Disclaimer: This TR is outdated. Please refer to the NSDI version of the paper!

Airavat: Security and Privacy for MapReduce

Indrajit Roy, Hany Ramadan, Srinath Setty, Ann Kilzer, Vitaly Shmatikov, Emmett Witchel.
The University of Texas at Austin
{indrajit, ramadan, srinath, akilzer, shmat, witchel}@cs.utexas.edu

Abstract

The cloud computing paradigm, which involves distributed computation on multiple large-scale datasets, will become successful only if it ensures privacy, confidentiality, and integrity for the data belonging to individuals and organizations.

We present Airavat, a novel integration of decentralized information flow control (DIFC) and differential privacy that provides strong security and privacy guarantees for MapReduce computations. Airavat allows users to use arbitrary mappers, prevents unauthorized leakage of sensitive data during the computation, and supports automatic declassification of the results when the latter do not violate individual privacy.

Airavat minimizes the amount of trusted code in the system and allows users without security expertise to perform privacy-preserving computations on sensitive data. Our prototype implementation demonstrates the flexibility of Airavat on a wide variety of case studies. The prototype is efficient, with run-times on Amazon's cloud computing infrastructure within 25% of a MapReduce system with no security.

1 Introduction

Large-scale distributed computations on data from multiple sources, enabled by programming models such as MapReduce and commonly referred to as *cloud computing*, are increasingly popular. The ultimate promise of cloud computing is based on its envisioned ubiquity: every Internet user will contribute his or her individual data and obtain useful services from the cloud. For example, a user's clickstream can be used to provide targeted advertising, while health-care applications of the future may use an individual's DNA sequence to develop tailored drugs and other personalized medical treatments.

Performing distributed computation on sensitive individual data raises serious privacy concerns. High-visibility privacy fiascoes were caused recently by public releases of anonymized individual data, such as AOL search logs [23] and the movie-rating records of Netflix subscribers [40]. The datasets in question were released to support legitimate data-mining and collaborative-filtering research, but naïve anonymization turned out to be easy to reverse in many cases.

A user of a cloud computing environment faces several threats to his or her data. For example, consider a medical patient who is deciding whether to participate in a large health-care study. First, she may be concerned that a careless or malicious application operating on her data as part of the study may expose it—for instance, by writing it into a world-readable file that will then be indexed by a search engine. Second, she may be concerned that even if all computations are done correctly and securely, the result itself (*e.g.*, the aggregate health-care statistics computed as part of the study) may leak sensitive information about her personal medical record.

These privacy concerns present an interesting challenge to the cloud computing vision. Most platforms for large-scale distributed computation are designed without special consideration for the privacy, confidentiality, or integrity of individual inputs. Yet it is unlikely that people and organizations will allow their data to be used in such computations without strong security and privacy guarantees. How to design a practical system that supports efficient distributed computations and, at the same time, assures all contributors that their privacy will not be compromised, is an important research question. In this paper, we aim to answer it.

We design and implement a practical system for large-scale distributed computation that provides rigorous privacy and security guarantees to the individual data owners whose information has been used in the computation. Our system augments MapReduce with novel protection mechanisms, which combine decentralized information-flow control (DIFC) and differential privacy. DIFC secures storage and computation done in the MapReduce framework, guaranteeing that no unauthorized data accesses or leakages occur during the processing. Differential privacy guarantees that the results of aggregate computations do not leak too much information about the individual inputs, at the cost of adding some random noise with a minor impact on the accuracy of the protected computations. DIFC and differential privacy are useful in and of themselves, but they have a further benefit when combined. Differential privacy provides a precise, mathematical basis for DIFC declassification.

Our approach minimizes the amount of trusted code in the system and enables individuals and organizations who do not have security and privacy expertise to contribute their proprietary data to large-scale MapReduce compu-

tations. At the same time, our approach assures that the privacy and integrity of their data is not violated.

Our contributions. We present Airavat,[1] a system for MapReduce computations that provides end-to-end confidentiality, integrity, and privacy guarantees using a combination of DIFC and differential privacy.

First, we demonstrate how to integrate DIFC into the MapReduce framework, paying particular attention to the division of labor between the MapReduce framework, the distributed file system and the operating system. Airavat uses DIFC to ensure that the system is free from unauthorized storage channels, such as mappers that leak data over unsecured network connections or leave it in unsecured local files.

Second, we show how to carry out privacy-preserving computations in the MapReduce framework. Airavat provides several trusted initial mappers and trusted reducers, and allows users to insert their own mappers while dynamically ensuring differential privacy.

Third, Airavat automatically ensures that as long as differential privacy holds, the classification level of a result can be safely reduced in the DIFC framework (*e.g.*, secrecy label may be removed, while keeping the integrity label).

Fourth, we implement several important data-mining and data-analysis algorithms using an Airavat prototype based on the Hadoop framework [1], executing in Amazon's EC2 compute cloud environment. Our experimental evaluation shows that the case-study algorithms, including clustering and classification, provide privacy, while producing answers that are within 95% accuracy and run times within 25% of an exact, non-privacy preserving computation.

Together, these contributions demonstrate that differential privacy can be cleanly combined with the DIFC security model and integrated into the MapReduce framework. Airavat provides a practical basis for secure, privacy-preserving, large-scale distributed computations.

2 Motivation

Recent advances in cloud computing have generated widespread user interest in storing sensitive data in cloud environments and allowing computations on this data. Krohn *et al.* argue that DIFC is the correct mechanism to use in such distributed settings, allowing users to retain control over their data while minimizing the number of trusted components [27].

MapReduce [13] is a programming model for distributed computation. It can be efficiently executed on a large number of commodity computers, as would be found in a data center or compute cloud. Because MapReduce was originally intended to run within a single data center, it has no integrated security model. Currently, running MapReduce in a compute cloud gives only the rudimentary authentication that comes with a cloud account.

DIFC is a mandatory access control model that allows data owners to protect the secrecy and integrity of data items by labeling them with appropriate protection categories. A DIFC implementation ensures that all uses of the data comply with the protection labels, thus providing end-to-end security guarantees. For example, DIFC makes it easy to enforce a policy where, after reading secret data, all of a process' output is secret. The process can no longer communicate over an unsecured network connection and any file it writes is labeled secret. Such a strong policy cannot be implemented with the discretionary access control model currently provided by compute clouds.

Access control does not address every privacy concern of potential MapReduce users. In a typical large-scale computation such as calculating health-care statistics or analyzing Web searches of thousands of users, the result of the computation depends on all inputs. If processed solely within the DIFC framework, the result will be labeled with all of the data owners' labels and cannot be made public.

Fortunately, DIFC enables the creators of a category to declassify data belonging to that category (this makes DIFC more powerful and programmable than the information flow control policies developed in the 1970s for military computers [14,24]). Of course, all instances of declassification must be audited to ensure that the declassified values do not leak too much sensitive information. While DIFC greatly reduces the amount of code that must be audited, in large-scale distributed computing scenarios, users who are not security and privacy experts may find even this limited analysis quite challenging. So when is it safe to publish the result of a computation on data originating from multiple owners?

In this paper, we use **differential privacy** to provide a framework for privacy-preserving computations, a method that is easy to use even by non-experts. Differential privacy is a powerful methodology for reasoning about and achieving privacy when computing on large datasets [16]. First, it gives a robust, composable definition of privacy. The output of a computation should not depend too much on any single input. This definition naturally bounds the amount of information the computation leaks about any of its inputs. Second, differential privacy can be achieved by adding random noise to the output of a function. The mechanisms for adding the random noise do not depend on the specifics of the function's implementation, nor (in contrast to methods such as k-anonymity) on the syntactic properties of the internal data representation.

Differential privacy has a synergetic relationship with DIFC in our system. In addition to enabling a large variety of privacy-preserving computations (whose implementations are secured by DIFC), differential privacy provides a sound basis for reducing the classification level of the resulting outputs.

Airavat is the first system to combine the access control benefits of the DIFC model with the privacy guarantees of

[1] The all-powerful king elephant in Indian mythology, known as the elephant of the clouds.

the differential privacy model. Airavat reduces the secrecy labels on the results of MapReduce computations only if it can guarantee that privacy will be preserved. Integrity labels are not affected. By contrast with the traditional DIFC model, the user does not need to write and audit custom declassifying code for each computation. It is sufficient to specify a bound on information leakage.

