A Solution For Privacy Protection In MapReduce

37-106530
Tran Quang

2012/02/08

Abstract

Recently, the development of storage and networking technology have made processing tremendous data become real. As a result, the demand of discovering knowledge from the bigdata by using tools such as statistical analysis and data mining naturally become higher. Using MapReduce a software framework introduced by Google in 2004 to implement computations on clusters of commodity computers is an economical solution. However, malicious MapReduce framework or source codes can leak the sensitive data through computation process.

An approach which giving user the least privilege on MapReduce-based system can solve the problem. Therefore, in our research, we propose a MapReduce-based system limiting the access to system resource by using SELinux. Moreover, noise were added to the output of the Reduce to ensure the computational result can not signal the presence of a sensitive data. Our prototype implementation demonstrates the efficiency of preserving privacy on several cases.

Contents

Chapter 1

Introduction

The development of technology is changing the habit of using computer. Internet-based data storage and services also known as cloud computing are rapidly emerging to complement the traditional model of software and data being stored on desktop PCs and servers. Cloud computing is a solution to enhance computing experiences by enabling users to access software applications and data that are stored at off-site datacenters rather than on the user's own PC or at an organization's on-site datacenter. Cloud computing solutions are also being adopted by governmental organizations[1].

Main services offered by clouds are data storage and data processing through user-defined or publicly available programs. Both storage and processing services provided by the cloud are characterized by high performance and quality of service in that most cloud computing services offer reliable storage and management for very large amounts of data as well as highly scalable and parallel data processing facilities. Cloud computing services often rely on specific systems such as Hadoop MapReduce[2], an open source proposed by Google. MapReduce is being adopted by many academic researchers for data processing in different research areas, such as high-end computing, data intensive scientific analysis, large scale semantic annotation and machine learning. As a forerunner in this area, Amazon deploys MapReduce as a web service using Amazon Elastic Compute Cloud(EC2) and Amazon Simple Storage Service(S3). It provides a public data processing services for researchers, data analysts and developers to efficiently and cost-effectively process vast amounts of data [3].

Together with the advantages, computing using MapReduce also come with many problems. Besides communication security attacks such as eavesdropping attacks, replay attacks, and Denial of Service(DoS), MapReduce

also faces privacy issue since a careless or malicious application of MapReduce may expose sensitive data by writing it into a world-readable file or by outputting a specific result to signal the presence of a sensitive item in datasets of input. Many researches to data privacy were proposed especially in data mining[4][5][6]. Anonymization algorithms such as k-Anonymity[7], l-diversity[8], t-closeness[9] which remove identifiable information is widely adopted. Unfortunately, anonymization does not provide meaningful privacy guarantees[10, 12].

A big challenge of preserving privacy and security in cloud computing is that developers and users want to spend as little effort and system resources on security as possible. Airavat[11] introduced by I.Roy ensure the privacy in MapReduce. But, usability is critical since it confine users to use a few Reducers. Our research introduces new method to raise usability of Airavat.

1.1 Statement of Problem

Considering a supermarket chain JMarket, which holds a large database of customer transactions. Assuming that all records in the database have the form (customer info, item, number of item) with only one record for each customer. A market analyst, Peter collaborates with JMarket to mine the data to analyze the transaction patterns. JMarket put the data into Hadoop cluster and Peter writes MapReduce code to analyze it.

In computation using MapReduce, a programmer typically implements Mapper and Reducer interfaces to the map and reduce methods. The main risk for JMarket is the fact that Peter's code is untrusted and can therefore be unintentionally buggy and even malicious. Because a mapper can directly access to JMarket's records, hence it can save a part of specific customer's information in a file and even send it through network connection. However, it's not only in the Mapper process. Peter's program can steal sensitive data in even Reducer process. The example in List 1.1 show the method that attacker can use to steal sensitive data.

In the code sample, attacker tries to find a presence of a specific customer "Suzuki" in data at Line 14. Then, he output the keyword "Suzuki" or manipulating to key variable at line 15 or to set a big value to output result at line 16. In addition, as line 33, he can also scan the list of value in Reducer to find presence (or absence) of a strange value and signal to Reducer's result by output "12345678" at line 35. Clearly, the result of the computation in

this case violates the privacy of the customer. Such a leak can effect a lot to JMarket's business and may present a serious reputational risk if individual JMarket transactions were made public without the agreement of customers.

In sum, attacker can scan the presence or absence of an item in datasets and use storage channel like network or writing file to leak information. Moreover, because the outputs of mappers and reducers are list of key/values pairs. An untrusted mapper or reducer may try to leak information of an item by encoding it in:

a. The values it outputs
b. The keys it outputs
c. The order in which it outputs key/value pairs.
d. Relationships between output values of different key.

Airavat could efficiently support distributed computations and to provide privacy guarantees. However, there are only 3 reducers(Sum,Count,Threshold) that users can use.

1.2 Objective of Research

In our research, we want to create a cloud based computational system with the following goal in mind:

1. Enables efficient distributed computations
2. Provides privacy's enhancement to results.
3. Supports a friendly usability that users can write Mapper and Reducer codes by himself with familiar programming model.

```
1 public static class Map extends MapReduceBase implements
2             Mapper<LongWritable, Text, Text, IntWritable> {
3
4   private Text word = new Text();
5
6   public void map(LongWritable key, Text value,
7                   OutputCollector <Text, IntWritable>
8                   output, Reporter reporter)
9                                   throws IOException {
10    String line = value.toString();
11    StringTokenizer tokenizer = new StringTokenizer(line);
12    while (tokenizer.hasMoreTokens()) {
13      String keyword = tokenizer.nextToken();
14       if (keyword.compareTO("Suzuki")){
15         word.set("Sazaki");
16         output.collect(word, 1000000);
17        } else {
18         word.set(keyword);
19         output.collect(word, 1);
20       }
21    }
22  }
23 }
24
25 public static class Reduce extends MapReduceBase implements
26  Reducer<Text, IntWritable, Text, IntWritable> {
27   public void reduce(Text key, Iterator<IntWritable>
28                 values, OutputCollector<Text, IntWritable>
29                 output, Reporter reporter) throws IOException {
30    int sum = 0;
31    while (values.hasNext()) {
32       int rValue = values.next().get();
33       if (rValue>=1000000)){
34            key.set("Sazaki");
35            sum = 12345678;
36            break;
37            } else {
38            sum += rValue;
39     }
40    }
41    output.collect(key, new IntWritable(sum));
42  }
43 }
```

List 1.1: Malicious Code Sample

1.3 Thesis Outline

This thesis is structured as follows. The next chapter describes the researches related to our thesis. Techniques (MapReduce, Differential Privacy, SELinux, Program Analysis) used in our proposal will be respectively presented in Chapter 3, Chapter 4, Chapter 5 and Chapter 6. Proposal is introduced in Chapter 7. We describe about experiment and evaluation in Chapter 8. Finally, we make conclusion and discuss about future work in Chapter 9.

Chapter 2

Related Work

A family of research work [65, 66, 67, 68, 69] called privacy preserving distributed data mining (PPDDM)[63] aims at performing some data mining task on a set of private databases owned by different parties. It follows the principle of Secure Multiparty Computation (SMC) [64] and prohibits any data sharing other than the final data mining result. Clifton et al. [63] present a suite of SMC operations, like secure sum, secure set union, secure size of set intersection, and scalar product, that are useful for many data mining tasks. In contrast, PPDP does not perform the actual data mining task, but concerns with how to publish the data so that the anonymous data are useful for data mining. We can say that PPDP protects privacy at the data level while PPDDM protects privacy at the process level. They address different privacy models and data mining scenarios.

The birth of Cloud Computing leads to a new challenge for protecting privacy and security. Airavat is a MapReduce (Hadoop)-based system which provides strong security and privacy guarantees for distributed computations on sensitive data. The authors implemented Airavat by re-using previous privacy-preserving concepts, like Mandatory Access Control (as implemented by SELinux) and differential privacy. Differential privacy works best for low-sensitivity computations, where the maximum influence any given input can have on the output of the computation is low.

Airavat was designed so that it could efficiently support distributed computations and to provide privacy guarantees. The current implementation supports both trusted and untrusted Mappers, but Reducers must be trusted (i.e., selected from one of the Reducers provided by Airavat). To implement Airavat, the authors used SELinux, and added SELinux-like mandatory access control to the HDFS. The JVM and MapReduce framework (Hadoop)

were also altered to enforce differential privacy. Mainly, the JVM was modified so that Mappers cannot use information from other Mappers, while Hadoop was modified to support differential privacy. Since users can only three Reducers (Sum,Count,Threshold), usability of Airavat is bad and in some case it's useless.

On the other hand, static code analysis has been rapidly developed. For example, FindBugs an open source program created by Bill Pugh and David Hovemeyer. It uses static analysis to identify hundreds of different potential types of errors in Java programs. FindBugs operates on Java bytecode, rather than source code. The software is distributed as a stand-alone GUI application. There are also plug-ins available for Eclipse, Netbeans, IntelliJ IDEA, Hudson, and Jenkins. Tools such as Findbugs can be used to detect potential bugs in a MapReduce program.

Chapter 3

New Trends with Cloud Computing

Some of trends are creating the era of Cloud Computing, which is an Internet-based development and use of computer technology. By combining a set of existing and new technologies from research ares such as Service-Oriented Architecture (SOA) and virtualization, cloud computing is regarded as such a computing revolution. In company with cloud computing, various business models are developed, which can be described by the terminology of "X as a service (Xaas)". X can be a software(Saas), Platform (Paas), and even Infrastructure(IaaS). As commercial activities on the Internet we already see gigantic clouds such as Amazon's EC2 and S3, Google App Engine, and Force.com by the Internet application vendors. The giant software vendors Microsoft also have launched the cloud services called Microsoft Azure. Furthermore, governmental activities are fast catching up commercial activities, including U.S, UK, and Canada.

Cloud Computing service provider such as Amazon S3, Google App Engine and Microsoft Azure provide users with scalable resources in the pay-as-you-use fashion at relatively low prices. For example, Amazon's S3 data storage service just charges $0.12 to $0.15 per gigabyte/month. Comparing to building own infrastructures, clients are able to save their investments by migrating business to the clouds. The rapid development of Cloud Computing will create a future where almost businesses will be moved into the cloud to reduce management and maintenance cost. One of the business is computation using cloud computing to process bigdata.

3.1 What is Cloud Computing?

According to NIST(The US National Institute of Standards and Technology) [13], Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

Essential Characteristics

1. On-demand self-service: A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each services provider.

2. Broad network access: Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g.,mobile phones, laptops, and PDAs).

3. Resource pooling: The providers computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

4. Rapid elasticity: Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

5. Measured service: Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the

utilized service.

Service Models

1. Cloud Software as a Service (SaaS): The capability provided to the consumer is to use the providers applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

2. Cloud Platform as a Service (PaaS): The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

3. Cloud Infrastructure as a Service (IaaS): The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

Deploy Models

1. Private Cloud: The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

2. Community Cloud: The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

3. Public Cloud: The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

4. Hybrid Cloud: The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

The definition can be summarized in visual form as Figure 3.1

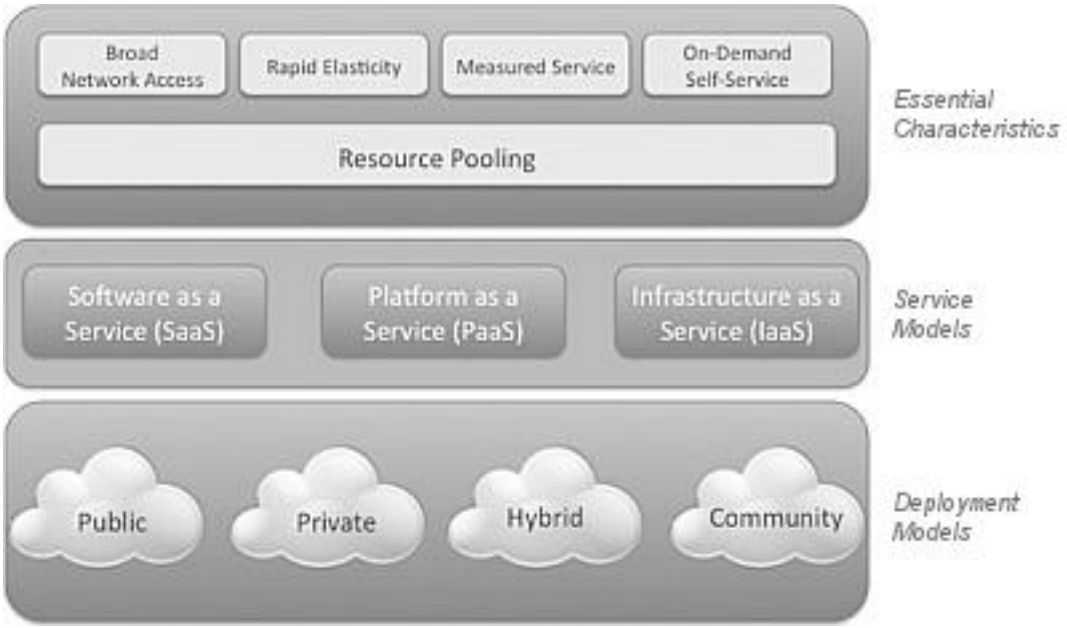

Figure 3.1: NIST-Visual Model Of Cloud Computing Definition

3.2 Benefits of Cloud Computing

Costs: The cloud promises to reduce the cost of acquiring,delivering, and maintaining power. Instead of investing in complex and expensive IT infrastructures, purchasing only the cloud services will drive down the cost.

