Airavat: Security and Privacy for MapReduce

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

Abstract

We present Airavat, a MapReduce-based system which provides strong security and privacy guarantees for distributed computations on sensitive data. Airavat is a novel integration of mandatory access control and differential privacy. Data providers control the security policy for their sensitive data, including a mathematical bound on potential privacy violations. Users without security expertise can perform computations on the data, but Airavat confines these computations, preventing information leakage beyond the data provider's policy.

Our prototype implementation demonstrates the flexibility of Airavat on several case studies. The prototype is efficient, with run times on Amazon's cloud computing infrastructure within 32% of a MapReduce system with no security.

1 Introduction

Cloud computing involves large-scale, distributed computations on data from multiple sources. The promise of cloud computing is based in part on its envisioned ubiquity: Internet users will contribute their individual data and obtain useful services from the cloud. For example, targeted advertisements can be created by mining a user's clickstream, while health-care applications of the future may use an individual's DNA sequence to tailor drugs and personalized medical treatments. Cloud computing will fulfill this vision only if it supports flexible computations while guaranteeing security and privacy for the input data. To balance the competing goals of a permissive programming model and the need to prevent information leaks, the untrusted code should be confined [30].

Contributors of data to cloud-based computations face several threats to their privacy. 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 which 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.*, aggregate health-care statistics computed as part of the study, may leak sensitive information about her personal medical record.

Traditional approaches to data privacy are based on syntactic anonymization, *i.e.*, removal of "personally identifiable information" such as names, addresses, and Social Security numbers. Unfortunately, anonymization does not provide meaningful privacy guarantees. High-visibility privacy fiascoes recently resulted from public releases of anonymized individual data, including AOL search logs [22] and the movie-rating records of Netflix subscribers [41]. The datasets in question were released to support legitimate data-mining and collaborative-filtering research, but naïve anonymization was easy to reverse in many cases. These events motivate a new approach to protecting data privacy.

One of the challenges of bringing security to cloud computing is that users and developers want to spend as little mental effort and system resources on security as possible. Completely novel APIs, even if secure, are unlikely to gain wide acceptance. Therefore, a key research question is how to design a practical system that (1) enables efficient distributed computations, (2) supports a familiar programming model, and (3) provides precise, rigorous privacy and security guarantees to data owners, even when the code performing the computation is untrusted. In this paper, we aim to answer this question.

Mandatory access control (MAC) is a useful building block for securing distributed computations. MAC-based operating systems, both traditional [26, 33, 37] and recent variants based on decentralized information flow control [45, 49, 51], enforce a single access control policy for the entire system. This policy, which cannot be overridden by users, prevents information leakage via storage channels such as files, sockets, and program names.

Access control alone does not achieve end-to-end privacy in cloud computing environments, where the input data may originate from multiple sources. The output of the computation may leak sensitive information about the inputs. Since the output generally depends on all input sources, mandatory access control requires that only someone who has access rights to all inputs should have access rights to the output; enforcing this requirement would render the output unusable for most purposes. To be useful, the output of an aggregate computation must be "declassified," but only when it is safe to do so, *i.e.*, when it does not reveal too much information about any single input. Existing access control mechanisms simply delegate this declassification decision to the implementor. In the case of untrusted code, there is no guarantee that the output of the computation does not reveal sensitive

information about the inputs.

In this paper, we present Airavat,[1] a system for distributed computations which provides end-to-end confidentiality, integrity, and privacy guarantees using a combination of mandatory access control and differential privacy. Airavat is based on the popular MapReduce framework, thus its interface and programming model are already familiar to developers. *Differential privacy* is a new methodology for ensuring that the output of aggregate computations does not violate the privacy of individual inputs [11]. It provides a mathematically rigorous basis for declassifying data in a mandatory access control system. Differential privacy mechanisms add some random noise to the output of a computation, usually with only a minor impact on the computation's accuracy.

Our contributions. We describe the design and implementation of Airavat. Airavat enables the execution of trusted and untrusted MapReduce computations on sensitive data, while assuring comprehensive enforcement of data providers' privacy policies. To prevent information leaks through system resources, Airavat runs on SELinux [37] and adds SELinux-like mandatory access control to the MapReduce distributed file system. To prevent leaks through the output of the computation, Airavat enforces differential privacy using modifications to the Java Virtual Machine and the MapReduce framework. Access control and differential privacy are synergistic: if a MapReduce computation is differentially private, the security level of its result can be safely reduced.