TCB and adversary model. Airavat trusts the underlying hardware and assumes that the OS correctly implements DIFC. For Linux, that means trusting the whole OS, but for an OS like HiStar [49] or Asbestos [46] the TCB for DIFC is smaller than the entire kernel. Airavat relies on the Java virtual machine to enforce independence of mappers provided by the user (see Section 5.2.2 for the definition and discussion of independent mappers). Airavat runs trusted exporter processes (Section 5.1.2) on all machines that access the network. These processes mediate access to the network, exactly like DStar [50]. Airvat also runs a trusted process on data storage nodes that forks off distributed file system helper processes with the appropriate labels (Section 6.1).

We assume that the adversary has full knowledge of the algorithm that a computation provider runs on Airavat. He may attempt to access the input, intermediate, and output files or to reconstruct the values of the individual inputs from the result of the computation. Apart from the TCB outlined above, Airavat is designed to withstand active adversaries, including untrusted users who supply malicious mapper computations.

After we present the technical background for Airavat (Section 3), we present an overview of its programming model (Section 4). Then, we illustrate its use in an example (Section 4.5).

3 Background

This section provides the background information on MapReduce, decentralized information flow control (DIFC), and differential privacy.

3.1 MapReduce

MapReduce [13] is a framework for performing data-intensive computations in parallel on commodity computers. A MapReduce computation reads input files that are split into multiple chunks. Each chunk is assigned to a *mapper* that reads the file as a sequence of *(key, value)* pairs, performs some computation on them and outputs a list of pairs possibly from a different domain. In the next phase, *reducers* combine the values belonging to each distinct key according to some function and write a result into an output file. The framework ensures fault-tolerant execution of mappers and reducers while scheduling them in parallel on any machine (node) in the system. In MapReduce, *combiners* are an optional processing stage before the reduce phase. They are a performance optimization, so for simplicity, our model does not support them.

We define one MapReduce *operation* as a single mapper phase followed by a single reduction phase. A MapReduce computation consists of multiple operations. MapReduce operations read their initial input from a distributed file system, and write their final output to it. Intermediate computation can use the local file system and access the network. The distributed file system uses the local file system on the compute nodes for storage. To secure a MapReduce computation, Airavat secures access to the distributed file system, the local file system and the network.

Data providers supply the input data for MapReduce computations. *Computation providers* supply the code that operates on the data.

3.2 DIFC

Decentralized information flow control (DIFC) is a mandatory access control paradigm that allows users to specify how data can propagate through a system. Using the terminology of Krohn *et al.* [26], DIFC uses *labels* to denote the sensitivity of data while *capabilities* allow the principals to acquire or drop labels. A label L is a set of tags, where tags are a compact identifier drawn from a large but arbitrary domain. We will use the term "entity" to denote either a data item, or a principal.

Each entity x has an associated secrecy label(S_x) and an integrity label(I_x). A tag t in the secrecy label S_x means that the entity either contains or has accessed information tagged with t. Similarly, a tag t in the integrity label I_x means that the entity itself is endorsed by tag t. Intuitively, the secrecy label is used to prevent data leakage while the integrity label is used to prevent data corruption.

In addition to labels, each principal p has a capability set C_p. This set determines what tags can be added or removed by the principal from its labels. If the principal has the t^+ capability, then it can add tag t to its labels. To remove a tag t from its label, the principal should have the t^- capability. Intuitively, the t^+ capability is used to *classify* or *endorse*, while the t^- capability is used to *declassify* or *de-endorse*. We will refer to C_p^+ and C_p^- as the add and remove capabilities present in C_p.

DIFC is decentralized because any principal can create a new secrecy or integrity tag. The principal obtains both the plus and minus capability for the tag when it creates a new one. Creating a new tag creates a new category of *taint* that the system propagates to entities along with the data. Traditional IFC systems only allowed system administrators to create new tags, which made the systems difficult to administer and to program effectively. DIFC systems are intended to eliminate information leaks via storage channels, but cannot eliminate timing channels [28] or probabilistic channels [43].

Label change. Formally, a principal p can change its label from L_1 to L_2 if it has the capability to add the additional tags in L_2 and can drop the tags that are not present in L_2:

$$(L_2 - L_1) \subseteq C_p^+ \text{ and } (L_1 - L_2) \subseteq C_p^-$$

Label reduction. We call any label change that causes an entity's label to become less restrictive a *label reduction*. For example, removing a security tag from a file's label is a label reduction. In traditional DIFC systems, label reductions happen in trusted, hand-audited code. In Airavat, labels can be reduced if the labeled value (typically, the result of an aggregate computation on a large dataset) satisfies differential privacy. When we say that in Airavat differential privacy is the basis for "declassification," this is really a shorthand for the more general notion of label reduction.

Flow restriction. Information can flow from an entity x **to** y if the following rule is satisfied:

$$S_x \subseteq S_y \text{ and } I_y \subseteq I_x$$

The above rule enforces the LaPadula secrecy model of *no read up, no write down* [5] and Biba's integrity model of *no read down, no write up* [7]. Notice that the flow restriction rule applies to the *current* labels of the entities. A principal may acquire or drop its labels using the label change rules so that it may perform an operation in compliance with the flow restriction (*e.g.*, a principal may remove a secrecy tag before writing to an unlabeled file).

3.3 Differential Privacy

In cloud computing, large-scale distributed computations operate on data items which originate from different sources and belong to different principals. To reason about privacy of individual pieces of data and to make safe decisions about reducing the classification level of the results of aggregate computations, we must answer the fundamental question about what it means for a computation to preserve the "privacy" of its inputs.

Airavat uses the recently developed framework of *differential privacy* to answer this question. There are several recent surveys on differential privacy [16–19]. Intuitively, a computation on a set of inputs is differentially private if, for any of its input elements, the computation produces roughly the same result whether this element is included in the input dataset or not. Formally, a randomized computation $\mathcal{F}$ satisfies ϵ-differential privacy (where ϵ is the privacy parameter) if for all datasets D and D' which differ on at most one element, and for all outputs $S \subseteq Range(\mathcal{F})$,

$$\Pr[\mathcal{F}(D) \in S] \leq \exp(\epsilon) \times \Pr[\mathcal{F}(D') \in S]$$

Here probability is taken over the randomness of the computation $\mathcal{F}$. The privacy parameter ϵ can be intuitively interpreted as the upper bound on the amount of information leaked by the computation about any of its inputs. We call ϵ the *privacy bound*.

There are several reasons why differential privacy is the right notion of privacy for cloud-computing scenarios. First, unlike other notions of privacy (briefly surveyed in Section 8), it is composable: a composition of two differentially private computations is also differentially private (of course, ϵ may increase). Second, differential privacy does not make arbitrary assumptions about the adversary. When satisfied, it holds regardless of the auxiliary or background knowledge that the adversary may possess. Differential privacy is a *relative* rather than absolute notion: it assures the owner of any individual piece of data that the same privacy violations, if any, will occur whether this piece of data is included in the aggregate computation or not. Therefore, no additional privacy risk arises from participating in the computation.

While differential privacy may seem like a relatively weak guarantee, stronger properties *cannot* be achieved without making an unjustified assumption that certain information will never be available to the adversary [16, 17]. Superficially plausible but unachievable definitions include "the adversary does not learn anything about the data that he did not know before" [12] and "the adversary's posterior distribution of possible data values after observing the result of the computation is close to his prior distribution" [21].

We emphasize that in contrast to syntactic properties such as k-anonymity (see Section 8), differential privacy is a property of the computation rather than the data. Therefore, it is a good fit for distributed-computation scenarios considered in this paper. Furthermore, it provides a sound basis for declassifying the results of aggregate computations because, if the computation is differentially private, it is guaranteed not to reveal "too much" about any of its inputs.

3.3.1 Function sensitivity

Differential privacy is intimately related to the concept of function *sensitivity*, which measures the maximum difference between the value of a function on any two inputs. For function $f : D \rightarrow R^k$, the sensitivity of f is

$$\Delta(f) = \max_{D,D'} \|f(D) - f(D')\|_1$$

for any D, D' differing in at most one element. In this paper, we will be primarily interested in functions that produce a single output, *i.e.*, $k = 1$.

