of v outside the loop. The objective here is to discover all the variables and operations that define v across all execution paths in the iteration. This evaluation ends when it hits the input key-value list or moves out from the scope the iteration.

For each execution path discovered, our approach checks all the operators that define the loop-dependent variables. The fold becomes associative and commutative if these properties hold across all the paths. This can only be verified empirically, as the problem in general is undecidable. However, all we really need here is just a *sufficient* condition that covers the operations a real-world reducer performs, which are typically simple. Specifically, a form of the computation analyzed in our research is as follow: for each execution path i in an iteration with a path condition P_i, the reducer calculates an aggregation $v \longleftarrow v \bigotimes \delta_i$, where δ_i, which is a function, and P_i do not contain any loop-dependent variables, and $\bigotimes$ is an operator. Essentially, this computation aggregates all δ_i for each input key-value pair using $\bigotimes$. It is clearly associative and commutative if $\bigotimes$ has all these properties. Such a type of aggregation, though simple, actually generalizes the fold operations performed by the vast majority of real-world MapReduce jobs. A prominent example is sum that adds the values of the same key together. Other examples include finding the optimal alignment between two strings, where the edit distance is minimal (operator here is min), and calculating different statistics according to the values of the inputs, which are loop independent.

```
...
int sum = 0;
int i;
for (IntWritable val : values) {
        i = val.get();
        if(i > 10)
                sum += i;
}
result.set(sum);
context.write(key, result);
...
```

Figure 5: Code analysis example

Such an aggregation can be identified from the define-use chain of the variable v. Our approach evaluates all the statements on the chain to ensure that v is the only loop-dependent variables used, and the operation on the variable only happens between it and the program elements independent of other iterations, and always through the same, associative and commutative operator. We also check all the branch conditions encountered, which need to be loop independent. For the example in Figure 5, our analyzer backtracks the statements that work on the live variable `sum`, which cause the loop dependence. All such statements meet the above conditions, particularly, the addition operator is associative and commutative. Therefore, we conclude that the fold loop has all the desired properties and can be used to build a combiner. In our research, we implemented this automatic analysis in the prototype.

4.2 Code Transformation

Once the fold loop is found to be associative and commutative, Sedic uses it to build a combiner, which is deployed to both the public cloud and the private cloud to preprocess map outputs. The results of this operation are fed to a new reducer on a private node to complete the computation. In this section, we describe our code transformation technique that supports this data processing.

Our approach. Like the code analyzer, our transformation tool was also implemented as a Soot `jtp` pass [11]. It works on the Jimple representation of the Hadoop job the user uploads and saves the new code to a newly-created combiner class `CombinedData` before resubmitting the modified job to the cloud. Specifically, the tool separates the fold loop from the rest part of the reducer. As illustrated in Figure 4, the loop sits right between variable declarations/initializations and the posterior loop operations that ultimately export the outcomes of the computation to `outputCollect` of a Hadoop job. Using the information provided by the code analyzer, our tool locates the fold loop and its related variable declarations on Soot's structural representation of Jimple code, and copies

these nodes to the new combiner class. The new program structure is then converted directly into Java class files by Soot. Note that when the whole reducer includes just an associative and commutative loop and standard output functions (e.g., `Context.write`), our tool simply adds `SetReducer` to the original job to make the reducer also the combiner with proper initial value adjustment.

Although conceptually simple, this code transformation does bring in a few interesting technical issues. First, the handling of the variables in the reducer needs to be well thought-out. Remember that the combiner generated thereby is supposed to be run on both the public and private clouds. If we simply copy to it the variables and their initializations from the original reducer, we could end up with aggregating the initial values of these variables multiple times, which leads to an incorrect outcome. To see how this could happen, let us look at the example in Figure 4. In the fold loop that does the sum, if we keep the initial value of the aggregation variable `i0 = 5` in the combiner, this value will be added twice to its partial sums produced on both the public cloud and the private cloud, and added again by the new reducer, making the final result larger than what it is supposed to be by 10. Our solution is to set the initial value of the fold operation to an identity element of the fold operator $\bigotimes$ to avoid this double-counting. For example, when the operator is an addition, we can set the value to 0; when it is a multiplication, we initialize it as 1. The original initial values are only aggregated by the new reducer. Second, although the new reducer and combiner share large amount of code with the original reducer, a naive copy of Jimple `Stmt` from original reducer will not work because the `ValueBox`es in each Jimple `Stmt` refers to variables in original reducer. Besides copying the `Local` chain and `Trap` chain, we also patch the values in each `ValueBox` so it refers to variables in the new `Local` chain. Finally, we also need to add an interface between the new combiner and the new reducer. Specifically, our tool sets all the live variables of the fold loop as the output of the combiner and put them into a data structure called `CombinedData`, which is essentially a list of output objects. We further modify the loop within the new reducer, which extracts elements from the lists provided by the combiners on both clouds and aggregates these elements using the operator $\bigotimes$. We also need to remove the computation of δ from the reducer to avoid duplicate computation of δ.