Access: The cloud promises universal access to high-powered computing and storage resources for anyone with a network access device. By providing

such capabilities, cloud computing helps to facilitate telework initiatives, as well as bolster an agency continuity of operations (COOP) demands.

Scalability and Capacity: The cloud is an always-on computing resource that enables users to tailor consumption to their specific needs. Cloud computing allows IT infrastructures to be expanded efficiently and expediently without the necessity of making major capital investments. Capacity can be added as resources are needed and completed in a very short period of time.

Resource Maximization: Cloud computing eases the burden on IT resources already stretched thin, particularly important for agencies facing shortages of qualified IT professionals.

Collaboration: The cloud presents an environment where users can develop software-based services that enhances collaboration and fosters greater information sharing, not only within the agency, but also among other government and private entities.

Customization: Cloud computing offers a platform of great potential for creating and modifying applications to address various requirements. Its inherent agility means that specific processes can be easily altered to meet shifting agency needs.

3.3 Hadoop

Recently, many companies like IBM, Google, VMWare, Amazon, .. etc have provided products and strategies for Cloud computing. One of products which is free and trustable is Hadoop. Hadoop was introduced to the world in the fall of 2005 as part of a Nutch subproject of Lucene by the Apache Software Foundation. It was inspired by the MapReduce and Google File System originally developed by Google Labs. Hadoop is most popular as a means to classify content on the Internet (for search keywords), but it can be used for a large number of problems that require massive scalability. For example, what would happen if we wanted to grep a 10TB file? On a traditional system, this would take a terribly long time. But Hadoop was designed with these problems in mind and can make the task quite efficient. Hadoop is efficient because it works on the principle of parallelization, allowing data to process in parallel to increase the processing speed. Hadoop

is also scalable, permitting operations on petabytes of data. In addition, Hadoop relies on commodity servers, making it inexpensive and available for use by anyone. Hadoop is ideal on Linux as a production platform, with the framework written in the Java? language. Applications on Hadoop may be developed using other languages such as C++.

3.3.1 Hadoop Architecture

Hadoop is made up of a number of elements. At the bottom is the Hadoop Distributed File System (HDFS), which stores files across storage nodes in a Hadoop cluster. Above the HDFS (for the purposes of this article) is the MapReduce engine, which consists of JobTrackers and TaskTrackers. MapReduce is discussed in more detail in the coming sections.

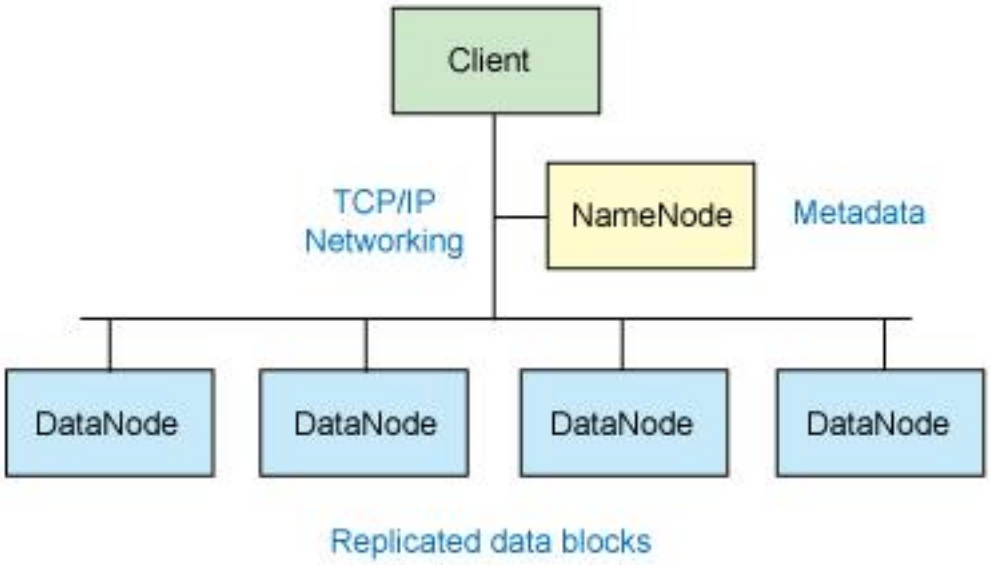

Figure 3.2: Simplified view of a Hadoop cluster

HDFS: To an external client, the HDFS appears as a traditional hierarchical file system. Files can be created, deleted, moved, renamed, and so on. But due to the special characteristics of HDFS, its architecture is built from a collection of special nodes (see Figure 3.2). These are the NameNode (there is only one), which provides metadata services within HDFS , and the DataNode, which serves storage blocks for HDFS. As only one NameNode may exist, this represents an issue with HDFS (a single point of failure). Files stored in HDFS are divided into blocks, and those blocks are replicated to multiple computers (DataNodes). This is quite different from traditional

RAID architectures. The block size (typically 64MB) and the amount of block replication are determined by the client when the file is created. All file operations are controlled by the NameNode. All communication within HDFS is layered on the standard TCP/IP protocol.

NameNode: The NameNode is a piece of software that is typically run on a distinct machine in an HDFS instance. It is responsible for managing the file system namespace and controlling access by external clients. The NameNode determines the mapping of files to replicated blocks on DataNodes. For the common replication factor of three, one replica block is stored on a different node in the same rack, and the last copy is stored in a node on a different rack. Note that this requires knowledge of the cluster architecture.

Actual I/O transactions do not pass through the NameNode, only the metadata that indicates the file mapping of DataNodes and blocks. When an external client sends a request to create a file, the NameNode responds with the block identification and DataNode IP address for the first copy of that block. The NameNode also informs the other specific DataNodes that will be receiving copies of that block.

DataNode: A DataNode is also a piece of software that is typically run on a distinct machine within an HDFS instance. Hadoop clusters contain a single NameNode and hundreds to thousands of DataNodes. DataNodes are typically organized into racks where all the systems are connected to a switch. An assumption of Hadoop is that network bandwidth between nodes within a rack is faster than between racks.

DataNodes respond to read and write requests from HDFS clients. They also respond to commands to create, delete, and replicate blocks received from the NameNode. The NameNode relies on periodic heartbeat messages from each DataNode. Each of these messages contains a block report that the NameNode can validate against its block mapping and other file system metadata. When a DataNode fails to send its heartbeat message, the NameNode may take the remedial action to re-replicate the blocks that were lost on that node.

File operations: It's probably clear by now that HDFS is not a general-purpose file system. Instead, it is designed to support streaming access to large files that are written once. For a client seeking to write a file to HDFS, the process begins with caching the file to temporary storage local to the client. When the cached data exceeds the desired HDFS block size, a file

creation request is sent to the NameNode. The NameNode responds to the client with the DataNode identity and the destination block. The DataNodes that will host file block replicas are also notified. When the client starts sending its temporary file to the first DataNode, the block contents are relayed immediately to the replica DataNodes in a pipelined fashion. Clients are also responsible for the creation of checksum files that are also saved in the same HDFS namespace. After the last file block is sent, the NameNode commits the file creation to its persistent meta data storage (in the EditLog and FsImage files).

Linux cluster: The Hadoop framework can be used on a single Linux platform (for development and debug situations), but its true power is realized using racks of commodity-class servers. These racks collectively make up a Hadoop cluster. It uses knowledge of the cluster topology to make decisions about how jobs and files are distributed throughout a cluster. Hadoop assumes that nodes can fail and, therefore, employs native methods to cope with the failures of individual computers and even entire racks. Figure 3.3 shows the work distribution in a typical cluster.

3.3.2 Hadoop Applications

One of the most common uses for Hadoop is in Web search. While not the only application of the software framework, it succinctly identifies its strengths as a parallel data processing engine. One of the most interesting aspects of this is called the Map and Reduce process, which was inspired by Google's development. This process, called indexing, takes textual Web pages retrieved by a Web crawler as input and reports the frequency of words found in those pages as the result. This can then be used through Web search to identify content from defined search parameters.

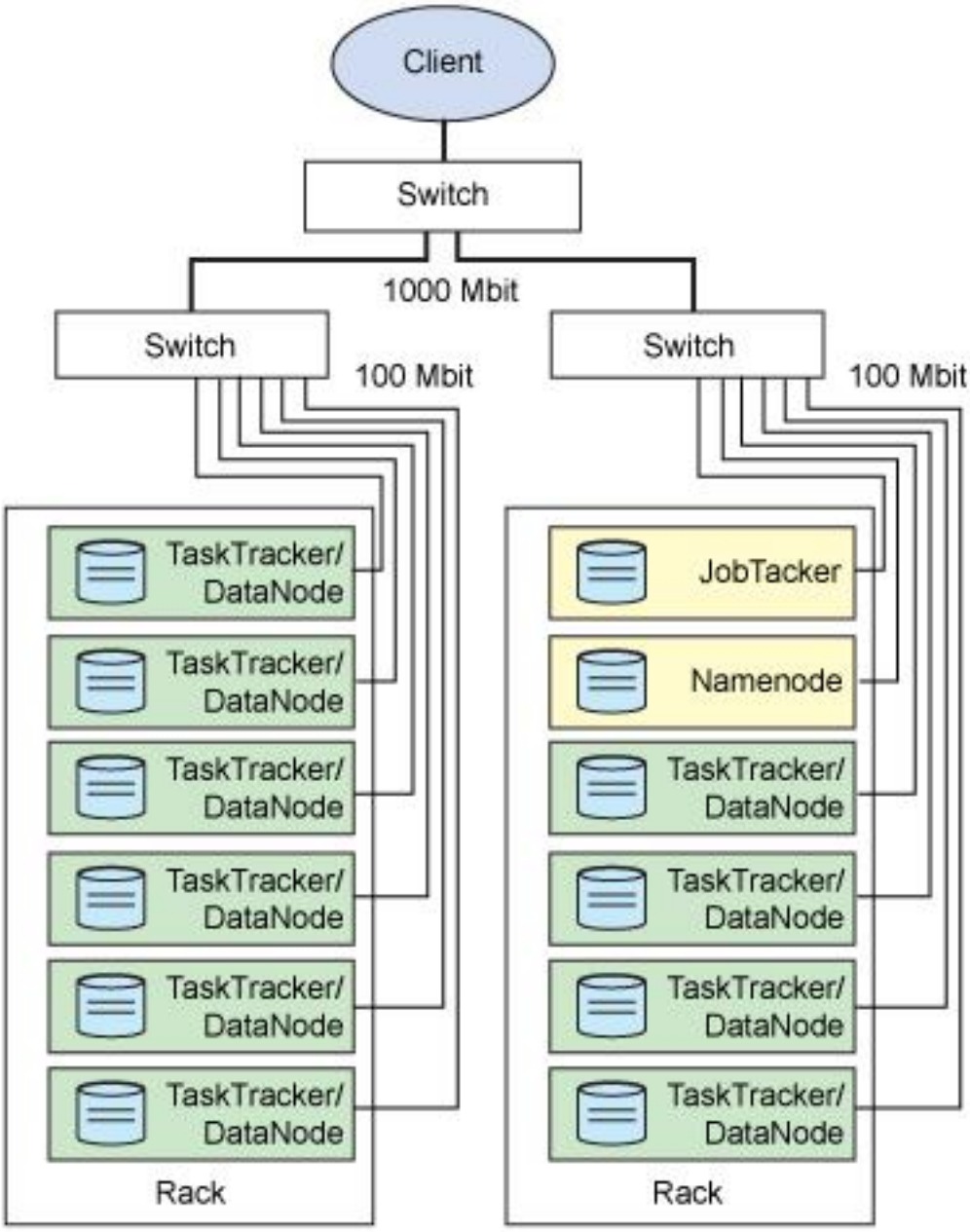

Figure 3.3: A typical Hadoop cluster

3.4 MapReduce

3.4.1 What is MapReduce?

MapReduce [25] is a “ programming model and an associated implementation for processing and generating large data sets ”. It was first developed at Google by Jeffrey Dean and Sanjay Ghemawat. Their motivation was derived from the multitude of computations that were carried out everyday in Google that involved huge amounts of input data. These computations usually happened to be conceptually straightforward. For instance, finding the most frequent query submitted to Google 's search engine on any given day or keeping track of the webpages crawled so far. But the input to these computations would be so large that it would require distributed processing over hundreds or thousands of machines in order to get results in a reasonable amount of time. And when distributed systems came into picture, a number of problems like carefully distributing the data and partitioning or parallelizing the computation made it difficult for the programmer to concentrate on the actual simple computation.

Dean and Ghemawat saw a need for an abstraction that would help the programmer focus on the computation at hand without having to bother about the complications of a distributed system like fault tolerance, load balancing, data distribution and task parallelization. And that is exactly what MapReduce was designed to achieve. A simple yet powerful framework which lets the programmer write simple units of work as map and reduce functions. The framework then automatically takes care of partitioning and parallelizing the task on a large cluster of inexpensive commodity machines. It takes care of all the problems mentioned earlier, namely, fault tolerance, load balancing, data distribution and task parallelization. Some of the simple and interesting computations for which MapReduce can be used include (see [25] for details on each).

1. Distributed Grep - finding patterns in a number of files at the same time.
2. Count of URL access frequency.
3. Reverse Web-Link Graph - Given a list of htarget, sourcei pair of URLs, finding htarget, list(source)i, i.e., finding all the URLs that link to a given target.
4. Construction of Inverted Indices from crawled web pages.
5. Distributed Sort.