Note that many common functions have low sensitivity. For example, a function that simply counts the number of elements satisfying a certain predicate has sensitivity 1. Similarly, the sensitivity of a function that sums up elements of a dataset, all of which come from a bounded range, is the maximum value in that range. Intuitively, the less sensitive a function, the less noise needs to be added to mask the dependence of its output on any input and thus achieve differential privacy.

3.3.2 Laplacian noise

There are many mechanisms for achieving differential privacy [4, 19, 20, 35]. In this paper, we will use the mechanism that adds Laplacian noise to the output of the compu-

tation:

$$f(x) + (\mathrm{Lap}(\Delta f/\epsilon))^k$$

where $\mathrm{Lap}(\Delta f/\epsilon)$ is a symmetric exponential distribution with standard deviation $\sqrt{(}2)\Delta f/\epsilon$.

3.3.3 Privacy budget

It may appear desirable to provide an absolute privacy guarantee that holds regardless of the number and nature of computations carried out on the data. Unfortunately, an absolute privacy guarantee cannot be achieved for meaningful definitions of privacy. A fundamental result by Dinur and Nissim [15] shows that the entire dataset can be decoded after at most a linear number of queries unless so much noise is added as to render the answer useless. This is a serious, but inevitable limitation of interactive privacy mechanisms. Non-interactive mechanisms either severely limit the utility of the data, or make unrealistic assumptions about the adversary's knowledge (see [18] and Section 8).

Informally, the differential privacy mechanisms we use in this paper ensure that no more than an ϵ amount of information is leaked by the computation about any single input. The bounds compose: if each of n computations has the privacy bound of ϵ, then their composition has the privacy bound of $n \times \epsilon$. By contrast, syntactic mechanisms such as k-anonymity can reveal the data completely if applied more than once [22]. When computations are done in parallel on disjoint data, only the worst bound counts.

The composability of differential privacy naturally gives rise to the concept of a "*privacy budget*" [20, 34]. The data provider sets a limit on the total permitted information leakage in advance. Each differentially private computation with a privacy bound of ϵ_i results in subtracting ϵ_i from this budget. Once the privacy budget is exhausted, no more results can be automatically declassified. The need to pre-specify a limit on how much computation can be done over a given dataset does constrain some usage scenarios. We emphasize, however, that there are no definitions of privacy that are robust, composable, and achievable in practice without such a limit.

4 Programming model

In this section, we provide an overview of how data and computation providers can use Airavat for secure, privacy-preserving computations.

4.1 Data provider

Data providers store data on a distributed file system and provide labels for the files. Data can also be labeled at a finer granularity than files using *schemas* (Section 5.1.1), which allow labeling of individual data records.

To allow automatic reduction of labels on the final output of computations that satisfy differential privacy, data providers must specify the *privacy bound*, the *privacy budget*, and the *reduction label* for their data. All of these values are public. As explained in Section 3.3, the privacy bound ϵ limits how much the result of the computation may depend on any single input.

The privacy budget is associated with the provider's dataset, as opposed to a specific computation. Each differentially private computation reduces the overall privacy budget by the privacy bound on that computation [20, 34]. Once the privacy budget is exhausted, no more automatic declassification can take place. As long as the privacy budget is not exhausted, the reduction label is removed from the labels associated with the result of the (differentially private) computation. A data provider who uses several categories of secrecy may have the result partially declassified by removing some, but not all, of the labels.

4.2 Computation providers

Computation providers (in this section called users) write the mappers and reducers executed by Airavat. In general, access control within mappers and reducers is enforced using conventional DIFC. The computation proceeds with a fixed set of labels.

If the data provider has approved the use of differential privacy on their data (*e.g.*, to automatically reduce the label of the result), Airavat restricts which mappers and reducers can be used in the computation. Airavat must know the function implemented by each reducer to add the appropriate random noise to its output and achieve differential privacy. Because it is difficult to determine the function computed by arbitrary code, Airavat supplies trusted reducers for operations that are common in data-analysis applications, such as sum and select.

Users can provide arbitrary mappers. To rely on differential privacy, however, they must specify the *sensitivity limit* of each mapper. This limit is needed to calculate the appropriate random noise added by the reducer to achieve differential privacy (Section 3.3.1). Rather than attempt to divine the sensitivity of arbitrary user code, Airavat simply takes the user-provided sensitivity limit and enforces it at runtime. If the output produced by the user-provided mapper exhibits greater sensitivity than the pre-specified limit, Airavat changes the output so that it remains within the limit. This change to the output can compromise the correctness of the mapper (since the sensitivity limit provided by the user is inconsistent with his or her code), but not the privacy guarantee given to the data owner. The details of this issue are discussed in Section 5.2.1.

4.3 Workflow

MapReduce computations in Airavat start with an Airavat-supplied mapper. The initial mapper must be trusted because it reads the data schema, which specifies how individual data records are labeled (see Section 5.1.1 for details). The initial mapper also provides a sampling of input data to allow multiple queries to be composed in parallel, which reduces the amount of noise needed to achieve differential privacy (see Section 3.3.3).

So long as a user computation starts with a trusted map-

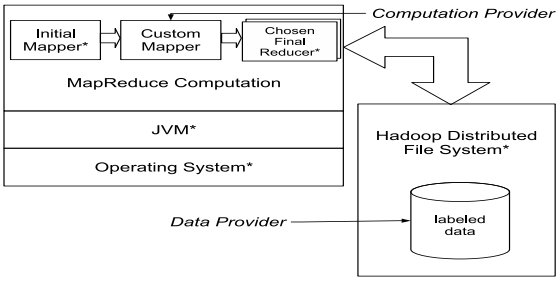

Figure 1: High level architecture of Airavat, trusted components are starred.

per and uses trusted reducers, the result of the computation can have its label reduced using differential privacy (as long as the overall privacy budget for the dataset is not exhausted). The user can supply arbitrary mapper stages that do not need to be audited. This creates a powerful and flexible computation platform that retains the provable guarantees of differential privacy.

4.4 Are differentially private results meaningful?

Some computations are difficult to make differentially private. Functions that are highly sensitive to specific inputs—for example, locating an interesting subsequence within a DNA sequence or determining the presence of a specific word in a Web page—require so much noise that the output is no longer a meaningful function of the input. In such computations, Airavat uses conventional DIFC labels to safeguard both the data and the final output.

Many common data-analysis functions have low sensitivity and can be successfully computed while satisfying differential privacy [19]. In general, if a function satisfies Lipschitz continuity (a stronger condition than regular continuity), then it can be computed in a differentially private manner with a small loss in accuracy [20].

4.5 Example: data-mining computations

As an example of Airavat in use, consider the Netflix Prize challenge, which invites users to design a better recommendation algorithm for the Netflix movie rental service. We envision contestants receiving accounts on EC2. They are allowed to participate only if they run the digitally signed Airavat versions of Linux and Hadoop, with the signatures checked by the EC2 virtual machine monitor. Figure 1 shows the high level architecture of Airavat.

The Netflix data set consists of 100M ratings by 480,000 users for 17,770 movies. One of the basic computations is to measure the average rating of a given movie. This must be done in a way that hides the rating assigned to the movie by any given user. Suppose that the privacy bound is set to $\epsilon = 0.2$. Data in Section 7 show that for a sample of movie ratings, this privacy bound can give results that are 95% accurate (for highly rated movies) compared to non-privacy-preserving computations. The actual accuracy depends upon how many users rated a movie. For movies that are heavily rated, Airavat needs to add less noise to mask the effect of any particular user hence the accuracy of the result is high. Netflix's privacy budget may impose the global limit of 5,000 computations. The reduction label is the entire secrecy label attached by Netflix to the input files, meaning that the result is made public since the secrecy label is entirely removed.

Each contestant starts their computation with a trusted mapper that extracts the relevant records from the actual dataset. Subsequent mappers provided by the contestants need not be audited. The system will add enough random noise to make the output differentially private.