To show the practicality of Airavat, we carry out several substantial case studies. These focus on privacy-preserving data-mining and data-analysis algorithms, such as clustering and classification. The Airavat prototype for these experiments is based on the Hadoop framework [2], executing in Amazon's EC2 compute cloud environment. In our experiments, Airavat produced accurate, yet privacy-preserving answers with runtimes within 32% of conventional MapReduce.

Airavat provides a practical basis for secure, privacy-preserving, large-scale, distributed computations. Potential applications include a wide variety of cloud-based computing services with provable privacy guarantees, including genomic analysis, outsourced data mining, and clickstream-based advertising.

2 System overview

Airavat enables the execution of potentially untrusted data-mining and data-analysis code on sensitive data. Its objective is to accurately compute general or aggregate features of the input dataset without leaking information about specific data items.

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

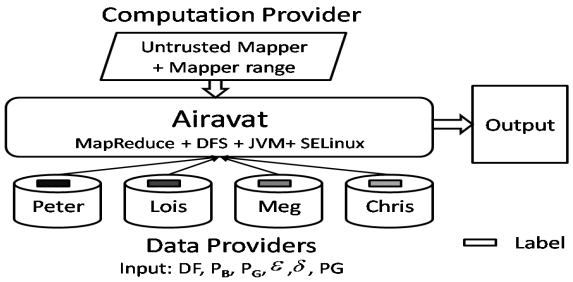

Figure 1: High-level architecture of Airavat.

As a motivating scenario, consider an online retailer, BigShop, which holds a large database of customer transactions. For now, assume that all records in the database have the form ⟨customer, order, date⟩, with only one record per customer. A machine learning expert, Bob, pays BigShop to mine the data for certain transaction patterns. BigShop loads the data into the Hadoop framework and Bob writes the MapReduce code to analyze it.

Such computations are commonly used for targeted advertising and customer relationship management, but we will keep the example simple for clarity and assume that Bob wants to find the total number of orders placed on a particular date D. He writes a mapper that looks at each record and emits the key/value pair $\langle K, \text{order}\rangle$ if the date on the record is D. Here, K is a string constant. The reducer simply sums up the values associated with each key K and outputs the result.

The main risk for BigShop in this scenario is the fact that Bob's code is untrusted and can therefore be unintentionally buggy or even actively malicious. Because Bob's mapper has direct access to BigShop's proprietary transaction records, it can store parts of these data in a file which will be later accessed by Bob, or it can send them over the network. Such a leak would put BigShop at a commercial disadvantage and may also present a serious reputational risk if individual BigShop transactions were made public without the consent of the customer.

The output of the computation may also leak information. For example, Bob's mapper may signal the presence (or absence) of a certain customer in the input dataset by manipulating the order count for a particular day: if the record of this customer is in the dataset, the mapper outputs an order count of 1 million; otherwise, it outputs zero. Clearly, the result of the computation in this case violates the privacy of the customer in question.

2.1 Architecture of Airavat

The three main entities in our model are (1) the data provider (BigShop, in our motivating example), (2) the computation provider (Bob, sometimes referred to as a user making a query), and (3) the computation frame-

work (Airavat). We aim to prevent malicious computation providers from violating the privacy policy of the data provider(s) by leaking information about individual data items.

Computation providers write their code in the familiar MapReduce paradigm, while data providers specify the parameters of their privacy policies. We relieve data providers of the need to audit computation providers' code for privacy compliance.

Figure 1 gives an overview of the Airavat architecture. Airavat consists of modifications to the MapReduce framework, the distributed file system, and the Java virtual machine with SELinux as the underlying operating system. Airavat uses SELinux's mandatory access control to ensure that untrusted code does not leak information via system resources, including network connections, pipes, or other storage channels such as names of running processes. To prevent information leakage through the output of the computation, Airavat relies on a differential privacy mechanism [11].

Data providers put access control labels on their data and upload them to Airavat. Airavat ensures that the result of a computation is labeled with the union of all input labels. A data provider, D, can set the *declassify flag* (**DF** in Table 1) to true if he wants Airavat to remove his label from the output when it is safe to do so. If the flag is set, Airavat removes D's label if and only if the computation is differentially private. Data providers must also create a privacy policy by setting the value of several privacy parameters (explained in Section 4).