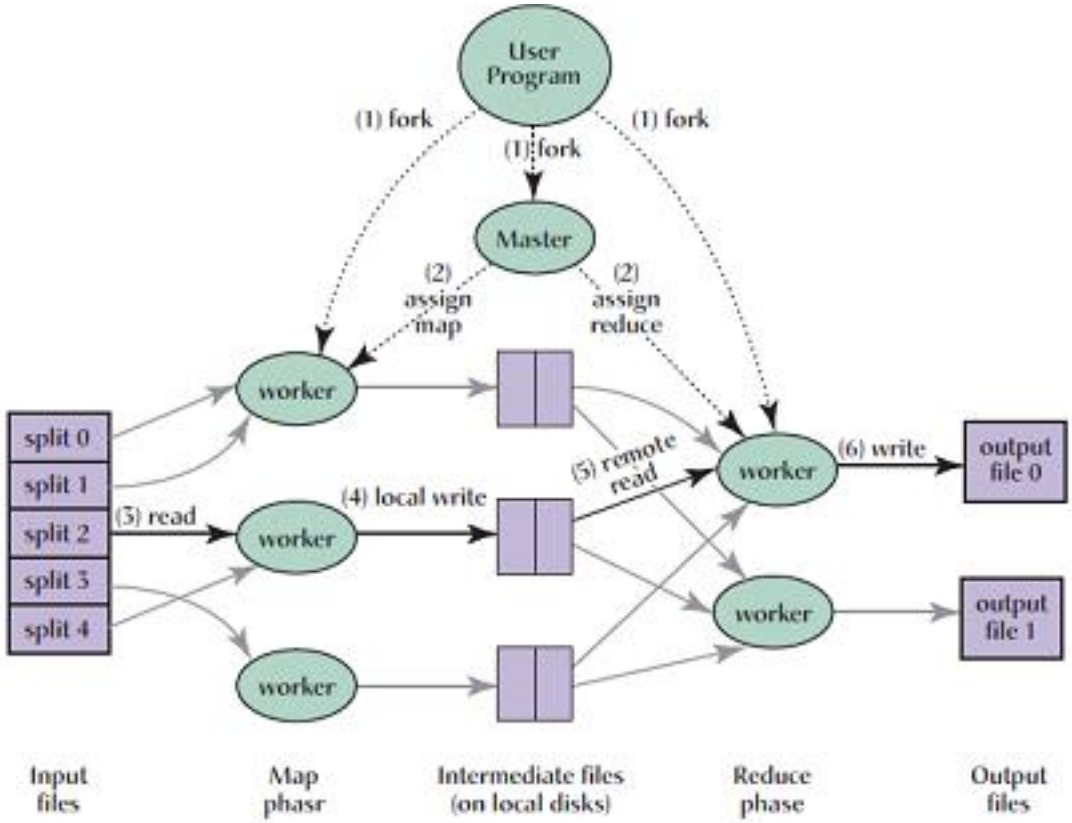

Figure 3.4: Map Reduce Execution Overview

To create a MapReduce job, a programmer specifies a map function and a reduce function. This abstraction is inspired by the ‘ map ’ and ‘ reduce ’ primitives present in Lisp and many other functional languages. The MapReduce framework runs multiple instance of these functions in parallel. The map function processes a key/value pair to generate another key/value pair. A number of such map functions running in parallel on the data that is partitioned across the cluster, produce a set of intermediate key/value pairs. The reduce function then merges all intermediate values that are associated with the same intermediate key (see figure 3.4).

$$map(k1, v1) \rightarrow k2, v2$$
$$reduce(k2, list(v2)) \rightarrow v3$$

Programs written in this functional style are automatically parallelized by the MapReduce framework and executed on a large cluster of commodity machines. As mentioned earlier, the run-time system takes care of the details

of data distribution, scheduling the various map and reduce functions to run in parallel across the set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any prior experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

3.4.2 MapReduce Breakdown

Figure 3.4 gives an overview of the MapReduce execution model. Before explaining the steps involved in a MapReduce job, lets clarify the terminology that will be used in this thesis.

Machine: an actual physical computer, which is part of a distributed cluster.

Task or Worker: a process running on a machine.

This can be thought of as a process handler. This will become more clear when we discuss Hadoop where a Node is basically a Java Daemon. Nodes run on the machines that are part of the cluster. Ideally, one machine will correspond to one node.

An entire MapReduce Job can be broken down into the following steps:
1. The MapReduce framework first splits the input data files into M pieces of fixed size - this typically being 16 megabytes to 64 megabytes (MB) per piece (controllable by the user via an optional parameter). These M pieces are then passed on to the participating machines in the cluster. Usually there are 3 copies (user controllable) of each piece for fault tolerance purposes. It then starts up many copies of the user program on the nodes in the cluster.

2. One of the nodes in the cluster is special the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. R is either decided by the configuration specified with the user-program, or by the cluster wide default configuration. The master picks idle workers and assigns each one a map task. Once the map tasks have generated the intermediate output, the master then assigns reduce tasks to idle workers. Note that all map tasks have to finish before any reduce task can begin. This is because the reduce tasks take output from any and every map task that may generate an output that it will need to consolidate.

3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to an instance of the user defined map function. The in-

termediate key/value pairs produced by the map functions are buffered in memory at the respective machines that are executing them.

4. The buffered pairs are periodically written to local disk and partitioned into R regions by the partitioning function. The framework provides a default partitioning function but the user is allowed to override this function for allowing custom partitioning. The locations of these buffered pairs on the local disk are passed back to the master. The master then forwards these locations to the reduce workers.

5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys (k2 in the definition given earlier for the reduce function) so that all occurrences of the same key are grouped together. The sorting is needed because typically many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used. Once again, the user is allowed to override the default sorting and grouping behaviours of the framework.

6. Next, the reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered , it passes the key and the corresponding set of intermediate values to the users reduce function. The output of the reduce function is appended to a final output file for this reduce partition.

7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

3.4.3 MapReduce Example

The following example 3.1 is given to understand more about MapReduce. There are map and reduce function for categorizing a set of numbers as even or odd.

```
map(String key, Integer values)
{
  //key : File Name
  //values : list of numbers
  for (each v in values) {
    if(v%2==0)
      EmitIntermediate("Even", v);
    else
      EmitIntermediate("Odd", v);
  }
}

reduce(String key, Iterator values)
{
  //key: Even or Odd
  //values : Iterator over list of numbers
  //(categorized as odd or even)

  String val = "";
  while(values.hasnext())
  {
    val=val+","+values.toString();
  }

  Emit(key, val);
}
```

List 3.1: Categorizing numbers sample

So given a list of numbers as (11,4,2,1,13,12,8,7, 9) the final output file will look like this:

Even 2,4,8,12
Odd 1,7,9,11,13

This is a very simple example where both the map and reduce function do not do anything much interesting. But a programmer has the freedom to write something a lot more complex in these functions.

3.5 Computation Model using Cloud Computing

MapReduce has been used in many computations in many fields to processing data on terabyte and petabyte scales. Many companies such as Yahoo!, Google, Amazon, and Facebook are using MapReduce. For example, MapReduce has been successfully used at Google for many different purposes. First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as MapReduce computations. For example, MapReduce is used for the generation of data for Google's production web search service, for sorting, for data mining, for machine learning, and many other systems. Third, developed an implementation of MapReduce that scales to large clusters of machines comprising thousands of machines. The implementation makes efficient use of these machine resources and therefore is suitable for use on many of the large computational problems encountered at Google.

One of advantage of Computation using cloud model is connectivity, we can connect easily to other cluster or renting some services such as Amazon EC2 at reasonable price. The EC2 service provides compute capacity for running Hadoop nodes. We can think of EC2 as a large farm of virtual machines. An EC2 instance is the AWS terminology for a virtual compute unit. Each Hadoop node will take up an EC2 instance. We rent an EC2 instance for only as long as you need and pay on an hourly basis. As of this writing, renting a compute unit with the equivalent power of a 1.0 GHz, 32-bit Opteron with 1.7 GB RAM and 160 GB disk storage costs $0.10 per hour. Using a cluster of 100 such machines for an hour will cost a measly $10.

Computation using MapReduce is scalable. The figure 3.5 shows that, when we increase the number of nodes to about three times, we can speed up the analysis of logs in 30 days of users to three times. One more case study is assist function of Yahoo Search box. For this search assist, 3 years of log-data was used. The table 3.1 shows the efficiency of using Hadoop. By this case, we can clarify the advantages of using cloud computing technology to compute bigdata.

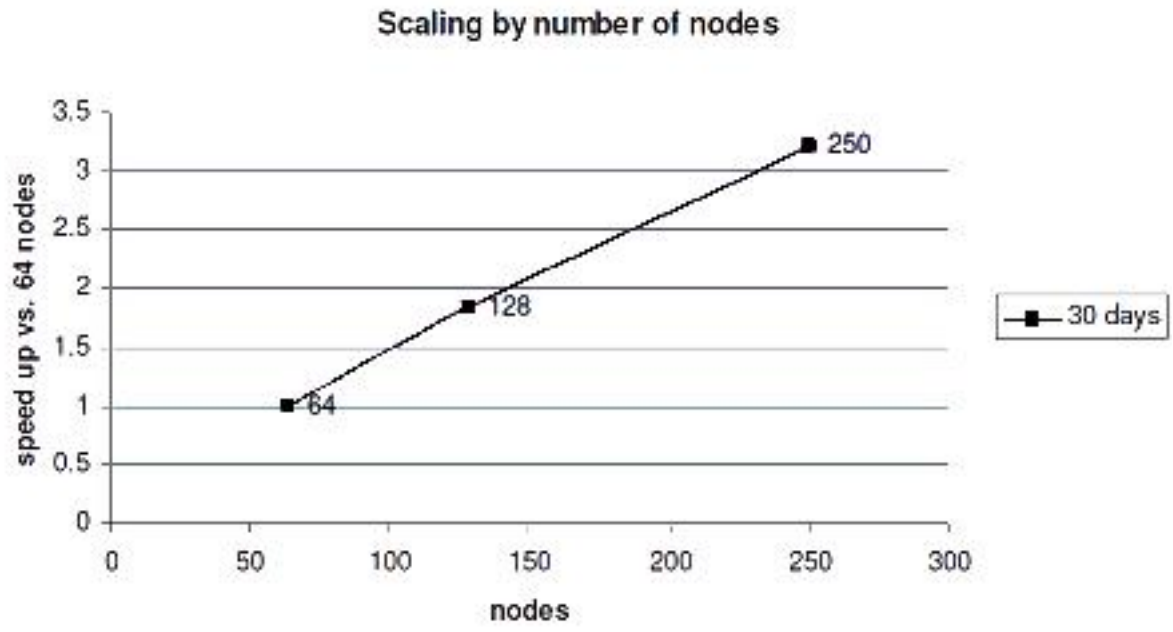

Figure 3.5: Userlog Analysis of Yahoo

	Before Hadoop	After Hadoop
Time	26 days	20 minutes
Language	C++	Python
Development Time	2-3 weeks	2-3 days

Table 3.1: Userlog Analysis of Yahoo

Chapter 4

Preserving Privacy in Large-scale Data Analysis

4.1 Privacy: Why and How

Large data-sets of sensitive personal information are becoming increasingly common no longer are they the domain only of census agencies: now hospitals, social networks, insurance companies, and search engines all possess vast amounts of sensitive data. These organizations face legal, financial, and moral pressure to make their data available to the public, but also face legal, financial, and moral pressure to protect the identities of the individuals in their dataset. Formal guarantees trading off privacy and utility are therefore of the utmost importance.

Unfortunately, the history of data privacy is littered with the failed attempts at data anonymization and ad-hoc fixes to prevent specific attacks. The most egregious of these attempts at anonymization simply scrub the dataset of obviously identifiable information (such as names), and release the remaining data in its complete form. These failed attempts at data privacy exhibit a common characteristic: a failure of imagination on the part of the data curator to account for what auxiliary information an attacker may have available, in addition to the " privately " released data set. It is this observation that will motivate differential privacy: a guarantee that provably holds independent of the attacker 's auxiliary information.

One of the first published attacks on publicly available data was due to Sweeney [42]. She attacked a dataset administered by the Group Insurance Commission, which consisted of medical records of employees of the state of

Massachusetts. All“ identifiable information ”(such as names, social security numbers, phone numbers, etc.) had been removed from this dataset, and so its release had been deemed safe. It contained only seemingly innocuous information such as birth date, gender, and zip code. However, there was another easily accessible data set that linked birthdate, gender, and zip code to names: the Massachusetts voter registration list. These three fields proved sufficiently unique in combination that they were often sufficient to uniquely identify an individual in the dataset. In particular, by cross referencing these two datasets, Sweeney was able to identify the medical records of William Weld, the sitting Governor of Massachusetts.

As more information of users are adding to the Database. Technology to preserve privacy in data has become more important. A number of techniques such as randomization and k-anonymity have been suggested in recent years in order to perform privacy-preserving. Furthermore, the problem has been discussed in multiple communities such as the database community, the statistical disclosure control community and the cryptography community. In some cases, the different communities have explored parallel lines of work which are quite similar. The key directions in the field of privacy-preserving in processing data are as follows:

Privacy-Preserving data publishing: These techniques tend to study different transformation methods associated with privacy. These techniques include methods such as randomization, k-anonymity, and l-diversity. Another related issue is how the perturbed data can be used in conjunction with classical data mining methods such as association rule mining. Other related problems include that of determining privacy-preserving methods to keep the underlying data useful (utility-based methods), or the problem of studying the different definitions of privacy, and how they compare in terms of effectiveness in different scenarios.

Changing the results of processing to preserve privacy: In many cases, the results of processing such as association rule or classification rule mining can compromise the privacy of the data. This has spawned a field of privacy in which the results of data mining algorithms such as association rule mining are modified in order to preserve the privacy of the data. A classic example of such techniques are association rule hiding methods, in which some of the association rules are suppressed in order to preserve privacy.