Even though the contestants' mappers are not constrained, our system is able to limit the amount of information leakage to the bound provided by Netflix. Netflix does not need to provide a declassification module that must understand the results produced by each user-provided MapReduce computation. Therefore, there is no risk of declassifying too much and no cost to develop and audit such a module. Furthermore, contestants' code can operate on unmodified data (with leakages prevented by DIFC), eliminating the need to change their algorithms to work with sanitized or perturbed records.

5 Design

This section describes the design of Airavat, the semantics of DIFC in a MapReduce environment, and how Airavat provides differential privacy.

5.1 DIFC in the Hadoop infrastructure

We describe the design of making the Hadoop file system and MapReduce framework DIFC aware. First, we briefly describe the Hadoop distributed file system (HDFS).

HDFS. HDFS provides distributed and fault tolerant data storage that can run on large clusters. It is designed to be efficient for applications that work on large datasets and have write-once, read-many characteristics. Specific details on HDFS can be found at the Hadoop website [1].

An HDFS cluster consists of a single master server called *NameNode* that stores metadata associated with files and that manages the file system namespace. There are multiple *DataNode* servers that store the actual contents of the file. Each file is internally stored as multiple blocks typically 64MB in size. These blocks are replicated over multiple datanodes for fault tolerance. The NameNode responds to client requests for file system operations, including the mapping of blocks to DataNodes. The DataNodes serve read and write requests to the client (see Figure 2). To avoid the NameNode becoming a bottleneck, HDFS never allows client data to flow through it. HDFS also supports rudimentary file permission checks.

5.1.1 Labeled files

Files in HDFS are labeled with secrecy and integrity labels that are managed by the HDFS code. HDFS labels are mapped to local OS labels by each local system (Sec-

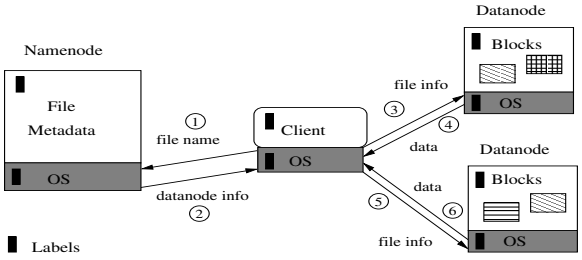

Figure 2: Simplified view of interaction of a client with the Hadoop file system. All network communication is encrypted and include labels of the participants.

tion 6.1). In a cloud computing environment, it is natural to have very large files, whose individual records have heterogeneous labels. Such files are protected by the union of the labels of the data they contain. However, most computations read only certain records from the file, and Airavat does not want to taint the computation with the complete label of the file. To alleviate this problem, Airavat uses a schema.

Schema. In Airavat, each HDFS file has an optional schema file that describes both the data layout and a mapping between fields and labels. For example, a file might have multiple lines where each line is a record consisting of two parts, a user's name and a web query string. The file might have a secrecy label, $\{S(u, q)\}$, indicating that it contains user identity information protected by secrecy label u and query strings protected by secrecy label q. However, the schema provides Airavat with enough information that, if the computation uses only query strings, then the output is labeled $\{S(q)\}$.

To use schemas, the user must specify an Airavat trusted initial mapper. The mapper first extracts the relevant parts of the data from the file and uses the schema to label this intermediate data.

5.1.2 DIFC on the network

Airavat borrows several mechanisms from DStar [50] to map the DIFC mechanisms of a single node to a networked environment. The Airavat environment is significantly simpler than DStar because it assumes that the compute nodes exist within a single administrative domain. All hosts have public/private key pairs and all hosts have access to a trusted mapping service that provides certificates mapping from network (IP) address to public key.

Exporter. On every MapReduce node and every user node, a trusted exporter process runs, whose job is to map from the network-visible label space to the OS-local label space. As in DStar, the exporter translates between the DIFC protection of the local system and the cryptographic protection of data on the network. All messages in the system are encrypted and have labels. An exporter can pass a capability to another exporter by encrypting it with their own private key (for integrity) and the public key of the recipient (for secrecy).

File operations. When a client creates a labeled file, she has to first give the associated capabilities to both the NameNode and the DataNodes where the data is kept. The capabilities allow the HDFS nodes to create the labeled files and to serve any future requests to read the file. First, the NameNode creates the file in the file system namespace. The client can then query the NameNode for a free block and stream the data to the DataNode that will hold the block. On receiving data from the client, the DataNode informs the NameNode after writing the block in its local file system. If the NameNode later replicates blocks to other DataNodes, the NameNode has to first confer the capabilities to those nodes. For file reads and writes, the client first contacts the NameNode, which may deny access if the client does not have the correct labels. If the check succeeds, the client is provided with the list of DataNodes where the files reside. In Airavat, the DataNodes also check the labels before the client may read or write the file.

5.1.3 MapReduce computation

In Airavat, the client submits a MapReduce job, providing the code for the mappers and reducers. Airavat simplifies the DIFC model for the computation by disallowing the computation from changing its labels—the computation has no capabilities. The client retains the capabilities and can use them, for example, to provide initial labels for the computation.

Eliminating capabilities from the MapReduce computation significantly decreases the complexity of the DIFC mechanisms without compromising flexibility. If a MapReduce computation needs to change its labels after some number of operations, then the user can split it into two computations, deferring label change to an optional post-processing step. When using a schema, a computation need only have the labels of the data records it reads.

5.2 Differential privacy model

For differentially private computations, Airavat requires the data provider to set the privacy bound ϵ and the privacy budget, P_B. The computation provider can write her own mapper but has to declare the sensitivity of her mapper, S_C. Airavat dynamically enforces that the sensitivity of the mapper does not exceed S_C by tracking the effect of each input to the mapper, and clamping down the output value if it would violate the declared sensitivity. While this may result in incorrect output, it preserves privacy.

5.2.1 Enforcing sensitivity

In this section we discuss how Airavat tracks and enforces the sensitivity limit. Determining the sensitivity of arbitrary code is difficult. To make it tractable, we restrict the types of reducers that can be used in the system. Airavat only allows four types of reducers—`SUM`, `COUNT`, `MEAN`, and `MEDIAN`. These reducers are trusted implementations of the functions with the same name.

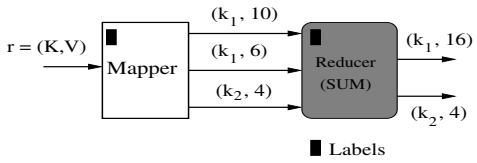

Figure 3: Example of a MapReduce operation when the reducer is a `SUM`. Trusted components are shaded.

Consider the case when the reducer is `SUM`. As shown in Figure 4, we assume that the mapper receives an input record r, and outputs a set of pairs with elements of the form (k_i, v_i). In the reduce phase, the reducer outputs the sum of values after grouping them on distinct keys. If all the keys k_i were distinct, then the input r affects the output of a key k_i by at most v_i. Note that the presence or absence of r can change the output sum of k_i by v_i. According to the declared sensitivity limit, an input can change the output by at most S_C. So Airavat ensures that $v_i < S_C$, otherwise it changes v_i to be a number less than S_C before the final reducer step.

To measure the effect of each input record, Airavat has a *sensitivity meter* associated with each mapper. The sensitivity meter tracks the sensitivity using a data structure `SM(r, ki)`. Intuitively, `SM(r, ki)` is the effect of the input record r on the final output associated with key k_i. In the above example, `SM(r,ki)` $=v_i$.

For each output (k_i, v_i) of a particular input r, the sensitivity meter updates its state using the following rule:

$$SM(r, k_i) = SM(r, k_i) + g(v_i)$$

Here, g depends on the type of the reducer. For the reducer `COUNT`, g is defined as the constant function $g(x) = 1, \forall x$. For the remaining reducers, g is the identity function.

Example. Consider the case where the input consists of a single record r, the program contains one mapper, and it outputs $\{(k_1, 10), (k_1, 6), (k_2, 4)$. If the reducer is anything apart from `COUNT`(figure 3), then the sensitivity with respect to k_1 is 16 $(10 + 6)$, and 4 for k_2. Intuitively, the presence or absence of $k_1(k_2)$ affects the output by at most 16(4). If the reducer type is `COUNT`, then sensitivity of this particular execution is 2 for key k_1 and 1 for k_2. `COUNT` only measures the cardinality of a set.