The computation provider must write his code in the Airavat programming model, which is close to standard MapReduce. The *sensitivity* of the function being computed determines the amount of perturbation that will be applied to the output to ensure differential privacy (§ 4). Therefore, in Airavat the computation provider must supply an upper bound on the sensitivity of his computation by specifying the range of possible values that his mapper code may output. Airavat then ensures that the code never outputs values outside the declared range and perturbs those within the range so as to ensure privacy (§ 5.1). If malicious or incorrect code tries to output a value outside its declared range, the enforcement mechanism guarantees that no privacy breach will occur, but the results of the computation may no longer be accurate.

Apart from specifying the parameters mentioned above, neither the data provider, nor the computation provider needs to understand the intricacies of differential privacy and its enforcement.

2.2 Trusted computing base of Airavat

Airavat trusts the cloud provider and the cloud-computing infrastructure. It assumes that SELinux correctly implements MAC and relies on the MAC features added to the MapReduce distributed file system, as well as on the (modified) Java virtual machine to enforce certain properties of the untrusted mapper code (see Section 5.3). Airavat includes trusted implementations of several reducer functions.

Participant	Input
Data provider	Labeled data (**DB**) Declassify flag (**DF**) Privacy parameters (ϵ, δ) Privacy budget ($\mathbf{P_B}$) Code to determine privacy groups* (**PG**)
Computation provider (user making a query)	Mapper range ($\mathbf{M_{min}}$, $\mathbf{M_{max}}$) Independent mapper code (**Map**) Number of outputs (**N**) Code to determine partitions* (**PC**) Map of partition to output* (**PM**) Max keys output for any privacy group* (**n**)
Airavat	Trusted reducer code (**Red**) Modified MapReduce (**MMR**) Modified distributed file system (**MDFS**) SELinux policy (**SE**)

Table 1: Parameters and components provided by different participants. Optional parameters are starred.

We assume that the adversary is a malicious computation provider who has full control over the mapper code supplied to Airavat. The adversary may attempt to access the input, intermediate, and output files created by this code, or to reconstruct the values of individual inputs from the result of the computation.

2.3 Limitations of Airavat

Airavat cannot confine every computation performed by untrusted code. For example, a MapReduce computation may output key/value pairs. Keys are text strings that provide a storage channel for malicious mappers. In general, Airavat cannot guarantee privacy for computations which output keys produced by untrusted mappers. In many cases, privacy *can* be achieved by requiring the computation provider to declare the key in advance and then using Airavat to compute the corresponding value in a differentially private way.

MapReduce computations that necessarily output keys require trusted mappers. For example, printing the top K items sold in a store involves printing item names. Because a malicious mapper can use a name to encode information about individual inputs, this computation requires trusted mappers. By contrast, the number of iPods sold can be calculated using an untrusted mapper because the key ("iPod" in this case) is known prior to the answer being released. (See Section 5 for details.)

3 MapReduce and MAC

Table 1 lists the components of the system contributed by the data provider(s), computation provider(s), and Airavat. The following discussion explains entries in the table

in the context of MapReduce computations, mandatory access control, or differential privacy. We place a bold label in the text to indicate that the discussion is about a particular row in the table.

3.1 MapReduce

MapReduce [9] is a framework for performing data-intensive computations in parallel on commodity computers. A MapReduce computation reads input files from a distributed file system which splits the file into multiple chunks. Each chunk is assigned to a *mapper* which reads the data, performs some computation, and outputs a list of key/value pairs. In the next phase, *reducers* combine the values belonging to each distinct key according to some function and write the 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, we defer them to future work. Airavat secures the execution of untrusted mappers (**Map**) using a MAC OS (**SE**), as well as modifications to the MapReduce framework (**MMR**) and distributed file system (**MDFS**).

3.2 Mandatory access control

Mandatory access control (MAC) assigns security attributes to system resources and uses these attributes to constrain the interaction of subjects (*e.g.*, processes) with objects (*e.g.*, files). In contrast to discretionary access control (*e.g.*, UNIX permissions), MAC systems (1) check permissions on every operation and transitively enforce access restrictions (*e.g.*, processes that access secret data cannot write non-secret files) and (2) enforce access rules specified by the system administrator at all times, without user override. MAC systems include mainstream implementations such as SELinux [37] and AppArmor [1] which appear in Linux distributions, as well as research prototypes [45, 49, 51] which implement a MAC security model called decentralized information flow control (DIFC). Our current implementation uses SELinux because it is a mature system that provides sufficient functionality to enforce Airavat's security policies.