Query auditing: Such methods are akin to the previous case of modifying the results of data mining algorithms. Here, we are either modifying

or restricting the results of queries. Methods for perturbing the output of queries are discussed in [33], whereas techniques for restricting queries are discussed in [34, 35].

Cryptographic Methods for Distributed Privacy: In many cases, the data may be distributed across multiple sites, and the owners of the data across these different sites may wish to compute a common function. In such cases, a variety of cryptographic protocols may be used in order to communicate among the different sites, so that secure function computation is possible without revealing sensitive information. A survey of such methods may be found in [36].

Theoretical challenges in high dimensionality: Real data sets are usually extremely high dimensional, and this makes the process of privacy preservation extremely difficult both from a computational and effectiveness point of view. In [37], it has been shown that optimal k-anonymization isNP-hard. Furthermore, the technique is not even effective with increasing dimensionality, since the data can typically be combined with either public or background information to reveal the identity of the underlying record owners. A variety of methods for adversarial attacks in the high dimensional case are discussed in [38, 39].

In our research, we deal with protecting privacy of data of computation system which information leaks when a user intends to write a malicious code. Therefore privacy of inputing and outputting data is focused on. The following chapter will discuss about input privacy, output privacy and techniques used to solve the problems.

4.2 Input Privacy

The input data should be processed before sending to computation system. Most methods for protecting sensitive data use some form of transformation on the data in order to perform the privacy preservation. This reduction in granularity results in someloss of effectiveness of datamanagement or processing algorithms. This is the natural trade-off between information loss and privacy. Some examples of such techniques are as follows:

1. The randomization method: The randomization method is a technique for privacy-preserving data analysis in which noise is added to the data in order to mask the attribute values of records. The noise added is sufficiently large so that individual record values cannot be recovered. Therefore, techniques are designed to derive aggregate distributions from the perturbed records. Subsequently, data analysis techniques can be developed in order to work with these aggregate distributions.

2. The k-anonymity model and l-diversity: The k-anonymity model was developed because of the possibility of indirect identification of records from public databases. This is because combinations of record attributes can be used to exactly identify individual records. In the k-anonymity method, we reduce the granularity of data representation with the use of techniques such as generalization and suppression. This granularity is reduced sufficiently that any given record maps onto at least k other records in the data. The l-diversity model was designed to handle some weaknesses in the k-anonymity model since protecting identities to the level of k-individuals is not the same as protecting the corresponding sensitive values, especially when there is homogeneity of sensitive values within a group. To do so, the concept of intra-group diversity of sensitive values is promoted within the anonymization scheme [83].

4.3 Output Privacy

4.3.1 Data Perturbation

The problem with data perturbation is that doing it indiscriminately can have unpredictable impacts on data mining. Vladimir Estivill-Castro and Ljiljana Brankovic explored the impact of data swapping on data mining for decision rules (combinations of attributes that are effective at predicting a target class value)[41]. Full swapping (ensuring that no original records appear in the perturbed data set) can prevent effective mining, but the authors concluded that limited swapping has a minimal impact on the results. The key for privacy preservation is that you don ’t know which records are correct; you simply have to assume that the data doesn ’t contain real values.

Randomization, adding noise to hide actual data values, works because most data mining methods construct models that generalize the data. On average, adding noise (if centered around zero) preserves the data ’s statistics, so we can reasonably expect that the data mining models will still be correct. Let ’s assume we ’re building a model that classifies individuals into “ safe ” and “ unsafe ” driver categories. A likely decision rule for such a model would state that drivers between the ages of 30 and 50 are likely to be safe. Now assume we add random noise to the ages to prevent discovery of an individual ’s driving record simply by knowing his or her age. Some safe drivers between the ages of 30 and 50 will be moved into other age brackets, and some unsafe drivers who are younger or older will be moved into the 30 to 50 bracket, but the 30 to 50 bracket will also lose unsafe drivers and gain safe drivers from other age brackets. On the whole, drivers in the 30 to 50 range will still likely be safer, even on the perturbed data.

The problem is that knowing the overall values is not sufficient for building a decision tree. Data mining must also help us figure out where to make the decision points, not just the decision for those ranges, for example, why do we use 30 to 50 as the range? Why not 35 to 55 or 25 to 60? Data mining algorithms automatically find appropriate points to make such splits, but these points can be obscured by adding noise to the data. Let ’s assume the data is distributed as shown in Figure 4.1. On the original data, the natural breaks in the data fall at ages 30 and 50, so a data mining algorithm will pick these as decision points. After adding noise, the data no longer has these obvious points, so a data mining algorithm is likely to pick bad decision points and produce poor results.

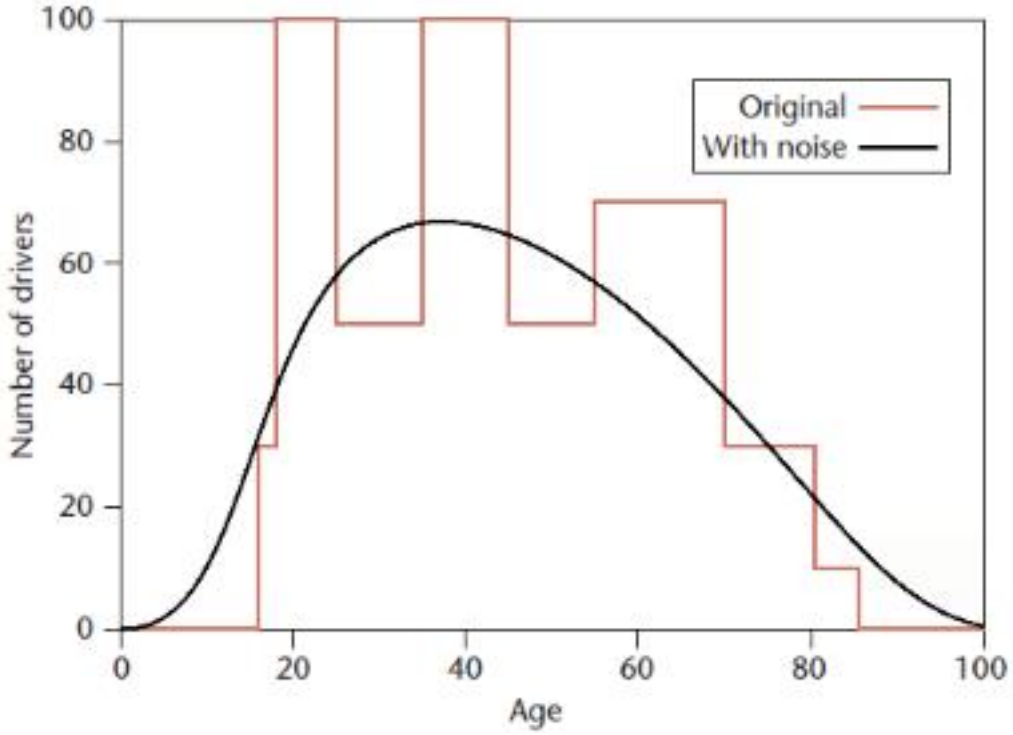

Figure 4.1: Loss of decision points in randomized data. While the data shows clear groups of people in their 20s, 40s, and 60s, these distinctions disappear when noise is added.

In a previous work[5], Rakesh Agrawal and Ramakrishnan Srikant presented a solution to this problem. Given the distribution of the noise added to the data, as well as the randomized data set, they could reconstruct the data set 's distribution (but not actual data values). This enabled a data mining algorithm to construct a much more accurate decision tree than mining the randomized data alone, which approaches the accuracy of a decision tree constructed on the real data.

Agrawal and Srikant define the reconstruction problem as follows: Given a cumulative distribution F_y and the realizations of n random samples $X_1 + Y_1, X_2 + Y_2, ..., X_n + Y_n$ estimate F_x The basic idea is to use Bayes rule to estimate the posterior distribution for a given point. For example:

$$F'_{X_i}(a) = \frac{\int_{-\infty}^{a} f_Y(w_1 - z) f_x(z) dz}{\int_{-\infty}^{\infty} f_Y(w_1 - z) f_x(z) dz} \tag{4.1}$$

To estimate the posterior distribution function F'_X given $x_1 + y_1, x_2 + y_2, ..., x_n + y_n$ the distribution functions for all of the X_i is averaged.

$$F'_X(a) = \frac{1}{n}\sum_{i=1}^{n} F'_{X_i} = \frac{1}{n}\frac{\int_{-\infty}^{a} f_Y(w_1 - z)f_x(z)dz}{\int_{-\infty}^{\infty} f_Y(w_1 - z)f_x(z)dz} \tag{4.2}$$

The corresponding density function, f'_X, can be obtained by differentiating F_X

$$f'_X(a) = \frac{1}{n}\sum_{i=1}^{n} F'_{X_i}\frac{f_Y(w_i - a)f_X(a)}{\int_{-\infty}^{\infty} f_Y(w_1 - z)f_x(z)dz} \tag{4.3}$$

Although f_X isn't really known, we can estimate it by using the normal distribution as the initial estimate and iteratively refine this estimate by applying Equation 3.3. Similar approaches have also been developed for learning association rules from randomized data, so solutions for other data mining tasks are certainly feasible. Although we won't get the same exact data mining results postrandomization as prerandomization, researchers have experimentally shown the results to be accurate enough in the case of both classification[5], and association rule mining.

One concern with randomization approaches is that the very techniques that let us reconstruct distributions also give us information about the original data values. As a simple example, let's reexamine figure 4.1. From the distribution, we don't see any drivers younger than 16. Assume we're told that the randomization was done by adding noise randomly chosen from the range $[-15, 15]$ and based on this and the noisy data set, we reconstruct the original distribution. This doesn't appear to tell us the age of any individual, a driver who is 40 years old is equally likely to have his or her age given as anything from 25 to 55, but what about an individual whose age is shown as figure 4.1 in the noisy data? We know (from the reconstructed distribution) that no drivers are younger than 16, so the driver whose age is given as figure 4.1 in the noisy data must be 16 years old.

4.3.2 Differential Privacy

Differential privacy[15, 16, 17, 18], intuitively is a computation on a set of inputs is differentially private if, for any possible input item, the probability that the computation produces a given output does not depend much on whether this item is included in the dataset or not. Formally, a computation

F satisfies (ε, σ)-differential privacy [19] (where ε and σ are privacy parameters) if, for all datasets D and D' whose only difference is a single item which is present in D but not D', and for all outputs S $\subseteq$ Range(F).

$$\Pr[F(D) \in S] \leq exp(\varepsilon) \times Pr[F(D') \in S] + \sigma$$

We can also understand this definition is as follows. Given the output of the computation, one cannot tell if any specific data item was used as part of the input because the probability of producing this output would have been the same even without that item. Not being able to tell whether the item was used at all in the computation precludes learning any useful information about it from the computation's output alone.

The computation F must be randomized to achieve privacy (probability in the above definition is taken over the randomness of F). Deterministic computations are made privacy-preserving by adding random noise to their outputs. The privacy parameter ε controls the tradeoff between the accuracy of the output and the probability that it leaks information about any individual input.

The purpose of the σ parameter is to relax the multiplicative definition of privacy for certain kinds of computation. Consider TOPWORDS, which calculates the frequency of words in a corpus and outputs the top 10 words. Let D and D' be two large corpora, the only difference is that D contains a single instance of the word " sesquipedalophobia, " while D' does not. The probability that TOPWORDS outputs " sesquipedalophobia " is very small on input D and zero on input D'. The multiplicative bound on the ratio between these probabilities required by differential privacy cannot be achieved (since one of the probabilities is zero), but the absolute difference is very small. The purpose of σ in the definition is to allow a small absolute difference in probabilities. It can be safely set to 0. Therefore, in our system, we don't use σ as an input parameter.

There are many mechanisms for achieving differential privacy. In our system, we will use the mechanism that adds noise to the output of a computation.

$f : D \rightarrow Y^k$:

$$f(x) + (R(\Delta f/\varepsilon))^k$$

where $R(\Delta f)$ is a function returns a random value from $-|\Delta f|$ to $|\Delta f|$.

Function Sensitivity: A function's sensitivity measures the maximum change in the function's output when any single item is removed from or added to its input dataset. Intuitively, the more sensitive a function, the more information it leaks about the presence or absence of a particular input. Therefore, more sensitive functions require the addition of more random noise to their output to achieve differential privacy.

The sensitivity of a function $f : D \rightarrow Y$ is :

$$\Delta(f) = Max(||f(D) - f(D')||$$

for any D,D' that are identical except for a single element, which is present in D, but not in D'. In addition, we focus only on functions that produce a single output. i.e., k = 1.

Many common functions have low sensitivity. For example, a function that counts the number of elements satisfying a certain predicate has sensitivity 1 (because the count can change by at most 1 when any single element is removed from the dataset). The sensitivity of a function that sums up integers from a bounded range is the maximum value in that range. Malicious functions that aim to leak information about an individual input or signal its presence in the input dataset are likely to be sensitive because their output must necessarily differentiate between the datasets in which this input is present and those in which it is not present.

In general, estimating the sensitivity of arbitrary untrusted code is difficult. Especially, adding sufficient random noise to the output to guarantee privacy is a big problem. If the code is malicious and attempts to output values.The computation have to add enough noise to guarantees privacy, but the results may no longer be accurate.

Sensitivity of SUM: Consider a use of SUM that takes as input 100 integers and returns their sum. If we know in advance that the inputs are all 0 or 1, then the sensitivity of SUM is low because the result varies at most by 1 depending on the presence of any given input. Only a little noise needs to be added to the sum to achieve privacy.