Privacy groups. Definitions of differential privacy apply to each record individually and provide privacy guarantees for each record. They determine the sensitivity of a function by measuring the effect of a record's presence or absence. However, to provide privacy guarantees that are meaningful in practice, it is sometimes important to group data and calculate sensitivity with respect to the group. For example, in the AOL dataset each record is a search made by a particular user. For this dataset, we would like our privacy guarantee to apply to the user, not to an individual record.

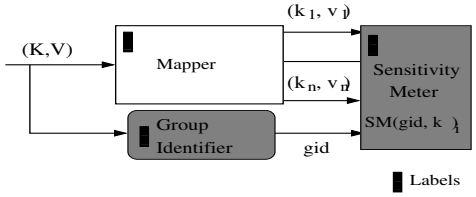

Figure 4: Details of the input used by the sensitivity meter. Trusted components are shaded.

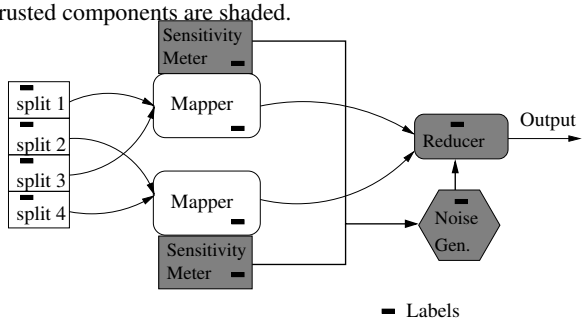

Figure 5: Simplified overview of sensitivity calculation and noise generation for a given MapReduce computation. Trusted components are shaded.

Airavat provides the ability to group records and to thus guarantee differential privacy to groups. The user can provide a program that takes a record as input and emits the group id, `gid`, for that record. This code also executes with the labels of the computation to prevent it from leaking data. We modify the sensitivity meter to track the effect of each group instead of just a record. As shown in figure 4, the sensitivity meter uses gid as an index in the data structure `SM(gid,ki)`.

Providing privacy for each record is a special case where each record has its own group identifier. Airavat assumes that each record belongs to at most one group.

Aggregating sensitivity from multiple mappers. A single MapReduce operation may run mappers on many different machines. Input elements from the same privacy group may go to different mappers. Airavat must therefore merge the state of multiple sensitivity meters. In the reduce phase, each reducer fetches the data from all the sensitivity meters. The reducer merges them and finds the maximum sensitivity amongst all groups for each key. If this value is less than the declared bound S_C, then the reducers can reduce the label of the result after adding noise generated as Lap(S_C/ϵ). If the sensitivity is exceeded, then Airavat forces the final reducer to pick values less than S_C instead of the value provided by the mapper. Figure 5 gives a overview of the how the sensitivity enforcement is integrated with the MapReduce framework.

5.2.2 Trusted reducers and independent mappers

To enforce the sensitivity bound, we must limit the effect that any input can have on the output. There is no way to determine a reasonable bound on the effect, for map-

pers and reducers that are complete black boxes. Instead, we impose restrictions on the mappers and the reducers. Airavat provides a small set of trusted reducers (e.g., SUM) whose sensitivity is known.

For mappers, Airavat must ensure that each invocation of a mapper is *independent* from the effects of any previous *(key,value)* pairs from the input. Independence ensures that the dynamic sensitivity calculation does not underestimate the sensitivity of an input record. For example, Airavat disallows the case where a mapper stores a pair (k_1, v_1) for input r_1 and then uses v_1 when computing the output for input record r_2. Then the effect of r_1 is not just v_1, but also depends on the output corresponding to r_2. Calculating sensitivity for non-independent mappers is difficult, so Airavat makes sure that each invocation of a mapper is independent. As described in Section 6.4, we modify the JVM to dynamically detect whether a mapper is independent or not. For non-independent mappers, Airavat does not reduce the label of the output.

The initial mapper is trusted and can be used to partition or select data from the input files. This mapper need not be independent because it outputs labeled files that are then used as inputs to the main computation.

What can Airavat compute privately? Given the restrictions on the mapper and reducer functions, it is natural to ask which computations can Airavat execute with differential privacy. Airavat is powerful enough to allow differentially private computation of *all* algorithms in the statistical query model [25], including various data mining algorithms such as k-Means, Naive Bayes, principal component analysis, and linear classifiers such as perceptrons.

We demonstrate our claim by showing that Airavat can implement the SuLQ primitive introduced by Blum et. al. [8]. In that paper, Blum et. al. show that the SuLQ primitive can be used to compute any algorithm in the statistical query model. Algorithm 1 depicts the structure of the SuLQ primitive. The user makes a query g to the database that returns a noisy version of the result. Figure 6 gives the corresponding MapReduce operation in Airavat for this SuLQ primitive. Intuitively, the mapper evaluates the predicate g on each data record while the reducer sums the values and adds the appropriate noise. Thus, any algorithm that can be written in the SuLQ framework using this primitive can be computed in a distributed and parallelized fashion using Airavat.

Algorithm 1 SuLQ Algorithm A(R)

Require: a query (g:D $\rightarrow$ [0,1])
Return $\sum_i g(d_i) + Noise$

5.2.3 Multiple MapReduce operations

Sometimes a single computation may involve a pipeline of multiple MapReduce operations. In such cases, the system needs to determine the effect of the initial data on the final output after it has undergone multiple transformations through the different MapReduce operations. Loosely speaking, the sensitivity of such a composition can be estimated by the sensitivity of the final MapReduce operation.

```
map(Key k1, Val v1){
  ...
  output (k2, g(v1))
}
reduce(Key k2, List l2){
  ...
  for e in l2:
      newVal.add(e)
  newVal.add(Airavat.getNoise(ϵ, S))
  output(newVal)
}
```

Figure 6: Pseudo code of how to implement the SuLQ primitive in Airavat.

To simplify the discussion let us assume that a mapper outputs only one *(key,value)* pair for a given input and the reducer is SUM. Abusing notation, a MapReduce operation on input set T can be thought of as

$$H_1(T) = \sum_{x \in T}{}^{k} f_1(x)$$

Here the mapper is the implementation of f_1 while $\sum^k$ is the reducer that groups the output of the mapper by keys, k, and sums the individual values. The result of applying H_1 on set T is a set of elements, $H_1(T)$, not necessarily just a single number.

A pipeline of two MapReduce operations, H_1 followed by H_2, where the input is the set T_1 is the composition $H_2(H_1(T_1))$. Expanding, we can write the final output set O as:

$$O = \sum_{y \in T_2}{}^{k} f_2(y) \text{ where } T_2 = \sum_{x \in T_1}{}^{k} f_1(x)$$

If f_2 is bounded by B then the sensitivity of the composition is at most B. Notice that the presence or absence of any element in T_1 can affect the final output by atmost B. This bound is not tight because T_2 may be a restricted domain for f_2 such that the value B is never achieved.

If the mapper in H_1 can output p_1 different keys for any one input record the sensitivity of the composition is $B \cdot p_1$. Intuitively, the p_1 different keys may later on get combined in H_2 which will contribute at most $B \cdot p_1$ in the final output. Generalizing this, if there are n MapReduce operations in the pipeline and a mapper in operation i emits a maximum of p_i keys on any input and the final mapper function is bounded by B, then the sensitivity of the composition is at most $B \cdot \prod p_i$.

5.3 Handling timing channels

A malicious user may attempt to leak data by introducing mappers that act as timing channels. For example, a mapper code could take longer to output data if it has read

sensitive data. To handle these cases, Airavat requires the computation provider to give an estimate of the execution time. The final output is delayed to match this estimate.

[iR FIXME: If the computation does not finish, what value do you output? Btw, the Amazon account logs will show the resources used including the time.]

6 Implementation

Airavat consists of custom modifications to the Hadoop MapReduce framework, Hadoop file system (HDFS), and Hadoop network protocols. It includes a custom JVM for running user-supplied mappers , certified reducers, and a kernel module to add DIFC support to the Linux operating system. This section describes how we modified Hadoop to create Airavat. In our prototype, we modify $4,000$ lines of code in HDFS, $2,000$ lines of code in the MapReduce framework and 500 lines of code in the JVM.