SELinux divides subjects and objects into groups called *domains* or *types*. The domain is part of the security attribute of system resources. A domain can be thought of as a sandbox which constrains the permissions of the process. For example, the system administrator may specify that a given domain can only access files belonging to certain domains. In SELinux, one can specify rules that govern transition from one domain to another. Generally, a transition occurs by executing a program declared as the entry point for a domain.

In SELinux, users are assigned roles. A role governs the permissions granted to the user by determining which domains he can access. For example, the system administrator role (`sysadm_r`) has permissions to access the `ifconfig_t` domain and can perform operations on the network interface. In SELinux, access decisions are declared in a policy file which is customized and configured by the system administrator. The Airavat-specific SELinux policy to enforce mandatory access control and declassification (**SE, DF**) is described in Section 6.

4 Differential privacy

The objective of Airavat is to enable large-scale computation on data items that originate from different sources and belong to different owners. The fundamental question of what it means for a computation to preserve the privacy of its inputs has been the subject of much research (see Section 9).

Airavat uses the recently developed framework of *differential privacy* [11, 12, 13, 14] to answer this question. Intuitively, 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 input dataset or not. Formally, a computation $\mathcal{F}$ satisfies (ϵ, δ)-differential privacy [15] (where ϵ and δ 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 $S \subseteq Range(\mathcal{F})$,

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

Another intuitive way to 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 $\mathcal{F}$ must be randomized to achieve privacy (probability in the above definition is taken over the randomness of $\mathcal{F}$). 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 δ 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 mul-

tiplicative 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 δ in the definition is to allow a small absolute difference in probabilities. In many of the computations considered in this paper, this situation does not arise and δ can be safely set to 0.

In Section 9, we discuss why differential privacy is the "right" concept of privacy for cloud computing. The most important feature of differential privacy is that it does not make any assumptions about the adversary. When satisfied, it holds regardless of the auxiliary or prior knowledge that the adversary may possess. Furthermore, differential privacy is composable: a composition of two differentially private computations is also differentially private (of course, ϵ and δ may increase).

There are many mechanisms for achieving differential privacy [5, 14, 17, 40]. In this paper, we will use the mechanism that adds Laplacian noise to the output of a computation $f : D \to R^k$:

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

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

Privacy groups. To provide privacy guarantees which are meaningful to users, it is sometimes important to consider input datasets that differ not just on a single record, but on a group of records (**PG**). For example, when searching for a string within a set of documents, each input might be a line from a document, but the privacy guarantee should apply to whole documents. Differential privacy extends to privacy groups via composability: the effect of n input items on the output is at most n times the effect of a single item.

4.1 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.

Formally, the sensitivity of a function $f : D \to R^k$ is

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

for any D, D' that are identical except for a single element, which is present in D, but not in D'. In this paper, we will be primarily interested in 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. Therefore, we require the computation provider to furnish the range of possible outputs for his mappers and use this range to derive *estimated sensitivity*. Estimated sensitivity is then used to add sufficient random noise to the output and guarantee privacy regardless of what the untrusted code does. If the code is malicious and attempts to output values outside its declared range, the enforcement mechanism will chose a value within the range. The computation still guarantees privacy, but the results may no longer be accurate (§ 5.1).

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.

4.2 Privacy budget

Data providers may want 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 [10] shows that the entire dataset can be decoded with a linear (in the size of the dataset) number of queries. This is a serious, but inevitable, limitation. Existing privacy mechanisms which are not based on differential privacy either severely limit the utility of the data, or are only secure against very restricted adversaries (see [13] and Section 9).

The composability of differential privacy and the need to restrict the number of queries naturally give rise to the concept of a "*privacy budget*" ($\mathbf{P_B}$) [17, 38]. Each differentially private computation with a privacy parameter of ϵ

results in subtracting ϵ from this budget. Once the privacy budget is exhausted, results can no longer be automatically declassified. The need to pre-specify a limit on how much computation can be done over a given dataset constrains some usage scenarios. We emphasize that there are no definitions of privacy that are robust, composable, and achievable in practice without such a limit.

After the privacy budget has been exhausted, Airavat still provides useful functionality. While the output can no longer be automatically declassified without risking a privacy violation, Airavat still enforces access control restrictions on the untrusted code and associates proper access control labels with the output. In this case, outputs are no longer public and privacy protection is based solely on mandatory access control.