In general, the sensitivity of SUM is determined by the largest possible input. In this example, if one input could be as big as 1,000 and the rest are all 0 or 1, the probability of outputting any given sum should be almost the same with or without 1,000. Even if all actual inputs are 0 or 1, a lot of

noise must be added to the output of SUM in order to hide whether 1,000 was among the inputs.

Differential privacy works best for low-sensitivity computations, where the maximum influence any given input can have on the output of the computation is low.

Chapter 5

SELinux

SELinux is the primary Mandatory Access Control (MAC) mechanism built into a number of GNU / Linux distributions. SELinux originally started as the Flux Advanced Security Kernel (FLASK) development by the Utah university Flux team and the US Department of Defence. The development was enhanced by the NSA and released as open source software.

5.1 Core SELinux Components

Figure 5.1 shows a high level diagram of the SELinux core components that manage enforcement of the policy and comprise of the following:

1. A subject that must be present to cause an action to be taken by an object (such as read a file as information only flows when a subject is involved).

2. An Object Manager that knows the actions required of the particular resource (such as a file) and can enforce those actions (i.e. allow it to write to a file if permitted by the policy).

3. A Security Server that makes decisions regarding the subjects rights to perform the requested action on the object, based on the security policy rules.

4. A Security Policy that describes the rules using the SELinux policy language.

5. An Access Vector Cache (AVC) that improves system performance by caching security server decisions.

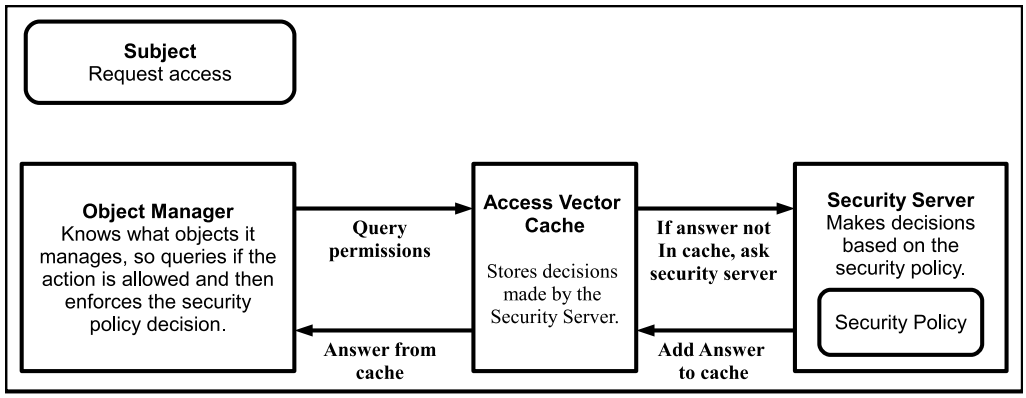

Figure 5.1: High Level Core SELinux Components

5.2 Mandatory Access Control (MAC)

Mandatory Access Control (MAC) is a type of access control in which the operating system is used to constrain a user or process (the subject) from accessing or performing an operation on an object (such as a file, disk, memory etc.).

Each of the subjects and objects have a set of security attributes that can be interrogated by the operating system to check if the requested operation can be performed or not. For SELinux the:

+ Subjects are processes.

+ Objects are system resources such as files, sockets, etc.

+ Security attributes are the security context.

+ Security Server within the Linux kernel authorizes access (or not) using the security policy (or policy) that describes rules that must be obeyed.

Note that the subject (and therefore the user) cannot decide to bypass the policy rules being enforced by the MAC policy with SELinux enabled. Contrast this to standard Linux Discretionary Access Control (DAC), which also governs the ability of subjects to access objects, however it allows users to make policy decisions.

SELinux supports two forms of MAC:

Type enforcement: Where processes run in domains and the actions

on objects are controlled by the policy. This is the implementation used for general purpose MAC within SELinux.

Multi-Level security: This is an implementation based on the Bell-La Padula (BLP) model, and used by organizations where different levels of access are required so that (for example in some defence/government systems) restricted information is separated from classified information (i.e. maintaining confidentiality). This allows enforcement rules such as ‘ no write down ’ and ‘ no read up ’ to be implemented in a policy by extending the security context to include security levels.

5.3 Type Enforcement (TE)

SELinux makes use of a specific style of type enforcement(TE) to enforce mandatory access control. For SELinux it means that all subjects and objects have a type identifier associated to them that can then be used to enforce rules laid down in a policy.

The SELinux type identifier is a simple variable-length string that is defined in the policy and then associated to a security context. It is also used in the majority of SELinux language statements and rules used to build a policy that will, when loaded into the security server, enforce the policy.

Because the type identifier (or just ‘ type ’) is associated to all subjects and objects, it can sometimes be difficult to distinguish what the type is actually associated with (it ’s not helped by the fact that by convention, type identifiers all end in ‘ _t ’). In the end it comes down to understanding how they are allocated in the policy itself and how they are used by SELinux services.

Basically if the type identifier is used to reference a subject it is referring to a GNU/Linux process or domain (i.e. domain type). If the type identifier is used to reference an object then it is specifying its object type (i.e. file type).

5.4 Constraints

Within a TE environment the way that subjects are allowed to access an object is via an allow rule, for example:

```
allow unconfined_t ext_gateway_t : process transition;
```

This is explained in more detail later, however it states that a process running in the $unconfined\_t$ domain has permission to transition a process to the $ext\_gateway\_t$ domain. However it could be that the policy writer wants to constrain this further and state that this can only happen if the role of the source domain is the same as the role of the target domain. To achieve this a constraint can be imposed using a constrain statement:

```
constrain process transition ( r1 == r2 );
```

This states that a process transition can only occur if the source role is the same as the target role, therefore a constraint is a condition that must be satisfied in order for one or more permissions to be granted (i.e. a constraint imposes additional restrictions on TE rules).

5.5 Role-Based Access Control (RBAC)

To further control access to TE domains SELinux makes use of role-based access control (RBAC). This feature allows SELinux users to be associated to one or more roles, where each role is then associated to one or more domain types as shown in Figure 5.2. Note that GNU / Linux users are not a direct part of the RBAC feature, they are associated to SELinux users via SELinux specific commands such as:

semanage login - That manages the association of GNU / Linux users (or groups of users) to SELinux users.

semanage user - That manages the association of SELinux users to roles.

Figure 5.2 shows how the SELinux user and roles are associated within the basic loadable modules that form the simple message filter.

SELinux users can be equated to groups or classes of user, for example in the Reference Policy there is $user\_u$ for general users with $staff\_u$ and $sysadm\_u$ for more specialized users. There is also a $system\_u$ defined that must never be associated to a GNU/Linux user as it a special identity for system processes and objects.

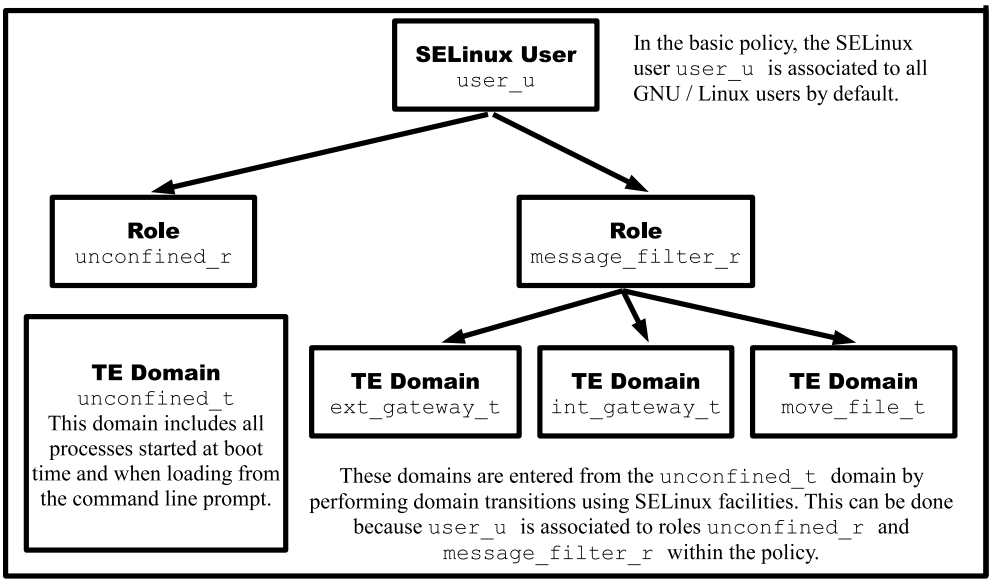

Figure 5.2: RBAC - Showing how SELinux controls access via user, role and domain type association.

5.6 Security Context

SELinux requires a security context to be associated with every process (or subject) and object that are used by the security server to decide whether access is allowed or not as defined by the policy.

The security context is also known as a ‘ security label ’ or just label that can cause confusion as there are many types of label depending on the context (another context!!).

Within SELinux, a security context is represented as variable-length strings that define the SELinux user, their role, a type identifier and an optional MCS / MLS security level as follows:

---

user:role:type[:level]

---

where:

user: The SELinux user identity. This can be associated to one or more roles that the SELinux user is allowed to use.

role: The SELinux role. This can be associated to one or more types the SELinux user is allowed to access.

type: When a type is associated with a process, it defines what processes (or domains) the SELinux user (the subject) can access. When a type is associated with an object, it defines what access permissions the SELinux user has to that object.

level: This field can also be know as a range and is only present if the policy supports MCS or MLS. The entry can consist of a single security level that contains a sensitivity level and zero or more categories and a range that consists of two security levels (a low and high) separated by a hyphen.

However note that:

1. Access decisions regarding a subject make use of all the components of the security context.

2. Access decisions regarding an object make use of all the components of the security context, however:

a) the user is either set to a special user called *system_u* or it is set to the SELinux user id of the creating process (as it serves no real purpose other than it can be used for audit purposes within logs).

b) the role is not relevant to security decisions and is always set to a special SELinux internal role of *object_r*.

Therefore for an object, the type (and level for MLS) are the only relevant security fields that are used in access decisions.

Examples of using *system_u* and *object_r* can be seen in the file system after relabeling and running the ls - Z command on various directories

Chapter 6

Program Analysis

Program analysis is the process of collecting control and data-flow information from a program. Dynamic program analysis uses the run-time behavior of a program in determining this information. Static program analysis, in contrast, derives this information solely from the text of a program and no run-time data is used. When used without qualification, program analysis refers to static program analysis. In our research we deal with static program analysis.

In our research, users write program by Java a modern object-oriented language that currently is garnering a great deal of attention within the computer programming community. One reason for this widespread attention is Java 's platform neutrality. This platform neutrality, the ability to run on virtually any computer system is possible because of the Java Virtual Machine. The Java Virtual Machine (JVM) is an abstract machine that processes an object language known as Java bytecode. Java source programs can be translated by Java compilers into bytecode programs (called class files) that are subsequently executed by the JVM. This chapter explains our approach to analyzing Java programs at the bytecode level.

One of the main attractions of the Java programming language is the ability to run its compiled code on many different system architectures. What makes this possible is the Java Virtual Machine (JVM), a very important cornerstone of Java technology.

The JVM is an abstract computing machine which can be implemented in software, microcode or directly on silicon. Like a real computer, the JVM has its own instruction set, called bytecodes, and utilizes various memory areas. It does not assume any particular implementation technology or host plat-

form. The JVM bytecodes are usually the end-product of a Java compiler. They are similar to machine code, but are not specific to any one processor. These platform-independent bytecodes are interpreted by a JVM.

```
1 public class ArrayOps
2 {
3 // Method to find the sum of an integer array
4     public static int sumOfArray(int[] theArray)
5     {
6     int sum=0;
7     for (int k=0; k<theArray.length; k++){
8         sum += theArray[k];
9     }
10    return sum;
11    } // sumOfArray
12 } // ArrayOps
```

List 6.1: Simple Java Class with sumOfArray Method

```
1 iconst_0 // Load constant 0 onto stack
2 istore_1 // Store 0 into var #1 (sum)
3 iconst_0 // Store constant 0 on stack
4 istore_2 // Store 0 into var #2 (k)
5 goto 16 // Branch to instruction at address 16
6 iload_1 // Load var #1 (sum) onto stack
7 aload_0 // Load var #0 (theArray) onto stack
8 iload_2 // Load var #2 (k) onto stack
9 iaload // Resolve "theArray[k]" and load onto stack
10 iadd // Add "theArray[k]" to "sum"
11 istore_1 // Store sum into var #1 (sum)
12 iinc 2 1 // Increment var #2 (k) by 1
13 iload_2 // Load var #2 (k) onto stack
14 aload_0 // Load var #0 (theArray) onto stack
15 arraylength // Load length of "theArray" onto stack
16 if_icmplt 7 // Branch to address 7 if "k < length"
17 iload_1 // Load var #1 (sum) onto stack
18 ireturn // Return top item on stack (sum)
```

List 6.2: JVM Bytecode Instructions for sumOfArray Method

Once a JVM has been properly developed for a given platform, the JVM can execute any bytecode program. The JVM instructions are contained in a tightly defined file format known as a class file, which is produced by a Java compiler. A Java compiler produces one class file for every Java class encountered in the source code, and the JVM is responsible for loading all related

class files upon program execution. In addition to the JVM instructions, the class file also contains an entity called the constant pool. The constant pool contains all constants that were encountered in the original source program, including a symbol table.

An example of a small algorithm written in Java, along with its corresponding JVM instructions from a class file, is shown in List 6.1 and List 6.2. List 6.1 presents a Java class with a method to find the sum of an array of integers. Line numbers are given at the start of each source line. List 6.2 presents the resulting JVM bytecode instructions as produced by Sun' s Java compiler. A brief description of each bytecode instruction is also given. To analyze the bytecode of a program, there are a lot of tools based on information control flow technology such as Soot, SemmleCode were developed.