6.1 Hadoop distributed file system

We have modified HDFS to support DIFC. We require that the NameNode and the DataNodes run on a DIFC-aware OS. We use the Laminar OS [42] in our prototype.

We modified the HDFS `inode` structure to hold the label of the file. Inodes are stored in the NameNode server. Any request for a file operation by a client is validated against the inode label. These checks consist of the rules given in Section 3.2. In the DataNodes, we modified the block information structure to contain the HDFS label of the file to which the block belongs. Each block is stored in the local file system protected by the OS labels. The block information contains this mapping of the HDFS label to the local OS label. The DataNode daemon is trusted and acquires the OS labels needed to read a block, by using these mappings.

HDFS exports a hierarchical namespace. We enforce the rule that secrecy increases with depth, while integrity decreases. As in Flume [26], when a client creates a labeled file, she first creates an empty file with the same labels, taints herself with the label to acquire private data, and then writes it to the newly created file in HDFS. This technique prevents a client from leaking information through the file name. As described in Section 5.1.3, the mappers and reducers in Airavat are not given capabilities to add or drop labels. Hence they cannot use the method of pre-creating files and then tainting themselves with a label. To circumvent this problem, Airavat first creates the MapReduce output directory (`${mapred.output.dir}` in Hadoop) when the computation is launched. This output directory is protected by the labels of the computation. Any mapper or reducer can write labeled intermediate or final output files inside this directory without having to pre-create those files.

6.2 MapReduce framework

In Airavat, the MapReduce computation runs with the labels that are provided by the client when the computation is launched. The client can only confer those labels for which she has the add capability. At a high level, the MapReduce framework reads the list of input files, splits the data into chunks and spawns mappers to process each chunk. We modified the framework so that it uses labels while interacting with the HDFS to read and write files.

Intermediate outputs from the map phase are written into labeled files. This use of labeled files ensures that only principals with the correct labels can read them. If a mapper or reducer only processes a particular record of the input, the intermediate labels might be shorter (less restrictive) than the labels of the original input. After the map phase, the reducers fetch the output from the mappers via the HDFS. These read requests are validated against the labels of the files.

6.3 Determining sensitivity and noise

As described in Section 5.2.1, each mapper has an associated sensitivity meter. The sensitivity meter keeps track of the sensitivity for each group and key pair, `SM(gid,key)`. For every `(input_key,value)` pair the `SM(gid,key)` is updated by the rule in Section 5.2.1. In Hadoop, the `MapRunner` class invokes the map function. This class calls the trusted code to determine the `gid` of each input record.

For efficiency, we use a hash table to implement the data structure `SM`. The size of the hash table is pre-determined and fixed. So, the sensitivity can be inflated by false collisions if there are a very large number of groups. Note that this is not a correctness problem and does not break the guarantees of differential privacy. Airavat might, however, falsely conclude that the mapper breaches the declared sensitivity bound of S_C. In the degenerate case where each record belongs to a group of its own, the sensitivity meter can be simplified to keep only the state `SM(key)`. We simply track the record that has the maximum effect for a given key. The modified update rule, after observing each output of the form (k_i,v_i) from different inputs, is $\texttt{SM}(k_i) = \max\{\texttt{SM}(k_i), \texttt{g}(v_i)\}$.

At the completion of the map task, each mapper writes the data structure `SM(gid,key)` to a labeled file. Each reducer then fetches the `SM(gid,key)` from all mappers and merges them together. The reducer then calculates the maximum sensitivity over all groups S_g and determines if this sensitivity is within the pre-defined bound S_C. The noise is calculated from the Laplace distribution, Lap(S_C/ϵ). If the calculated sensitivity exceeds S_C, then the reducer first chooses a value within the S_C sensitivity limit before combining values.

6.4 Ensuring independent mappers

While calculating the sensitivity of the input data, we assume that each invocation of the mapper function is independent from the effects of any previous input data. A mapper is stateful if it writes a value to storage during a map invocation and then uses that value in a later invocation. To

ensure that the mapper invocations are independent, Airavat must track accesses to files, the network and memory objects.

For files and the network, we use the simple rule that a map function is not allowed to access them. These checks are enforced by the operating system. Note that we allow the mapper to access the files and the network during its initialization. Mappers can override the `configure()` function to initialize themselves. This function is executed once after the mapper is loaded, using Java reflection.

For memory objects, we add access checks to two types of data: *objects*, which reside on the heap, and *statics*, which reside on the global pool. We had to modify the Java virtual machine to enforce these checks. In our prototype, we use Jikes RVM 3.0.0 [2], a Java-in-Java research virtual machine. Airavat prevents mappers from writing static variables. This restriction is enforced dynamically by using write barriers that are inserted whenever a static is accessed. Next, we modified the object allocator to add a word to each object header. This word points to a 64-bit number called the `invocation number (ivn)`. The Airavat JVM inserts read and write barriers for all objects. Before each write, the `ivn` of the object is updated to the current invocation number. Before a read, the JVM checks if the object's `ivn` is less than the current invocation number. If so, then the mapper is assumed to be stateful and a corresponding exception is thrown. After this exception, the current map invocation is re-executed. The final output of the MapReduce computation is labeled and can only be declassified using traditional DIFC methods.

Jikes RVM is not mature enough to run code as large and complex as the Hadoop framework. We therefore use the Hadoop Streaming feature to ensure that mappers run on Jikes while the remaining part of the framework executes on Sun JVM. The streaming utility forks a trusted Jikes process that loads the mapper using reflection. The Jikes process then executes the map function for each input provided by the streaming utility. The streaming utility communicates with the Jikes process using `stdin` and `stdout`. This communication is secured by the standard DIFC pipes provided by the operating system.

7 Evaluation

In this section, we evaluate the performance and accuracy of different MapReduce computations run using Airavat. Table 1 provides an overview of the benchmarks used. K-Means and Naive Bayes use the public implementations from Apache Mahout[3]. These experiments show that computations in Airavat have a low overhead, less than 25% compared to those running on unmodified Hadoop.

7.1 Airavat overheads

The experiments were run on Amazon EC2 with a cluster instance of 20 machines. This machine limit is imposed by Amazon for basic service users. EC2 also does not allow customers to run their own kernels, so we could not use the DIFC OS. Therefore, we also ran identical benchmarks on a 16 core local machine, with the DIFC OS. Overheads incurred due to the OS was less than 1%, across all the benchmarks. The benchmarks had to be adapted to work with Hadoop Streaming (as mentioned in the previous section).

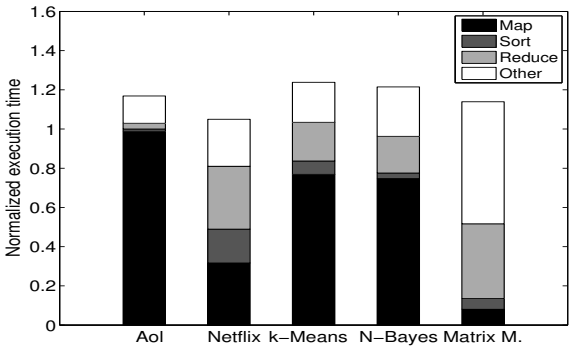

Figure 7: Normalized execution time of benchmarks when running on Airavat, compared to execution on Hadoop. Lower is better.

Benchmark	Sens. Data(KB)	#Mappers	Analysis Time	JVM overhead
AOL	780	19	5.9%	18.7%
Netflix	700	19	1.3%	14.2%
k-Means	120	30	7.1%	14.1%
Naive Bayes	98	23	8.7%	20.2%
Matrix Mult.	72	18	3.2%	20.6%

Table 2: Details of the analysis performed by the sensitivity meter, including size of data generated by the sensitivity meter, percentage of the total time spent processing sensitivity information and the JVM overhead for checking independence of mappers.

Figure 7 breaks down the execution time for each benchmark. The values are normalized to the execution time of the applications running on unmodified Hadoop. The graph depicts the percentage of the total time spent in the different phases like map, sort, and reduce. The category *Other* mainly represents the phase where the output data from the mappers is copied by the reducer. Note that the copy phase generally overlaps with the map phase. The benchmarks show that Airavat slows down the computation by less than 25%.