An example. For the example in Figure 4, the reducer calculates the mean of input values. Our code analyzer identifies the fold loop on the structural representation of the code, moves it, together with all related variables, to the new combiner class and further modifies the reducer structure. Such code is finally converted into Java class files. For the ease of understanding, we also present the Java source code of the combiner and reducer here, though the output of our tool is actually Java bytecode.

5. EVALUATION

In this section, we report the evaluation study of our privacy-aware MapReduce framework. Our objective is to understand whether these techniques are capable of significantly reducing the workload of the private cloud, scaling to a large amount of data and also maintaining an acceptable level of overall overheads, particularly a limited inter-cloud communication cost. To this end, we evaluated the performance of realistic, large-scale Hadoop jobs using our prototype, over a large-scale cloud test-bed. The details of the study are elaborated here.

5.1 Experimental Setting

Here, we describe the setting of our experimental study, including the MapReduce jobs and the data we used to evaluate our prototype, and our hybrid cloud-computing environment.

Computing jobs and data. In our study, we ran five Hadoop jobs over our prototype to evaluate its effectiveness and performance, which include a data analysis for target marketing, two intrusion detection analyses and two jobs for preparing spam detection. The target-marketing analysis was designed to understand the public's responses to different brand names. More specifically, we utilized Hadoop's `Grep` implementation to evaluate one day Twitter data we captured on April 16th, 2010, in an attempt to find out the comment words (e.g., "wonderful", "worst", etc.) associated with different product brand names. We randomly select some user and mark their tweets as sensitive date, just as if they set their privacy preferences to share data only with their friends. The two examples for intrusion detection systems (IDS) were all based on the DARPA Intrusion Detection Evaluation dataset [5]. The dataset includes the sniffing data from external networks, which was marked as public in our study, as well as that from the internal network, which was considered to be sensitive. Our analyses on the data involved finding all the ports connected by each IP address and determining the amount of traffic generated by individual hosts. The last two jobs prepared a Naive Bayesian classifier for detecting email spam: one of them counted the occurrences of a set of words on a spam keyword list and the other counted the total number of words in a large dataset. We utilized the published Enron email dataset [6] as the private data and a SPAM archive [12] as the public data. These jobs and their related datasets are described by Table 2 and 3. We also performed an additional experiment to understand the performance of Sedic on the dataset with various proportions of sensitive records, using a job that computed the average lengths of packet payloads over the IDS dataset. The code of the job came from Hadoop sample code [28].

Table 2: Descriptions of Hadoop Jobs

Job	Data set	Descriptions
Port Scan Detection	IDS data set	Find the TCP ports connected by each host
Traffic Statistics	IDS data set	Count the total amount of the traffic generated by each host (for detecting denial of service attacks)
Email Word Count	Spam data set	Count the total number of words in the spam dataset (for calculating Bayes probability)
Spam Keyword Count	Spam date set	Count the occurrences of each keyword on a given spam keyword list file(for calculating Bayes probability)
Grep	Twitter	Search for word patterns according to predefined regular expressions within the dataset, e.g., brand names and comment words such as awesome, wonderful, worst etc.

The hybrid cloud. We built our hybrid cloud on FutureGrid [7], an NSF-supported, across-the-country cloud test-bed [40]. The public cloud included 3 nodes located at the University of Chicago. Each machine has 8-core 2.93 GHz Intel Xeon, 24 GB memory, 862 GB local disk and Linux 2.6.18. The private cloud involved 3 nodes at Indiana University, each with 8 to 24 cores of Intel Xeon CPUs, 32G or 48GB memory, 1.6TB disk and also Linux 2.6.18. These two clouds were connected by a 40 MBps link.

5.2 Experimental Results

In our experiment, we evaluated both the effectiveness of our code transformation tool and the performance of our execution framework, which includes computational and communication overheads.