Chapter 7

Our Solution

7.1 System Overview

The objective of our research is enabling execution of potentially untrusted Mapper and Reducer code on sensitive data. User of our system can write Mapper and Reducer code by himself and submit to computation system. The system ensures that leak through storage channels like network connections, files or leak through the output of the computation is stopped.

7.1.1 Architecture of Computation System

Our system architecture is based on Airavat system, with some modifications to raise usability. There are three entities in our model are (1) data provider, (2) user who write program to analyze datasets, (3) the computation framework. We aim to prevent malicious program from violating privacy policy of data providers by leaking information about individual item in datasets. User can write their code in the familiar MapReduce paradigm. However, processing of key and value list in reduce phase are separated since attacker can scan content of key in reducer and change processing of value list. This pattern of information flow can violate privacy policy. In addition, data providers can specify the parameters of their privacy policies.

Figure 7.1 gives an overview of our system architecture.We modified MapReduce and Hadoop distributed file system to support SELinux policy since SELinux can interrupt storage channel leaks. To prevent information leakage through output of computation, our system relies on a different privacy mechanism [10].

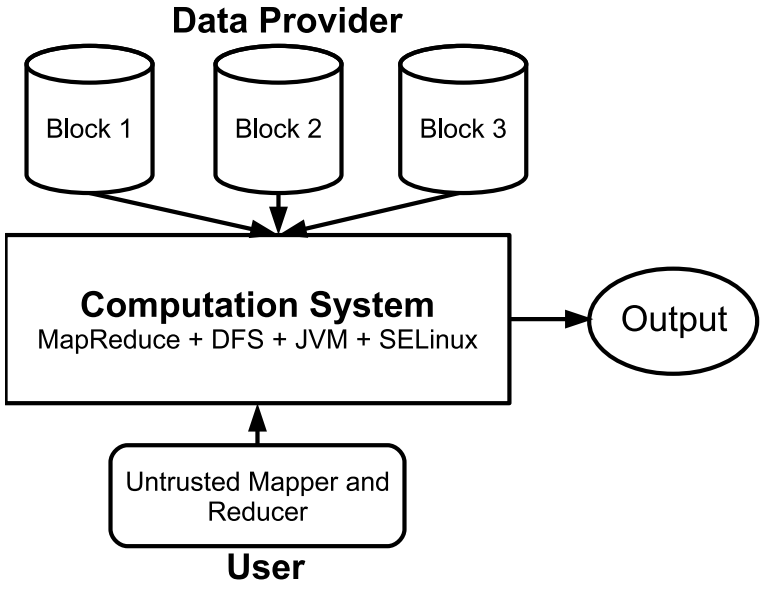

Figure 7.1: System Architecture

Table 7.1 shows the input parameters of each components. Data providers must set the value of some several privacy parameters. These parameters are part of the different privacy model. They control the trade-off between privacy and accuracy. User must to submit his code and number of outputs. Our system never outputs any keys produces by untrusted mappers and reducers. Instead, user submits a key or list keys as part of the query and our system returns (noisy) values associated with these keys. Exactly, system returns a value for every key, even if none of the mappers produced this key. Finally, the last keys are sorted and checked with sensitive keywords database. Sensitive keywords can not appear in result. This policy prevents untrusted mappers from outputting sensitive as key or signaling information by adding or removing from their output. Attacker also can not use key order as a channel to leak sensitive data. Therefore, by using this policy, we ensure that sensitive data can only be leaked through output values. The solution for this attack method will be explained later.

7.1.2 Trusted Computing Base

Our system trusts the cloud provider and cloud computing infrastructure. We assume that SELinux correctly implements access control policy. At-

Component	Input
Data provider	Privacy Parameters(ε)
User	Mapper Code(Map) Reducer Code(Red) Number of Outputs(N)
Computation system	Secured MapReduce(SMR) Secured Distributed File System(SDFS) SELinux Policy(SE)

Table 7.1: Input Parameters

tacker is malicious user who has full control over mapper code and reducer code submitted to our system. Attacker may try to access the files created by his code or to reconstruct the values of a sensitive input from outputs of the computation.

7.1.3 Processing Flow

Figure 7.2 shows processing flow of our system. After receiving mapper and reducer code of a user, our system analyzes reducer code to decide what kind of noise addition method will be applied to computation result. Byte-code pattern of functions processing a value list in reducer will be registered in pattern database of our system.

With reducer functions recognized by our system, enough noise addition method will be applied. This method ensure that, enough noise will be added to protect privacy, moreover it also raises accuracy of computation. Reducer functions which can not be detected will be processed with strong noise addition method. This method ensures privacy protecting. However, in general, accuracy of this method is worse than enough noise addition method.

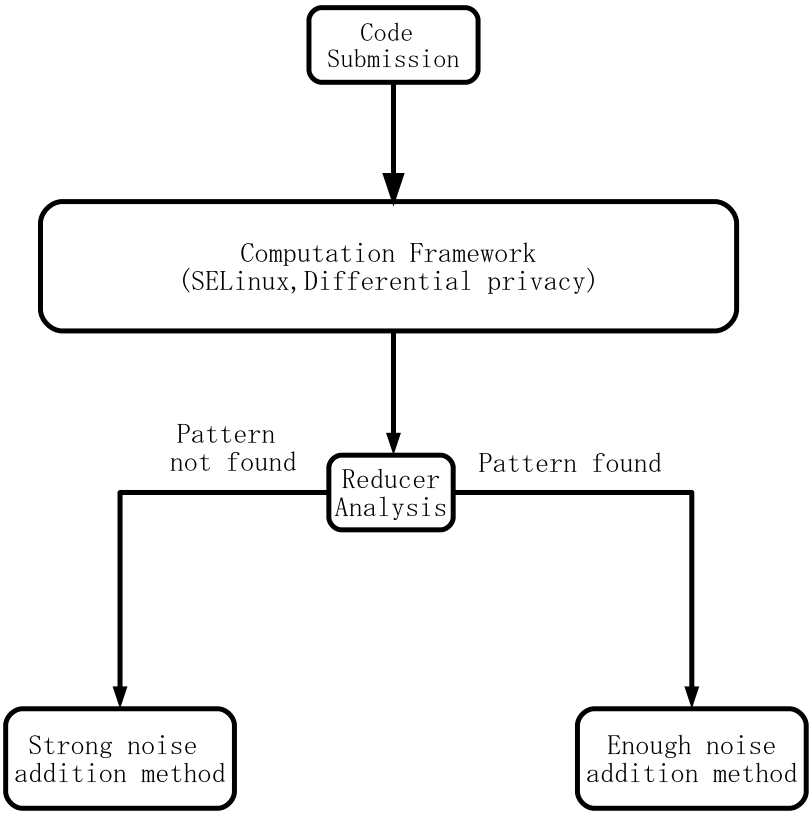

Figure 7.2: Processing flow

7.2 Mandatory Access Control

We use SELinux to create a sandbox-like environment to execute untrusted code. Moreover, it also ensures that local and HDFS files are safety from malicious users. Therefore, we created two domains for SELinux's policy. One trusted and the other untrusted. The trusted parts such as MapReduce framework and DFS execute inside the trusted domain. These processes can read and write trusted files and connect to the network. Untrusted parts, such as mapper and reducer written by a user execute in untrusted domain and have very limit permissions.

Domain	Object labeled	Detail
CST_t	Process	Trusted domain. Can access CST_*_t and common domains.
CST_exec_t	Executable	Used to transitions to CST_t domain.
CST_rw_t	File	Used to protects trusted files.
CST_notsec_t	File	Used to protect configuration files that contain no secrets. Can be read by untrusted code for initialization.
CSU_t	Process	Untrusted domain. Can access only CST_notsec_t and can read and write to pipes of CST_t domain.
CSU_exec_t	Executable	Used to transition to CSU_t domain.
CS_user	User type	Trusted user who can transition to CST_t domain.

Table 7.2: Defined SELinux domains

Table 7.2 shows the different domains and how they are used. CST_t is a trusted domain used by MapReduce framework and the distributed file system. Labels executables that launch the framework and file = system with the CST_exec_t type so the process executes in the trusted domain. This trusted domain reads and write only trusted files (labeled with CST_rw_t). No other domain is allowed to read or write these files.

MapReduce need to create configuration files that may later be read by untrusted processes for initialization. Therefore, we use CST_notsec_t domain to label configuration files. Actually, configuration files do not contain any sensitive data. MapReduce can use network connections since it's on

trusted domain.

Users who only have privilege can enter the trusted domain. We implement this restriction by creating a trusted SELinux user called CS_user. Only CS_user can execute files labeled with CST_exec_t and transition to trusted domain. System administrator has to map user who want to analyze datasets to CS_user.

The untrusted domain, CSU_t has very few privileges. A process in the untrusted domain can not connect to the network, nor read or write files. However, there are two exceptions. First, configuration files can be read. Second, communication between untrusted domain and trusted domain can be established by pipe. In different with Airavat, in our system, both mapper and reducer belong to this domain.

In contrast to Airavat, Reducer belongs to CST_exec_t. Therefore, attackers can not use reducer code to leak sensitive data through network or writing files on harddisk.

7.3 Reducer Analysis

To heighten the accuracy of computation, a reducer should be checked to detect what function it use to add enough noise. In our system, we use decompiler tools javap and jad[20] to analyze reducer. To be efficient, some optimization methods such as value numbering[43] were used.

Reducer submitted by user is compiled to Java bytecode. Because Java bytecode has limited instructions, therefore, it's possible to detect a certain pattern of Java bytecode. Moreover, we can also use patterns provided other people to detect reducer is malicious, or what kind of function it used. List 7.1 and List 7.2 show a sum function and its bytecode pattern. All of reducers which have bytecode pattern like List 7.2 will be detected as sum function. To search pattern, regular-expressions were used.

```
1 import java.util.*;
2
3 class ArrayListTest {
4   public static void main(String[] args) {
5     ArrayList list = new ArrayList();
6     list.add(new Integer(10));
7     list.add(new Integer(3));
8     list.add(new Integer(15));
9     int sum = 0;
10    for (int i = 0; i < list.size(); i++) {
11      sum = sum+(Integer)list.get(i);
12    }
13    System.out.println(sum);
14  }
15 }
```

List 7.1: Sum function source code

```
1 Loop:
2 iload_1 // Load an int value
3 iload_2 // Load an int value
4 iadd // Add two integers.
5 istore_2 // store int value into variable
6 goto Loop
```

List 7.2: Sum's bytecode pattern

In our system, we have registered bytecode pattern of five functions (sum, average, count, max, min) to pattern database. If bytecode pattern of a reducer is matched with a pattern in pattern database, enough noise will be

added to output values. Patterns of average, max, min are shows respectively in List 7.3, List 7.4, List 7.5, List 7.6.

```
1 Loop:
2 iload_1 // Load an int value
3 iconst_1 // Load an int value
4 iadd // Add two integers.
5 istore_2 // store int value into variable
6 goto Loop
```

List 7.3: Count's bytecode pattern

```
1 Loop:
2 iload_1 // Load an int value
3 iload_2 // Load an int value
4 iadd // Add two integers.
5 istore_2 // store int value into variable
6 goto Loop
7 iload_2
8 iload_3
9 idiv
10 istore_2
```

List 7.4: Averate's bytecode pattern

```
1 Loop:
2 aload_1
3 iload_2
4 if_icmple
5 aload_1
6 iload_2
7 istore_2
8 goto Loop
```

List 7.5: Max's bytecode pattern

```
1 Loop:
2 aload_1
3 iload_2
4 if_icmpge
5 aload_1
6 iload_2
7 istore_2
```

List 7.6: Min's bytecode pattern

7.4 Noise Addition Method

7.4.1 Enough Noise Addition Method

Our system analyzes reducers and add noise to the output of reducer. Adding noise ensures that privacy is protected. However, too much noise can affect to the accuracy of output. Therefore, with severral functions, enough noise will be applied. Noise of five functions (sum, average, count, max, min) is added by below algorithms.

Definitions:
Reducer input : (key,$< v_1, v_2, ..., v_n >$)
Range of value : [Min..Max]
$R(x)$: returning a random value from $-|x|$ to $|x|$.
ϵ : privacy parameter set by Data Provider

Sum function:
True output: SUM
Noise : $(1+\epsilon) \times R(Max)$
Output : SUM + $((1+\epsilon) \times R(Max))$

Count function:
True output: COUNT
Noise : $(1+\epsilon) \times R(1)$
Output : COUNT + $((1+\epsilon) \times R(1))$

Average function:
True output: SUM
Noise : $((1+\epsilon) \times R(Max))$/n
Output : SUM + $((1+\epsilon) \times R(Max))$/n

Max function:
True output: MAX
Noise : $((1+\epsilon) \times R(Max))$
Output : MAX(MIN) + $(1+\epsilon) \times R(Max))$

Min function:
True output: MIN
Noise : $((1+\epsilon) \times R(Max))$
Output : MIN + $(1+\epsilon) \times R(Max))$

7.4.2 Strong Noise Addition Method

A reducer receives a list of value associated with each key to process. In our system, key and value list are separated. Therefore, malicious user can not use key to manipulate result of value's processing. User that aim to leak information about an individual input or signal its presence in the input dataset are likely to create a strange value, for example an extremely big value as 100000 or small as -12345678. This strange value helps attacker to distinguish output of the list that it belongs to from outputs of other lists.