Table 2 presents results related to the sensitivity analysis for each benchmark. We measured the amount of data that had to be passed by the sensitivity meter to the reducer. In all the benchmarks this data is less than 0.01% of the input to the computation, indicating that the approach is scalable. In cases where each input record belongs to a group of its own (as in k-Means), the optimization of Section 6.3 brings down the generated data to a small constant value. Across all the benchmarks the sensitivity analysis took less than 9% of the total computation time. The JVM instrumentation adds upto 21% overhead in the map phase.

[2] www.jikesrvm.org

[3] www.lucene.apache.org/mahout/

Benchmark	Privacy Grouping	Reducer Primitive	#MapReduce Operations	Accuracy Parameter
AOL queries	Users	`MEAN`	Single	Avg. user count
Netflix recommendation system	Individual Rating	`COUNT`, `SUM`	Multiple	RMSE
k-Means	Individual points	`COUNT`, `SUM`	Multiple, till convergence	Intra-cluster variance
Naive Bayes	Individual articles	`SUM`	Multiple	Misclassification rate
Matrix multiply	Individual rows	`SUM`	Single	Frobenius Norm

Table 1: Details of the benchmarks, including the grouping of data, type of reducer used, number of MapReduce phases, and the parameter used to measure accuracy.

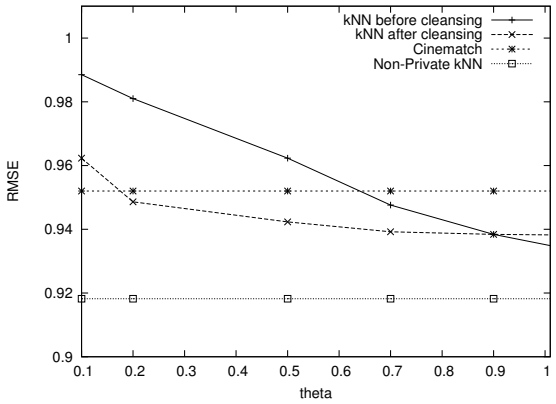

Figure 9: Effect of privacy bound on RMSE in Netflix recommender system. Lower is better.

7.2 Queries on AOL dataset

The AOL search logs released in 2006 contains about 20 million web queries from 650K AOL users. While the user identifiers were replaced with numbers, the web queries often provide unique identifiers, turning this release into a privacy fiasco. We perform two types of computation on this dataset. The first takes a list of keywords and outputs the total occurrences of the keywords in the search logs. The second computation returns the number of unique users who searched for any of the given 5 keywords. In both cases, the aim of Airavat is to guarantee privacy to the users who made the searches. The sensitivity value of the first computation is the maximum number of times a keyword may appear in any one user's search. In the second case, the sensitivity is 1 because the presence or absence of a user increases or decreases the number of the unique user count by at most one.

Figure 8 measures the accuracy of output as we vary the privacy factor ϵ. The curve shows that accuracy decreases when ϵ decreases because we added more noise. Also, popular keywords (keyword occurrence *high*) are more robust to noise than those that are uncommon.

7.3 Privacy preserving recommendation system with Netflix dataset

The Netflix data set consists of 100M movie ratings by 480K users to 17,770 movies. Users rate a movie on a scale of 1 to 5. Collaborative Filtering which works by aggregating the past behaviour of users, is an attractive solution for generating recommendations. Intuitively, the goal of recommendor system is to predict the missing ratings to movies by users based on ratings by other users to that movie or based on ratings to other movies by the same user.

In this experiment we implement a differentially private recommendor system based on Netflix data set, we adopt the differential privacy construction from [36]. The construction gurantees a relaxed form of differential privacy called (ϵ, δ)-differential privacy and masks the presence or absence of a single rating, there by providing per-rating privacy. The recommendor system is divided into two phases: in the first phase, privacy preserving computations are done on the private data to generate a covariance matrix that shows the correlation between every pair of movies; in the second phase, the non-private covariance matrix, Cov and a weights matrix, Wgt are released to a prediction algorithm like SVD or kNN.

In Airavat, all the data files used in first phase are tagged with DIFC labels to prevent leakage of private data. The file containing the covariance matrix and the weights matrix are unlabelled and can be released without leaking any private information. From this point, to predict ratings by user u, the prediction algorithm will use only Cov, Wgt and ratings by u and does not need to look up ratings by other users.

We use k-Nearest Neighbor (kNN) method of Bell and Koren [6], with Wgt matrix as the similarity matrix, we use the same parameters used in [36]: β_m =15, β_p =20, $B = 1$, and $k = 20$ with varying Gaussian noise parameter, θ. Our results are summarized in Figure 9. The results obtained are in line with the results obtained in [36]

7.4 Clustering Algorithm: k-Means

The k-Means algorithm clusters input vectors into k partitions. The partitioning aims to minimize the sum of the intra cluster variance. We use Lloyd's iterative heuristic to compute k-Means. The input consists of a set of vectors and an initial set of cluster centers. The algorithm proceeds in two steps [8]. In the first step, the cardinality of each cluster is calculated. In the second step, all the points in the new cluster are added up and then divided by the cardinality derived in the previous step. This gives the new cluster centers. Figure 10 shows the map and reduce functions to estimate the cardinality of the clusters.

Figure 8 plots the accuracy of the k-Means algorithm as we change the privacy factor. We assume that each point

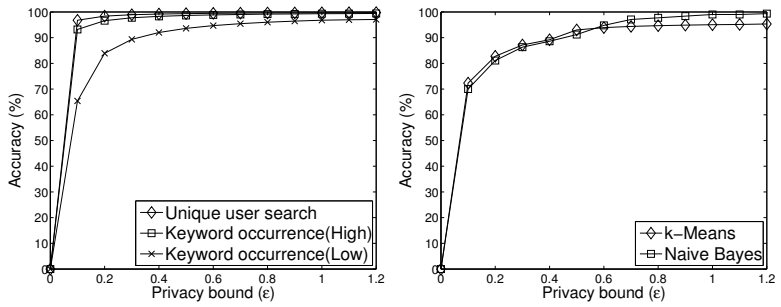

Figure 8: Effect of privacy bound on accuracy in (a) AOL queries, and (b) k-Means and Naive Bayes. Higher is better.

```
map(String key, String value){
 ...
 Vector point=decode(value);
 String id=Cluster.findClosetCenter(point);
 output(id,1);
}
reduce(String id, List cardinality){
 ..
for p in cardinality:
    size+=p;
 output(id, size+Airavat.getNoise(ϵ,S));
}
```

Figure 10: Pseudo code of mapper and reducer to approximate the size of clusters

belongs to a different user whose privacy has to be guaranteed. The sensitivity of the query that computes the cluster size is one. However, the sensitivity for calculating the actual cluster centers is the maximum value of any coordinate over all points. We measure the accuracy of the algorithm by computing the intra-cluster variance.

7.5 Classification Algorithm: Naive Bayes

Naive Bayes is a simple probabilistic classifier that applies the Bayes Theorem with assumptions of strong independence. During the training phase, the algorithm is given a set of feature vectors and the class labels to which they belong. The algorithm creates a model, which is then used in the classification phase to classify a new unseen vector.

Figure 8 shows the accuracy vs. privacy bound graph. We used the 20newsgroup dataset [4], which consists of different articles represented by words that appear in them. We train the classifier on one partition of the dataset and test it on another. The privacy bound affects the noise that is added to the model in the training phase. We measure the accuracy of the classifier by looking at the number of articles that were miss-classified.

7.6 Matrix Multiplication

A number of data mining algorithms (like SVD) use matrix multiplication of the form $A^T A$ as a basic step. This can be written as a summation, $\sum_i x_i^T x_i$, that fits naturally in the MapReduce framework [10]. We wrote a microbenchmark that computes $A^T A$. Each row belongs to a different user. If each element of the matrix is bounded by B, then the sensitivity of the computation is bounded by B^2. We use the Frobenius norm (Euclidean distance for matrices) as the accuracy metric. Our experiments show that the eigenvectors calculated on the noisy matrix are fairly robust to the noise added.