5 Enforcing differential privacy

Airavat supports both trusted and untrusted mappers. Because reducers are responsible for enforcing privacy, they must be trusted. The computation provider selects a reducer from a small set included in the system.

The outputs of mappers and reducers are lists of key/-value pairs. An untrusted, potentially malicious mapper may try to leak information about an individual input by encoding it in (1) the values it outputs, (2) the keys it outputs, (3) the order in which it outputs key/value pairs, or (4) relationships between output values of different keys.

MapReduce keys are arbitrary strings. Airavat cannot determine whether a key encodes sensitive information. The mere presence of a particular key in the output may signal information about an individual input. Therefore, Airavat never outputs any keys produced by untrusted mappers. Instead, the computation provider submits a key or list of keys as part of the query and Airavat returns (noisy) values associated with these keys. As explained below, Airavat always returns a value for every key in the query, even if none of the mappers produced this key. This prevents untrusted mappers from signaling information by adding or removing keys from their output.

For example, Airavat can be used to compute the noisy answer to the query "*What is the total number of iPods and pens sold today?*" (see the example in Section 5.4) because the two keys `iPod` and `pen` are declared as part of the computation. The query "*List all items and their sales*" is not allowed in Airavat, unless the mapper trusted. The reason is that a malicious mapper can leak information by encoding it in item names.

Trusted Airavat reducers always sort keys prior to outputting them. Therefore, a malicious mapper cannot use key order as a channel to leak information about a particular input record.

A malicious mapper may attempt to encode information by emitting a certain combination of values associated with different keys. As explained below, trusted reducers use the declared output range of mappers to add sufficient noise to ensure differential privacy for the outputs. In particular, a combination C of output values across multiple keys does not leak information about any given input record r because the probability of Airavat producing C is approximately the same with or without r in the input dataset.

In the rest of this section, we explain how Airavat enforces differential privacy for computations involving untrusted mappers. We use BigShop from Section 2 as our running example. We also briefly describe a broader class of differentially private computations which can be implemented using trusted mappers.

5.1 Range declarations and estimated sensitivity

Airavat reducers enforce differential privacy by adding exponentially distributed noise to the output of the computation. The sensitivity of the computation determines the amount of noise: the noise must be sufficient to mask the maximum influence that any single input record can have on the output (§ 4.1).

In the case of untrusted mappers, the function(s) they compute and their sensitivity are unknown. To help Airavat estimate sensitivity, we require the computation provider to declare the range of output values (M_{min}, M_{max}) that his mapper can produce. Airavat combines this range with the sensitivity of the function implemented by the trusted reducer (**Red**) into *estimated sensitivity*. For example, estimated sensitivity of the `SUM` reducer is $max(|M_{max}|, |M_{min}|)$, because any single input can change the output by at most this amount.

The declared mapper range can be greater or smaller than the true global sensitivity of the function computed by the mapper. While global sensitivity measures the output difference between any two inputs that differ in at most one element (§ 4.1), the mapper range captures the difference between *any* two inputs. That said, the computation provider may assume that all inputs for the current computation lie in a certain subset of the function's domain, so the declared range may be lower than the global sensitivity. In our clustering case study (§ 8.4), such an assumption allows us to obtain accurate results even though global sensitivity of clustering is very high (on "bad" input datasets, a single point can significantly change the output of the clustering algorithms).

The random noise added by Airavat to the output of MapReduce computations is a function of the data provider's privacy parameter ϵ and the estimated sensitivity. For example, Airavat's `SUM` reducer adds noise from the Laplace distribution, Lap($\frac{b}{\epsilon}$), where $b = max(|M_{max}|, |M_{min}|)$.

Example. In the BigShop example, Bob writes his own mapper and uses the `SUM` reducer to compute the total number of orders placed on date D. Assuming that a cus-

tomer can order at most 25 items on any single day, Bob declares his mapper range as $(0, 25)$. The estimated sensitivity is 25 because the presence or absence of a record can affect the order total by at most 25.

Privacy groups. In the BigShop example, we may want to provide privacy guarantees at the level of customers rather than records (a single customer may have multiple records). Airavat supports privacy groups (§ 4), which are collections of records that are jointly present or absent in the dataset. The data provider supplies a program (**PG**) that takes a record as input and emits the corresponding group identifier, `gid`. Airavat attaches these identifiers to key/value pairs to track the dispersal of information from each input privacy group through intermediate keys to the output. The mapper range declared by the computation provider is interpreted at the group level. For example, suppose that each BigShop record represents a purchase, a customer can make at most 10 purchases a day, and each purchase contains at most 25 orders. If all orders of a single customer are viewed as a privacy group, then the mapper range is $(0, 250)$.