Code transformation. Like the typical reduction performed in a

Table 3: Descriptions of Datasets

Name	Sensitive Data	Public Data	Size of Sensitive Data	Size of Public Data	Percentage of Sensitive Content
IDS data set	The tcpdump files for inside	The tcpdump files for outside	17GB	15GB	54%
Spam data set	The Enron Email Dataset	SPAM archive download from http://untroubled.org/spam/	1.3GB	0.8GB	62%
Twitter data set	Tweets from randomly chosen users who are assumed to prefer to protect their tweets	Tweets from other users	123MB	491MB	20%

MapReduce job, all the reducers encountered in our experimental study were pretty simple. Actually, most jobs simply summed up the values produced by mappers according to the keys. Our analysis tool easily identified the fold loops within them and determined that they were all commutative and associative. These reducers were also set as combiners so that they could combine the data at the public and private clouds respectively, and then reduce the outputs on the private node. The exceptions include the tasks for collecting all the ports associated with individual IP addresses and for determining the average lengths of packet payloads. The fold loop of the former did not have a loop dependency and therefore was directly used as a combiner. The latter calculated the mean of all its inputs, which was analyzed and transformed as described in Section 4.2.

Performance. We ran the job that computes average payload lengths on the IDS dataset to analyze the performance of our execution platform in the presence of different public/sensitive data mixtures. Specifically, we considered the situation that sensitive data was uniformly distributed to every data block, since it represents the worst-case scenario with the most negative impact on the performance of our system. Under this scenario, we gradually raised the proportion of sensitive information, from 10% to 50%, to evaluate the amount of the computation being outsourced to the public cloud. The workload here was measured by the total task execution time, which was summed over the execution time of individual tasks (map and reduce) processed by private nodes. Such workload was compared with that of running the whole job within the private cloud, which we call *baseline*, to estimate the ratio of the computation being outsourced to the public cloud. We got all the runtime statistics from Hadoop log files, which has the detailed information about each task, including the starting time, running on which TaskTracker and finishing time. The outcomes of this study are illustrated by Figure 6. The workload the private cloud shouldered increases from about 69.68 seconds (an outsource ratio about 76%), when 10% of the dataset was sensitive, to about 168.88 seconds (about 40% of outsource ratio), when 40% of the data was private, compared with that caused by running the whole job on the private nodes. The workload dropped a bit when the ratio of the sensitive data went to 50%, probably due to the randomness in execution. Remember that the map task undertaken by the private cloud only processes the sensitive records within each block (Section 3.2). However, there apparently were noticeable overheads associated with initializing the task and seeking for these records within a block, which brought down the performance.

The performance of other jobs are described in Table 4. What we can see here is that all of them successfully moved a large portion of computation to the public cloud, in accordance with the ratio of the private information within the individual datasets they worked on. This even happened to the job that processed the Twitter dataset, whose distribution of sensitive data was more even compared with the other two datasets. In some cases, the outsource ratios are even higher than the proportion of the public data within the dataset (Word Count). We believe that this was caused by the randomness in the job execution. The overall job execution time was also low

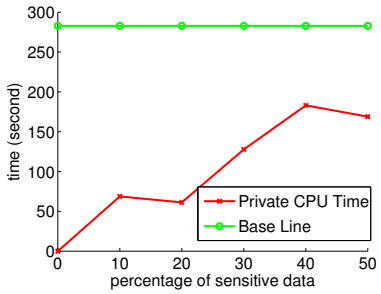

Figure 6: Performance vs. Sensitive Data Ratio

for our prototype: as an example, for port scan detection, it took about 3 minutes to complete the whole job (on 6 nodes, 3 private and 3 public), even below the baseline, which used about 6 minutes (on 3 private nodes). This is actually quite reasonable, given the fact that our hybrid-cloud computing leveraged the resources from the public cloud.

Communication overheads. Our experiments also show that the new reduction structure automatically generated by Sedic did contribute to the effective control of inter-cloud data transfers, one of the major hurdles to the extensive use of cloud computing. Specifically, we compared the bandwidth consumption of the original Hadoop jobs with that incurred after their reducers were transformed. The results are presented in Table 4. As we can see here, in the presence of code transformation, the flood of the traffic stream between the public and the private clouds was reduced to trickles: for example, in the case of port scan detection, over 1.5 GB data was reduced to merely 8.2 MB using the automatically generated combiner on the public cloud. This saving in bandwidth consumption can significantly improve the performance of hybrid-cloud computation, given the fact that inter-cloud bandwidth is typically low. For example, the link between Amazon EC2 and Amazon S3 is around 50MBps [1] and the connections EC2 receives from the outside are often less than 10MBps [2]. The experimental results offer strong evidence that our code analysis and transformation techniques indeed work.