[4] http://people.csail.mit.edu/jrennie/20Newsgroups/

7.7 Summary

This section empirically makes the case that Airavat supports a variety of algorithms that can be written in a privacy preserving manner with acceptable accuracy loss. Many MapReduce programs (including those developed as community-wide efforts) can henceforth be run securely without the need for auditing them or their results. We restrict the reducers to the implementations that sum up the results. However, Airavat is also flexible because it can use traditional DIFC mechanisms to secure arbitrary mapper or reducer while inflicting modest overheads. This flexibility will benefit developers of more sophisticated algorithms that cannot be expressed using our stock reducers.

8 Related work

This section surveys relevant related work in decentralized information flow control and differential privacy.

Decentralized information flow control (DIFC) Information flow control mechanisms were developed to provide end-to-end security in military systems [14, 24]. Myers et. al. proposed the decentralized model for information flow control enabling users to label their data and also declassify it [38]. Recent research has focused on building secure systems using language level constructs [37, 39, 44], the operating system [26, 46, 49] or both [42]. Airavat adopts the labeling scheme used in Flume [26] to denote the sensitivity of data and limit its access by principals. We note that these systems provide security for applications that run on a single machine.

Secure program partitioning [48] assumes that hosts in a distributed setting may have different trust levels. It partitions a program, mapping the code and data to different hosts while upholding the security policy specified with labels. Swift applies similar techniques to partition web applications [9]. In contrast, Airavat does not partition

programs and assumes that all hosts pose a homogeneous threat to user data.

DStar is a framework that extends DIFC to an arbitrary network of machines [50] using local OS DIFC support and cryptography over the network. Airavat uses several DStar mechanisms including a trusted exporter process that regulates network access and certificates to bind IP addresses to host public keys (see Section 5.1.2), but Airavat makes the simplifying assumption that all hosts reside in the same administrative domain.

Differential privacy McSherry proposed a system of differential privacy via Privacy Integrated Queries (PINQ) [34]. PINQ provides a set of primitive data operations in the LINQ framework, each of which is implemented to provide differential privacy. It supports computations that are compositions of their primitives. While our definitions of privacy are effectively the same, our goal is substantially different and complimentary to PINQ. We focus not on the operations that the data owner performs on his own data prior to its public release, but on preserving privacy in large-scale, distributed MapReduce computations that operate on the data from multiple owners. We aim to use differential privacy as the basis for declassification in this setting.

Alternate definitions of privacy. Cryptographically secure multi-party computation(as in [31]) ensures that a distributed protocol leaks no more information about the individual inputs than is revealed by the result of the computation. This concept of privacy focuses on keeping the intermediate steps of the computation secret and is not appropriate in our setting, where the goal is to ensure that the result itself does not leak too much information about the inputs.

While differential privacy mechanisms tend to employ output perturbation (*i.e.*, adding random noise to the result of a computation), several approaches to privacy-preserving data mining focus on input perturbation (*i.e.*, adding random noise to individual data entries). Unfortunately, the guarantees provided in this case are usually average-case and do not imply anything about the privacy of individual inputs. For example, the algorithm given in the seminal paper by Agrawal and Srikant [3] fails to hide individual data entries, as shown by Agrawal and Aggarwal [2]. In turn, Evfimievski *et al.* show that the definitions of [2] are too weak to provide individual privacy [21].

The concept of k-anonymity focuses on non-interactive releases of relational data and requires that every record in the released dataset be syntactically indistinguishable from at least $k-1$ other records on the so-called quasi-identifying attributes (like ZIP code and date of birth) [11, 45]. This is achieved by syntactic generalization and suppression of these attributes (*e.g.*, [29]). Unfortunately, k-anonymity does not provide meaningful privacy guarantees. It fundamentally assumes that the adversary's knowledge is limited to the quasi-identifying attributes and thus fails to provide any protection against adversaries who have additional information [32, 33]. It does not hide whether a particular individual is in the database [41], nor the sensitive attributes associated with any individual [30, 32]. Multiple releases of the same dataset or even knowledge of the k-anonymization algorithm may completely break the protection [22, 51]. Modifications of k-anonymity, such as l-diversity [32] and m-invariance [47], suffer from many of the same flaws. Therefore, k-anonymity and its variants do not provide an appropriate privacy notion for cloud computing.

9 Conclusion

Airavat is the first system that integrates the DIFC model with concepts of differential privacy, allowing declassification without the need to audit code. We show how these techniques can be used to secure distributed computations that are expressed in the MapReduce model.

References

[1] *Hadoop*. `http://hadoop.apache.org/core/`.

[2] D. Agrawal and C. Aggarwal. On the design and quantification of privacy-preserving data mining algorithms. In *PODS*, 2001.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, 2000.

[4] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*, 2007.

[5] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Number MTR-2547, Vol. 1, 1973. MITRE.

[6] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 43–52, Washington, DC, USA, 2007. IEEE Computer Society.

[7] K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, USAF, 1977.

[8] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the SuLQ framework. In *In PODS*, 2005.

[9] Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng. Secure web application via automatic partitioning. In *SOSP*, 2007.

[10] Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288, 2006.

[11] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-anonymity. *Secure Data Management in Decentralized Systems*, 2007.

[12] T. Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift*, 15, 1977.

[13] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1), 2008.

[14] Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria (orange book)*, 1985.

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

[16] C. Dwork. Differential privacy. In *ICALP*, 2006.

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

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

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

[20] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[21] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy-preserving data mining. In *PODS*, 2003.

[22] S. Ganta, S. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *KDD*, 2008.
[23] S. Hansell. AOL removes search data on vast group of web users. New York Times, Aug 8 2006.
[24] Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn. A retrospective on the vax vmm security kernel. *IEEE Trans. Softw. Eng.*, 17(11), 1991.
[25] Michael Kearns. Efficient noise-tolerant learning from statistical queries. *J. ACM*, 45(6):983–1006, 1998.
[26] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard OS abstractions. In *SOSP*, 2007.
[27] Maxwell Krohn, A. Yip, M. Brodsky, Robert Morris, and Michael Walfish. A World Wide Web Without Walls. In *Hotnets*, 2007.
[28] B. W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, 1973.
[29] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *SIGMOD*, 2005.
[30] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and ℓ-diversity. In *ICDE*, 2007.
[31] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.
[32] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.
[33] D. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Halpern. Worst-case background knowledge for privacy-preserving data publishing. In *ICDE*, 2007.
[34] F. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. Manuscript, 2009.
[35] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, 2007.
[36] Frank McSherry and Ilya Mironov. Differentially private recommendar systems: Building privacy into the netflix prize contenders. In *Under submission*, 2009.
[37] A. C. Myers. JFlow: practical mostly-static information flow control. In *POPL*, 1999.
[38] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *SOSP*, pages 129–142, October 1997.
[39] A. C. Myers, N. Nystrom, L. Zheng, and S. Zdancewic. Jif: Java information flow. Software release. http://www.cs.cornell.edu/jif, July 2001.
[40] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. *IEEE Symposium on Security and Privacy*, 2008.
[41] M. Nergiz, M. Atzori, and C. Clifton. Hiding the presence of individuals from shared database. In *SIGMOD*, 2007.
[42] Indrajit Roy, Donald E. Porter, Michael D. Bond, Kathryn S. Mckinley, and Emmett Witchel. Laminar: Practical fine-grained decentralized information flow control. In *PLDI*, 2009.
[43] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Comm.*, 2003.
[44] V. Simonet and I. Rocquencourt. Flow Caml in a nutshell. In *Proceedings of the first APPSEM-II workshop*, pages 152–165, 2003.
[45] L. Sweeney. k-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.
[46] S. Vandebogart, P. Efstathopoulos, E. Kohler, M. Krohn, C. Frey, D. Ziegler, F. Kaashoek, R. Morris, and D. Mazières. Labels and event processes in the asbestos operating system. *TOCS*, 2007.
[47] X. Xiao and T. Tao. m-invariance: Towards privacy preserving republication of dynamic datasets. In *SIGMOD*, 2007.
[48] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *In TOCS*, 2002.
[49] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in HiStar. In *OSDI*, 2006.
[50] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières. Securing distributed systems with information flow control. In *NSDI*, 2008.
[51] L. Zhang, S. Jajodia, and A. Brodsky. Information disclosure under realistic assumptions: Privacy versus optimality. In *CCS*, 2007.