In MapReduce processing flow, when the mapping phase has completed, the intermediate (key, value) pairs must be exchanged between machines to send all values to reducer. The process that exchanges intermediate data is known as shuffling and sorting. Moreover, all values for the same key are always reduced together regardless of which mapper is its origin. In our system, we add a new process to check the max, min value in intermediate (key,value) pairs and frequency of each value.

** Inputs and definitions **
SV : set of values which appears 1 time.
MV : set of values which appears 2 times.
Reducer input : (key,$< v_1, v_2, ..., v_n >$)
Get(ψ) : function to get a random value from ψ set .
Red(ϕ) : reducer that user writes to process ϕ value list.
$R(x)$: returning a random value from $-|x|$ to $|x|$.
ϵ : privacy parameter set by Data Provider.
****

01. PrepareData(key,$< v_1, v_2, ..., v_n >$){
02. $\psi = < v_1, v_2, ..., v_n >$;
03. for(i=1;$i < n + 1$;i++){
04. if (v_i *in* SV) {
05. v_i = Get(ψ);
06. }
07. }
08. $\phi = < v_1, v_2, ..., v_n >$;
09. Output = RedFunc(ϕ) $\times$ (1+R(ϵ));
10. return Output;
11. }

List 7.7: Strong Addition Algorithms

Because, a value that may leak information of a specific item in datasets must be unique from other values. Therefore, as List 7.7, we replace values that appear one time by other values which frequency is bigger than two, or simply remove it. However, there's a possibility that we have many unique value. Therefore, in some case the result will be incorrect. This is a disadvantage of this method.

Chapter 8

Experiment and Evaluation

8.1 Environment

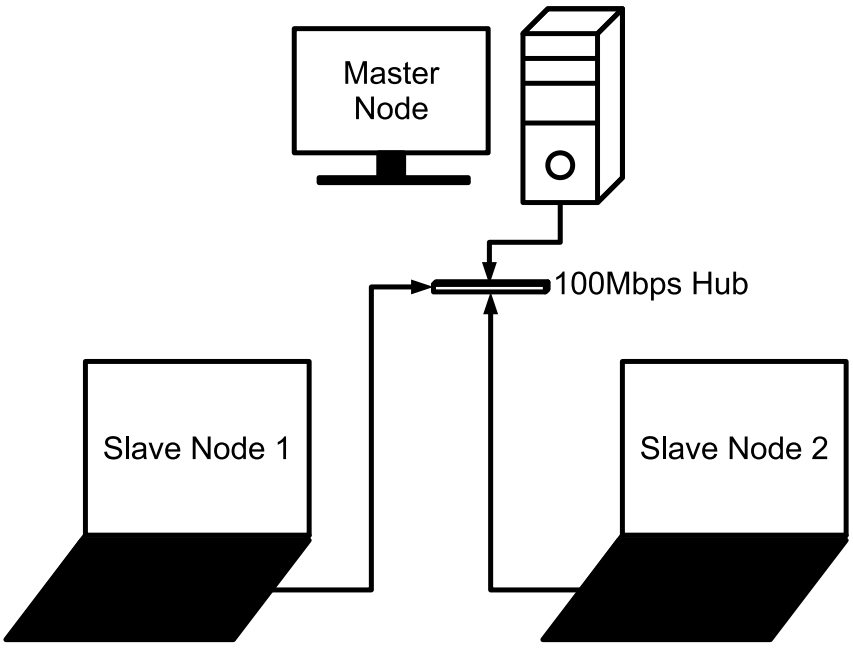

Figure 8.1: Experiment Cluster

As Figure 8.1 ,we ran all experiments on cluster made up from three normal computers:

Master Node : CeleronD 2.4 Ghz, 512Mb memory

Slave Node 1 : Core2Duo 1.8 Ghz, 512Mb memory

Slave Node 2 : PentiumM 1.5 Ghz, 512Mb memory

For safety, we set the replication number to 2 in "conf/hadoop-site.xml" file.

8.2 Experimental

To evaluate the efficiency of noise addition algorithms, we made two experiment.

AccessCount: This experiment counts the access of each links in a log file received from Nikkei[30]. The log file included the IP address, link and time. Sensitive information in this file is IP Address.

The function used in this program is sum. Therefore, our system could detect the function and applied enough noise addition method. Key in this program is a link, and the value is 1. As a result, the noise is a random number from -1 to 1. The result of this experiment is showed as Table 3.

In this experiment, user made two queries to our system. In the first time, we wrote a non-malicious program to count the access number of every links. In the second time, we inserted a condition that value in a (key,value) pair will be changed from 1 to 0 if there's an access from IP address "130.69.244.66" . Table 8.1 shows the desired result of attacker. If in the second query, the access number of a link is lower by 1, he can predict the existence of IP address "130.69.244.66" in the log file. However, by applying our algorithms, attacker can not get his desired result. The real result that attacker received is showed as Table 8.2. Since noise were added, attacker can not conclude that IP address "130.69.244.66" is exist or not. In this experiment, we ensure that privacy is protected. Moreover, difference between the real value and noise added value is less than 0.01%.

Link	First Output	Second Output
/article/COLUMN/20060224/230573/	32164	32163
/linux/index.html	25124	25123
/linux/	18579	18578
/linux/image/cover_small_1002.jpg	12152	12151
/linux/image/object_square_01.gif	12104	12103
...	...	...
...	...	...
...	...	...

Table 8.1: Desired Output Result

Link	First Output	Second Output
/article/COLUMN/20060224/230573/	32164	32163
/linux/index.html	25124	25123
/linux/	18579	18578
/linux/image/cover_small_1002.jpg	12152	12151
/linux/image/object_square_01.gif	12104	12103
...	...	...
...	...	...
...	...	...

Table 8.2: Real Output Result

Average Score: In this experiment, we compute the average score of each family name in an entrance examination of Foreign Trade University, a Vietnamese university in 2011. The data included the family name, day of birth and score. In this experiment, we removed the average function from bytecode pattern database. Therefore, strong noise addition method will be applied in this experiment. Range of value in this case is from 0 to to 30. Therefore, output result of average must be less than 30. We use the following strategy to attack the privacy:

1. A person whose family name is uncommon can be detected if attacker know the birthday.
2. Attacker replaces the value to 100000 if a specific birthday is in the data of a uncommon family name.
3. Since the number of a rare family name is small, the output result must be over 30.

In our experiment, 100000 is replaced by a random number from 0 to 30, therefore output is less than 30. We confirmed that by applying strong noise addition, the privacy is protected. However, because the person who has the unique highest score(29.5) also has a uncommon family name(Bui). Since 29.5 is a unique value, it's replaced by a random number from 0 to 30. As a result, although reducer code is not malicious, the result is different by 14.4%.

Chapter 9

Security Analysis

In our research, we assume some environment assumption and attacking method that users use to steal sensitive data as below:

9.1 Environment Assumption

Hadoop Assumption:

1. Hadoop Framework is secure. Therefore, attackers can not directly access to HDFS and get individual data. Moreover, data which are transfered to Hadoop Framework are protected.

2. User can only manage Mapper and Reducer.

3. Mapper and Reducer can access to network and storage device by writing access method code.

Network Assumption:

1. Communications among nodes participating in Hadoop are secure.

2. A user who uses computation system has network control of a normal user.

Database Assumption:

1. One sensitive data object appears only one time in database. Therefore, this object can relate to only one key in MapReduce.

2. Mapper indirectly access to database since it can receive a part of data from database from Input phase.

3. Object of computation is bigdata where amount of every value is big. Therefore, potentially malicious values that attackers created are distinguished by small amount.

9.2 Attacking Method

With environment assumption above, in Airavat and our system, below attacking method are considered:

Storage channel:
Attackers use Mapper to scan records in database and then save sensitive data they found out to storage device or send to a server using code written in Mapper. They can also send sensitive data they found out to Reducer through key or value, and also use code written in Reducer to leak sensitive data (known by presence of a specific key or a specific value) to storage device or network object.

Sensitive key output:
Attackers use Mapper to scan records in database and directly write sensitive data or encode it to a key. Then, this key is sent to Reducer. Reducer makes no change for this key or tries to encode it to a defined keyword. Finally, the key processed in Reducer is sent to output phase. Attackers can know information of sensitive data by watching the key appears in final output.

Computation output:
1. Attackers use Mapper to scan records in database and signal the information of a sensitive data by encoding it to a critical value or a defined key.
2. Then, this critical value and defined key is sent to Reducer. In Reducer phase, attackers change the function applying to value list depending on the presence of a defined value or key. Moreover, thanks to its specificity with processing function applied to value list used in Reducer, attackers can get the desired output in output phase to conclude the information of a sensitive data. To distinguish this defined value from value of other keys, attackers

try to encode to a unique value.

Breaking differential privacy:

Attackers known about different privacy, and in Mapper they try to break this technique by set value of a key to a critical value as computation output channel method. However in Mapper code, from the moment it found out sensitive data object, all value from this moment are set to the same critical value. If amount of a critical value is n. Differential privacy can only gurantee that attackers are confused to conclude this critical value appearing n times or n-1 times. Therefore, by watching the final result, if attackers can conclude the presence of this critical value, they can get sensitive data they want. As a result, the differential privacy approach is broken.

Multiple query:

1. Attacker known about unique value replacement algorithms, and tries to break this algorithms. In the first query, he sets the a value to a unique critical value α if sensitive data exists and to common value β if not exist. He does this query for many times. He always compares the output result of key with the first query. If the difference is extremely big than ϵ as we introduced in Section 7.4.2, he can concludes that β does not exist, therefore the random replacement of α each time causes the big difference.

2. Then attacker sets value to β nevertheless sensitive data exists or not. Once again, he compares compares the output result of key with the first query. If the difference is extremely big than ϵ. He can also concludes that α exist and is replaced by β.

Moreover, according to a fundamental result by Dinur and Nissim [70], the entire dataset can be decoded with a linear (in the size of the dataset) number of queries.

Multiple critical value:

Attackers known about unique value replacement algorithms, and try to break this algorithms by setting the same critical value for many sensitive data in Mapper. If amount of this value is not unique, it will be not replaced by our system. Moreover, by watching the final result, attackers can also know the amount of this critical value in value list.

9.3 Defenses Against The Attack

With environment assumption we talked in Section 9.1, default MapReduce is not unsecured. Since Mapper and Reducer are not limited, attackers can write malicious code to leak sensitive data by using storage channel, sensitive key output and computation output method.

Defenses of Airavat:
In Airavat, the following codes are added for the defenses against the attacks in 9.2.

1. The roles of SELinux are defined. They are classified as trusted and untrusted ones. Moreover, they are designed so that untrusted code does not leak information via system resources, including network connections, pipes, or other storage channels sucsh as names of running processes. In particular, an untrusted role can use only pipes for communications with Hadoop framework. A trusted role can use all privilege of a normal user in Linux.

2. All mappers are given an untrusted role.

3. Only pre-registered reducers can be executed. They are provided by the administrator of Airavat. Inside each reducer, a perturbation code for differential privacy is inserted. Moreover, they are designed so that no external channel but the standard I/O can be utilized.

4. JikeRVM is modified to make processing of a key in mapper is independent from others.

We investigate how Airavat can keep security against each attack listed in 9.2.

1. Storage channel attack:
Because of the privileges assigned to an untrusted mapper, a mapper cannot communicate with external channels but pipes. Moreover, only registered reducers can be executed, which do not communicate with any storage channel. Therefore, although attackers get a sensitive data in mapper, they can not take out this information.

2. Sensitive key output attack:
Before outputting, keys are checked by the sensitive keyword database.

Therefore, although attackers get a sensitive data in mapper, they can not write this information to the output result.

3. Computation output attack:

Airavat inherits Differential Privacy technique from PPDM (Privacy preserving data mining). Moreover, in Airavat, there are only three pre-registered reducers. Therefore, it's possible for Airavat to compute how much noise should be inserted to computation result to make attackers confused to conclude that critical value exists or not.

4. Breaking differential privacy attack:

Because Airavat executes mapper on modified JikeRVM, processing of a key in mapper is independent from others. Therefore, attackers can not use breaking differential privacy attack.

5. Multiple query attack:

Because Airavat applies differential privacy, attackers can not conclude the existence of a value in value list by watching the computation result. Therefore, although value α is replaced by any value in value list, attackers still confuse to conclude the existence of new replaced value. Moreover, for safety, from the second time query, Airavat increases enough noise compared with the previous query.

6. Multiple critical value attack:

Attackers can only conclude the existence of a sensitive data when they know that all critical values they created appears in value list. However, as we described in Section 9.2, attackers are confused to conclude the amount of critical value is n or n-1. Therefore, this attacking method is impossible in Airavat.

Airavat can block all method we considered above. However, the usability is weak because users can only use three trusted reducers provided by Airavat.

Defenses of Our System:

To improve usability weakness of Airavat, in our system, we inherit all techniques from Airavat and aim to let users write reducer by themselves. Therefore, the following parts are added to our system.

1. Users can write reducers. However, they can only write processing function for value list. Moreover, reducer is assigned to untrusted role as

mapper in Airavat.

2. Reducer is analyzed. With reducer we can detect by comparing with pre-registered reducers in our system, differential privacy is applied to insert noise.

3. Unique value replacement algorithms is implemented in undetectable reducers to replace potentially malicious unique value.

We evaluate how our system can keep security against each attack listed in 9.2.

1. Storage channel attack:
Because of the privileges assigned to untrusted mapper and reducer, mapper and reducer cannot communicate with external channels but pipes. Therefore, although attackers get a sensitive data in mapper or even in reducer, they can not take out this information.

2. Sensitive key output attack:
Because, there are no processing of key in reducers, defense of Airavat to this attack is still efficient in our system.