6. RELATED WORK

Security protection in cloud computing. Most system security research on the cloud focuses on data-storage security [47] and virtualization security [33]. Little effort has been made to facilitate security and privacy protections during the computation specific to this new computing platform (despite the rich literature on more generic secure-computing techniques). One exception is Airavat [41], a system that ensures mandatory access control and differential privacy [26] when MapReduce operations are performed on sensitive data. Different from Airavat, which trusts the cloud platform it is running upon, our work aims at protecting sensitive data from the public cloud, which will be achieved by a security mechanism designed for the hybrid-cloud platform.

Table 4: Performance

ID	Job	Task Execution Time of the Whole Job (Baseline, in seconds)	Total Task Execution Time in Private Cloud(in seconds)	Outsource Ratio	Public Data Ratio	Inter-cloud Communication w/o Transformation (in Bytes)	Inter-cloud Communication w. Transformation (in Bytes)
1	Port scan detection	1909.79	928.86	51%	46%	1512075005	8215706
2	Traffic statistics	1723.87	890.25	48%	46%	672975164	582889
3	Word count	148.48	66.12	55%	38%	422691678	294
4	Spam Keyword Count	155.70	99.03	36%	38%	68598926	13173
5	Grep	17.85	5.24	71%	80%	2158	156

Secure computation outsourcing. Secure computation outsourcing has been studied for more than a decade. Early research is mainly on delegating cryptographic operations (e.g., modular exponentiations) to a set of untrusted helpers [38, 31]. More recent studies include the techniques for secure computing of edit distances and string alignments [19, 16, 32, 20, 44], which are too heavyweight for data-intensive computations. Effort has also been made to securely outsource the computations such as linear algebra operations [15] and machine-learning tasks [25]. For example, Peer-for-Privacy decomposes a category of data mining algorithms into vector addition steps and distributes them to multiple nodes on a cloud, which can be securely evaluated through a special secret sharing scheme [25]. All these approaches, however, incur a large amount of communication during the computation. Also, secret-sharing based approaches may bring in new policy challenges: once the data has been shared to multiple parties, an organization completely loses the control of it, since these parties can work together to restore the data; it is still unclear whether the organization needs to sign an agreement with each of them, which these parties may want to avoid for liability concerns [36], and if so, what the agreement will look like. This concern is also applied to the approaches that decompose a computation problem [44] into small sub-problems and allocates them to multiple problem solvers, under the assumption that these parties will not collude.

Computation split. The idea of splitting a computation among multiple parties for security purposes has been explored under different scenarios. For example, Swift [21] uses a secure information-flow analysis [39] to partition a web application into client and server components. Other examples include distributing a genomic computation to two or more parties [44, 48]. However, none of these techniques are designed for data-intensive computations, the focus of our research.

7. DISCUSSION

Sedic is designed to work on the data whose sensitive records are known to its owner and can therefore be marked out. This is true in most real-life situations: for example, documents in organizations' file systems oftentimes are already labeled by their access privileges. On the other hand, there are situations when the owner herself has no idea about which part of the data is sensitive. A prominent example here is *mapping of human DNA sequences*, which is also known as *read mapping* [45]. The task is to align a short human DNA sequence to a long reference genome, so as to identify the genetic location of the sequence. The challenge here is that before the alignment, we have no idea where the sequence belongs, no to mention whether it carries sensitive information. In this case, we have to come up with a specific solution to such a problem, by carefully considering its special features.

Our approach is built upon Hadoop, which does not support iterative MapReduce [23]. To execute a task that needs to perform multiple rounds of map-reduce operations, we have to break it down into multiple Hadoop jobs. Also, Sedic requires an additional step to label the output of one job so that its follow-up job can be split between the public and the private clouds. In the future research, we plan to move our design to Twister [27] to support iterative MapReduce, which is important to a set of data mining analyses [24]. We could also embed a lightweight information-flow tracking and declassification mechanisms into the execution framework to enable automatic labeling of a job's sensitive outputs.

The design of Sedic can also be improved. For example, our experimental study found that the data block involving public or sensitive data alone can be more efficiently processed than those containing both types of information. A straightforward solution seems to be simply re-organizing the data, clustering sensitive records from different blocks into new blocks. This approach, however, could cause the outcomes of the computation to be incorrect, as the keys generated by mappers are often related to the positions of the blocks in the datasets. To solve this problem, we need to attach such position information to individual records being clustered. The question is, whether the extra workload to process such information can lead to a significant performance degradation. This will be investigated in the future research.

Our current code analysis and transformation tool can only handle the type of the reducers as described in Section 4.1. This approach seems to be good enough, as the vast majority of the reducers in real-world jobs are rather simple. However, it is still important to understand whether there are strong demands for more complicated reduction operations, which may not be associative. If such demands are indeed there, the analysis and transformation techniques certainly need to be improved to accommodate the new applications. This issue, again, is on our research agenda.

8. CONCLUSION

Commercial cloud services, such as the Amazon EC2, enable their customers to process a large amount of data at a low cost. This benefit, however, comes with privacy risks: the computing tasks of organizations often involves sensitive data and therefore cannot be directly delegated to the public cloud without proper protection. Such protection cannot be expected from traditional secure outsourcing techniques, which often cannot handle the large amount of data such computation involves. A more practical solution is to split the computation so as to move the workload unrelated to sensitive data to the commercial cloud, while keeping the rest within an organization's private cloud. This hybrid computing paradigm needs to be supported by a new privacy-aware computation framework. To this end, we present Sedic, the first secure data-intensive computing system, in this paper. Our approach leverages the special features of MapReduce to schedule individual map tasks over a carefully planned data placement, in a way that the tasks within the private cloud only work on sensitive data and those on the public cloud only processes public data. As a result, all the workload that does not involve private information can be offloaded to the low-cost commercial cloud. To avoid an intensive data exchange

between clouds, Sedic also automatically analyzes the reducer of a legacy MapReduce job to extract a combiner for aggregating the map outcomes on the public cloud. We implemented our techniques on Hadoop and evaluated our prototype on FutureGrid, a large-scale cloud test-bed. Our study shows that without jeopardizing user privacy, Sedic effectively outsourced a large amount of computing workload to the public cloud, fully preserved the scalability of MapReduce and also conveniently accommodated legacy computing jobs.

Acknowledgements

We thank Kumar Bhaskaran, Milton H. Hernandez and Xiaolan (Catherine) Zhang for their valuable comments and advices, and Vijay Naik for his knowledge about hybrid cloud. We also thank anonymous reviewers for their insightful comments. Kehuan Zhang was also supported in part by the NSF CNS-0716292, CNS-1017782 and the IBM internship program.

9. REFERENCES

[1] Network performance within amazon ec2 and to amazon s3. http://blog.rightscale.com/2007/10/28/network-performance-within-amazon-ec2-and-to-amazon-s3/, 2008.

[2] Testing amazon web services bandwidth. http://jonathanmaim.com/2008/05/testing-amazon-web-services-bandwidth.html, 2008.

[3] Amazon virtual private cloud. http://aws.amazon.com/vpc/, 2011.

[4] Awareness, trust and security to shape government cloud adoption. http://www.lockheedmartin.com/data/assets/isgs/documents/CloudComputingWhitePaper.pdf, 2011.

[5] Darpa intrusion detection data set. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html, 2011.

[6] Enron email dataset. http://www.cs.cmu.edu/~enron/, 2011.

[7] Future grid portal. https://portal.futuregrid.org/, 2011.

[8] An introduction to distributed intrusion detection systems. http://www.symantec.com/connect/articles/introduction/distributed/intrusion/detection/systems, 2011.

[9] Social media data helping to target extend demand gen campaigns. http://www.demandgenreport.com/archives/feature-articles/594-social-media.html, 2011.

[10] Sony: Hacker stole playstation users' personal info. http://edition.cnn.com/2011/TECH/gaming.gadgets/04/26/playstation.network.hack/index.html, 2011.

[11] Soot: a java optimization framework. http://www.sable.mcgill.ca/soot/, 2011.

[12] Spam archieve. http://untroubled.org/spam/, 2011.

[13] Summary of the amazon ec2 and amazon rds service disruption in the us east region. http://aws.amazon.com/message/65648/, 2011.

[14] Target marketing. http://en.wikipedia.org/wiki/Target_market, 2011.