3. Computation output attack:
With detectable reducers, defense of Airavat to this attack is still efficient. The problem is defense of undetectable reducers. However, the threat from the existence of unique critical value in value list is removed because of enforcement of our replacement algorithms.

4. Breaking differential privacy attack:
We inherit from Airavat mapper policy indepence. Mapper is executed on modified JikeRVM which makes processing of a key in mapper is independent from other keys. Moreover, because in reducer there are no processing of key. Therefore, defense inherited from Airavat is still efficient in this attack.

5. Multiple query attack:
With detectable reducers, same as Airavat. Our system can protect privacy from this attack. However, with undetectable reducers, in many cases our system does not work well. Especially, with high-sensitivity reducers, where the influence some given input can have on the output of the computation is extremely high, the difference between the some queries is extremely

big. Therefore, privacy protecting in our system is broken.

6. Multiple critical value attack:

Since unique value replacement algorithms can only work with unique critical value, multiple critical value attack will break our algorithms. Because object of computation is bigdata which amount of every value is almost big, we can improve our algorithms by replacing values with small amount by values with common amount. However, this approach will decrease the accuracy of computation.

Chapter 10

Conclusion and Future Work

Protecting sensitive data from untrusted code is an important security challenge. Successfully solving this challenge will have the widespread effect of decreasing security breaches, increasing customer confidence in software products, and spurring the adoption of the multi-billion dollar business of cloud computing services. This dissertation presented a solution to preserve privacy in computation system using MapReduce. Compared with Airavat, usability of our proposal is better. Because, beside mapper code, a user can also write reducer by himself to analyze data. Generally, our system ensure the accuracy since in large scale computation in which the unique value almost does not exist. However, in some case accuracy is bad. Especially in the case that has a lot of unique values or attacker tries to use high sensitivity function. If difference between the output of new query and the last query is small enough, and a critical value that attackers create appears at most one time. MapReduce computation system using our proposal almost work well with attacks described in Section 9.2 except the multiple query attack.

In future research, we intend to use static code analysis tools such as Findbugs[22] and other researches[28, 29], to analyze mappers and reducers. This approach keeps computation system from executing unintentional bugs such as infinite loop to heighten the efficiency and availability. Moreover, to deal with more attacking methods, we intend to use Taint analysis to detect critical value more correct.

Acknowledgment

First of all, i would like to express my sincere gratitude to my supervisor Professor Sato Hiroyuki for guiding me throughout two years of master period, and giving me many invaluable advice at times when I needed it the most. Research is not always easy, but continuous consolation and support from my Professor helped me a lot.

I want to thank my friends who have supported me in all aspect in life. I studied a lot from them. I am also thankful to all members of my part-time company, they taught me many tips in programming.

I also like to thank Jasso and Nitori international scholarship foundation for their financial support. They gave me a chance to implement my dream in The University of Tokyo.

A final thanks to my parents, my older sister and my younger brother. They always support and console me. I could not visit my little niece on her birth in the last day of last december. However, the birth of her brings to me a lot of energy to write this thesis.

Bibliography

[1] Chief Information Officer's Council (CIO). Cloud computing: Benefits and risks of moving federal it into the cloud, 2010.

[2] Apache Hadoop. Welcome to apache hadoop, 2010. http://hadoop.apache.org/.

[3] Amazon. Elastic mapreduce, 2010. http://aws.amazon.com/elasticmapreduce/.

[4] Willi Klosgen, Anonymization Techniques for Knowledge Discovery in Databases. In Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95). Montreal, Canada. August 1995. AAAI Press, ISBN 0-929280-82-2. pp.186-191.

[5] R. Agrawal and R. Srikant. Privacy-Preserving Data Mining. In Proceedings of the ACM SIGMOD Conference on Management of Data. Dallas, Texas. May 2000. pp.439-450

[6] B. Pinkas. Cryptographic techniques for privacy preserving data mining. SIGKDD Explorations, 4(2). Dec. 2002

[7] Aggarwal C. On k-anonymity and the curse of dimensionality. In Proc. of the 31st International Conference on Very Large Data Bases (VLDB'05), Trondheim, Norway

[8] Ashwin Machanavajjhala , Daniel Kifer , Johannes Gehrke , Muthuramakrishnan Venkitasubramaniam, L-diversity: Privacy beyond k-anonymity, ACM Transactions on Knowledge Discovery from Data (TKDD), v.1 n.1, p.3-es, March 2007

[9] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In ICDE, pages 106–115, 2007.

[10] S. Hansell. AOL removes search data on vast group of web users. New York Times, Aug 8 2006.

[11] Indrajit Roy , Srinath T. V. Setty , Ann Kilzer , Vitaly Shmatikov , Emmett Witchel, Airavat: security and privacy for MapReduce, Proceedings of the 7th USENIX conference on Networked systems design and implementation, p.20-20, April 28-30, 2010, San Jose, California

[12] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In S&P, 2008.

[13] National Information Technology Laboratory
http://www.nist.gov/itl/csd/cloud-102511.cfm

[14] Se-linux. National Security Agency
http://www.nsa.gov/research/selinux/index.shtml.

[15] C. Dwork. Differential privacy. In ICALP, 2006.

[16] C. Dwork. An ad omnia approach to defining and achieving private data analysis. In PinKDD, 2007.

[17] C. Dwork. Ask a better question, get a better answer: A new approach to private data analysis. In ICDT, 2007.

[18] C. Dwork. Differential privacy: A survey of results. In TAMC, 2008.

[19] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In EUROCRYPT, 2006.

[20] Kouznetsov, P., Jad the fast Java Decompiler, http://www.kpdus.com/jad.html

[21] Preston Briggs , Keith D. Cooper , L. Taylor Simpson, Value numbering, Software Practice & Experience, v.27 n.6, p.701-724, June 1997

[22] Findbugs. http://findbugs.sourceforge.net/

[23] Java virtual machine instruction
http://www.java.sun.com/docs/books/jvms/

[24] Hadoop tutorial
http://developer.yahoo.com/hadoop/tutorial/

[25] MapReduce tutorial
http://code.google.com/intl/ja/edu/parallel/mapreduce-tutorial.html

[26] JikeRVM, http://www.jikesrvm.org

[27] J. Bloch, Effective Java Programming Language Guide. Prentice Hall, 2001.

[28] Stefan Wagner , Florian Deissenboeck , Michael Aichner , Johann Wimmer , Markus Schwalb, An Evaluation of Two Bug Pattern Tools for Java, Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation, p.248-257, April 09-11, 2008

[29] Michael S. Ware , Christopher J. Fox, Securing Java code: heuristics and an evaluation of static analysis tools, Proceedings of the 2008 workshop on Static analysis, p.12-21, June 12-12, 2008, Tucson, Arizona

[30] Nikkei IT
http://itpro.nikkeibp.co.jp/article/MAG/2010032
3/346107/itplog.zip

[31] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the Netflix Prize contenders. In KDD, 2009.

[32] A. Evfimievski. Randomization in Privacy-Preserving Data Mining. In SIGKDD Explorations, 4(2): 43-48. December 2002.

[33] Blum A., Dwork C.,McSherry F., Nissim K.: Practical Privacy: The SuLQ Framework. ACM PODS Conference, 2005.

[34] Kenthapadi K.,Mishra N., Nissim K.: Simulatable Auditing, ACM PODS Conference, 2005.

[35] Nabar S., Marthi B., Kenthapadi K., Mishra N., Motwani R.: Towards Robustness in Query Auditing. VLDB Conference, 2006.

[36] PinkasB.: CryptographicTechniques for Privacy-Preserving DataMining. ACM SIGKDD Explorations, 4(2), 2002.

[37] Meyerson A., Williams R. On the complexity of optimal k-anonymity. ACM PODS Conference, 2004.

[38] Aggarwal C. C. On k-anonymity and the curse of dimensionality. VLDB Conference, 2004.

[39] Aggarwal C. C. On Randomization, Public Information, and the Curse of Dimensionality. ICDE Conference, 2007.

[40] V. Estivill-Castro and L. Brankovic, “ Data Swapping: Balancing Privacy Against Precision in Mining for Logic Rules, ” Proc. 1st Conf. Data Warehousing and Knowledge Discovery (DaWaK-99), LNCS 1676, Springer-Verlag, 1999, pp. 389-398.

[41] V. Estivill-Castro and L. Brankovic, “ Data Swapping: Balancing Privacy Against Precision in Mining for Logic Rules, ” Proc. 1st Conf. Data Warehousing and Knowledge Discovery (DaWaK-99), LNCS 1676, Springer-Verlag, 1999, pp. 389-398.

[42] L. Sweeney et al. k-anonymity: A model for protecting privacy. International Journal of Uncertainty Fuzziness and Knowledge Based Systems, 10(5):557?570, 2002. 2.1

[43] Cliff Click, Global code motion/global value numbering, Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation, p.246-257, June 18-21, 1995, La Jolla, California, United States

[44] Aho, Alfred V., Ravi Sethi and Jeffrey D. Ullman. Compilers: Principles, Techniques and Tools, (Reading, Massachusetts: Addison-Wesley) March 1986.

[45] Aristotle Research Group. Aristotle User ’s Manual (Ohio State University, Columbus) June 1996.

[46] Arnold, Ken, and James Gosling. The Java Programming Language, (Reading, Massachusetts: Addison-Wesley) July 1997.

[47] Cifuentes, Cristina. Reverse Compilation Techniques. Doctoral Dissertation. Queensland University of Technology, Australia. July 1994.

[48] Free Software Foundation, Inc. GCC ? The GNU C Compiler. http://www.gnu.org/software/gcc/gcc.html. 59 Temple Place- Suite 330, Boston, MA.

[49] Gosling, James. Java Intermediate Bytecodes. In Proceedings of the ACM SIGPLAN Workshop on Intermediate Representations (IR ’ 95,) pages 111-118, San Francisco, CA, January 22 1995. ACM SIGPLAN, ACM Press.

[50] Gosling, James, Bill Joy and Guy Steele. The Java Language Specification, (Reading, Massachusetts: Addison-Wesley) December 1996.

[51] Harbison, Samuel P., and Guy L. Steele, Jr. C: A Reference Manual, 3rd Edition, (Englewood Cliffs, New Jersey: Prentice Hall) 1991.

[52] Harrold, Mary Jean, Loren Larsen, John Lloyd, David Nedved, Melanie Page, Gregg Rothermel, Manvinder Singh and Michael Smith. Aristotle: A System for Development of Program Analysis Based Tools, ACM 33 rd Annual Southeast Conference, March 1995, pages 110-119.

[53] Harrold, Mary Jean, and Gregg Rothermel. Aristotle: A System for Research on and Development of Program-Analysis-Based Tools. (Technical Report OSU-CISRC-3/97-TR17, Ohio State University) March 1997.

[54] International Business Machines. NetRexx. http://www2.hursley.ibm.com/netrexx. 1133 Westchester Avenue, White Plains, NY.

[55] Kempe Software Capital Enterprises. GNAT and Java. http://www.adahome.com/Resources/Languages/gnatjava.html. La Gentihommiere route d 'Essertines CH-1416 Pailly, Switzerland.

[56] Lindholm, Tim, and Frank Yellin. The Java Virtual Machine Specification, (Reading, Massachusetts: Addison-Wesley) September 1996.

[57] Northeast Parallel Architectures Center, Syracuse University. F2J: A Prototype Fortran-to-Java Converter. http://www.npac.syr.edu/projects/prcr/f2j.html. 111 College Place, Syracuse, NY.

[58] Speton, Jean-Guy, and Hejeebu Narayanarao. Java Bytecode Control Flow Graph Builder Implementation, Proceedings of the First CS569 Symposium on Software Testing and Analysis (Oregon State University, Corvallis), 1996, pages 1-7.

[59] Sreedhar, Vugranam C. Efficient Program Analysis Using DJ Graphs. Doctoral Dissertation. McGill University, Canada. 1995.

[60] Thorn, Tommy. Programming Languages for Mobile Code, ACM Computing Surveys, September 1997, pages 213-239.

[61] Wahl, Nancy J. A Conceptual Framework for Distributed Debugging. Doctoral Dissertation. Vanderbilt University, Tennessee. December 1989.

[62] Don Lance , Roland H. Untch , Nancy J. Wahl, Bytecode-based Java program analysis, Proceedings of the 37th annual Southeast regional conference (CD-ROM), p.14-es, April 1999

[63] M. Kantarcioglu and C. Clifton. Privately computing a distributed k-nn classifier. In Proc. of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), volume LNCS 3202, pages 279-290, Pisa, Italy, September 2004.

[64] A. C. Yao. Protocols for secure computations. In Proc. of the 23rd IEEE Symposium on Foundations of Computer Science, pages 160-164, 1982.

[65] A.W. C. Fu, R. C.W.Wong. Privacy-preserving frequent pattern mining across private databases. In Proc. of the 5th IEEE International Conference on Data Mining (ICDM), pages 613-616, Houston, TX, November 2005. M. Kantarcioglu and C. Clifton. Privacy preserving data mining of association rules on horizontally partitioned data. IEEE Transactions on Knowledge and Data Engineering (TKDE), 16(9):1026-1037, 2004.

[66] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In Proc. of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), pages 639-644, Edmonton, AB, Canada, 2002.

[67] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In Proc. of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), pages 206-215, Washington, DC, 2003.

[68] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In Proc. of the 5th SIAM International Conference on Data Mining (SDM), pages 92-102, Newport Beach, CA, 2005.

[69] W. Du and Z. Zhan. Building decision tree classifier on private data. In Workshop on Privacy, Security, and Data Mining at the 2002 IEEE International Conference on Data Mining, Maebashi City, Japan, December 2002.

[70] I. Dinur and K. Nissim. Revealing information while preserving privacy. In PODS, 2003.