[15] M. J. Atallah and K. B. Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 48–59, New York, NY, USA, 2010. ACM.

[16] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, WPES '03, pages 39–44, New York, NY, USA, 2003. ACM.

[17] M. J. Atallah and J. Li. Secure outsourcing of sequence comparisons. *Int. J. Inf. Secur.*, 4:277–287, October 2005.

[18] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow. Blueprint for the intercloud - protocols and formats for cloud computing interoperability. *Internet and Web Applications and Services, International Conference on*, 0:328–336, 2009.

[19] M. Blanton and M. Aliasgari. Secure outsourcing of dna searching via finite automata. In S. Foresti and S. Jajodia, editors, *DBSec*, volume 6166 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2010.

[20] F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls. Privacy-preserving matching of dna profiles. Technical Report Report 2008/203, ACR Cryptology ePrint Archive, 2008.

[21] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng. Secure web applications via automatic partitioning. *SIGOPS Oper. Syst. Rev.*, 41:31–44, October 2007.

[22] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[23] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[24] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[25] Y. Duan, N. Youdao, J. Canny, and J. Zhan. P4p: Practical large-scale privacy-preserving distributed computation robust against malicious users abstract. In *Proceedings of the 19th USENIX Security Symposium*, Washington, DC, August 2010.

[26] C. Dwork. Differential privacy. In *in ICALP*, pages 1–12. Springer, 2006.

[27] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: A runtime for iterative mapreduce. Technical report, Indiana University, Bloomington, IN, 2010.

[28] T. A. S. Foundation. Apache Hadoop Project. http://hadoop.apache.org/, 2010.

[29] B. Furht. Cloud computing fundamentals. In B. Furht and A. Escalante, editors, *Handbook of Cloud Computing*, pages 3–19. Springer US, 2010.

[30] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[31] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In J. Kilian, editor, *Theory of Cryptography*, volume 3378 of *Lecture Notes in Computer Science*, pages 264–282. Springer Berlin / Heidelberg, 2005.

[32] S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy*, 2008.

[33] A. V. Konstantinou, T. Eilam, M. Kalantar, A. A. Totok, W. Arnold, and E. Snible. An architecture for virtual solution composition and deployment in infrastructure clouds. In *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, VTDC '09, pages 9–18, New York, NY, USA, 2009. ACM.

[34] B. Langmead, M. Schatz, J. Lin, M. Pop, and S. Salzberg. Searching for SNPs with cloud computing. *Genome Biology*, 10(11):R134+, November 2009.

[35] A. W. S. LLC. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/, 2010.

[36] A. W. S. LLC. Amazon Web Services Customer Agreement. http://aws.amazon.com/agreement/, 2010.

[37] M. C. I. Lockheed Martin, LM Cyber Security Alliance. Awareness, trust and security to shape government cloud adoption. http://www.lockheedmartin.com/data/assets/isgs/documents/CloudComputingWhitePaper.pdf, April 2010.

[38] T. Matsumoto, K. Kato, and H. Imai. Speeding up secret computations with insecure auxiliary devices. In *Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, pages 497–506, London, UK, 1990. Springer-Verlag.

[39] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Trans. Softw. Eng. Methodol.*, 9(4):410–442, 2000.

[40] NSF. Award abstract #091081 - futuregrid: An experimental, high-performance grid test-bed, 2009.

[41] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, pages 297–312. USENIX Association, 2010.

[42] M. C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[43] A. W. Services. Aws case study: Washington post. http://aws.amazon.com/solutions/case-studies/washington-post/, December As of 2010.

[44] D. Szajda, M. Pohl, J. Owen, and B. G. Lawson. Toward a practical data privacy scheme for a distributed implementation of the smith-waterman genome sequence comparison algorithm. In *NDSS*. The Internet Society, 2006.

[45] C. Trapnell and S. L. Salzberg. How to map billions of short reads onto genomes. *Nature biotechnology*, 27(5):455–457, May 2009.

[46] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer Berlin / Heidelberg, 2010.

[47] C. Wang, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 525–533, Piscataway, NJ, USA, 2010. IEEE Press.

[48] R. Wang, X. Wang, Z. Li, H. Tang, M. K. Reiter, and Z. Dong. Privacy-preserving genomic computation through program specialization. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 338–347, New York, NY, USA, 2009. ACM.

[49] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *In SIAM SDM*, pages 21–23, 2005.