5.2 Range enforcement

To prevent malicious mappers from leaking information about inputs through their output values, Airavat associates a *range enforcer* with each mapper. The range enforcer checks that the value in each key/value pair output by the mapper lies within its declared range. This check guarantees that the actual sensitivity of the computation performed by the mapper does not exceed the estimated sensitivity, which is based on the declared range. If a malicious mapper outputs a value outside the range, the enforcer replaces it with a value inside the range. In the latter case, differential privacy holds, but the computation may no longer produce accurate or meaningful results.

Range enforcement in Airavat prioritizes privacy over accuracy. If the computation provider declares the range incorrectly, the computation remains differentially private. However, the results are not meaningful and the provider gets no feedback about the problem, because any such feedback would be an information leak. The lack of feedback may seem unsatisfying, but other systems that tightly regulate information flow make similar tradeoffs. For example, MAC systems Flume and Asbestos make pipes (used for interprocess communication) unreliable and do not give the user any feedback about their failure because such feedback would leak information [29, 49].

Providing a mapper range is simple for some computations. For example, Netflix movie ratings (§8.3) are always between 1 and 5. When computing the word count of a set of documents, however, estimating the mapper range is more difficult. If each document is at most N words, and the document is a privacy group, then the $0 - N$ range will guarantee privacy of individual documents. Depending on the number of documents, such a large range may result in adding excessive noise. For some domains, it might not be possible to obtain a reasonable estimate of the mapper's range. Airavat gives accurate results only when the computation provider understands the sensitivity of his computation.

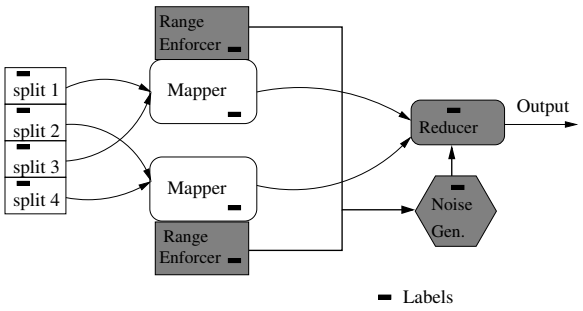

Figure 2: Simplified overview of range enforcers and noise generation. Trusted components are shaded.

In the BigShop example, the range enforcer ensures that in every ⟨K, V⟩ pair output by the mapper, $0 \leq V \leq 25$. Suppose a malicious mapper attempts to leak information by outputting $1,000$ when Alice's record is in the input dataset and 0 otherwise. Because $1,000$ is outside the declared range, the range enforcer will replace it with, say, 12.5. The difference between 0 and 12.5 is less than the estimated sensitivity. Therefore, enough noise will be added so that one cannot tell, by looking at the output, whether this output was obtained by adding noise to 12.5 or 0. The noisy output thus does not reveal whether Alice's record was present in the input dataset or not.

Distributed range enforcement. A single MapReduce operation may execute mappers on many different machines. These mappers may process input elements with the same key or privacy group. Airavat associates a range enforcer with each mapper and merges their states at the end of the map phase. After merging, Airavat ensures that the values corresponding to each key or privacy group are within the declared range (see Figure 2).

Example: "noisy sum." Figure 3 illustrates differential privacy enforcement with an untrusted mapper and the `SUM` reducer. This "noisy sum" primitive was shown by Blum *et al.* [6] to be sufficient for privacy-preserving computation of all algorithms in the statistical query model [27], including k-Means, Naive Bayes, principal component analysis, and linear classifiers such as perceptrons (for a slightly different definition of privacy).

Each input record is its own privacy group. The computation provider supplies the implementation of the actual mapper function **Map**, which converts every input record into a list of key/value pairs.

// Inputs and definitions

Data owner: **DB**, ϵ, $\delta = 0$, $\mathbf{P_B}$,
Computation provider: **Map**, $\mathbf{M_{min}}$, $\mathbf{M_{max}}$, **N**
Airavat: SUM (trusted reducer, **Red**)
b = max($|M_{max}|, |M_{min}|$)
μ = $(M_{max} - M_{min})/2$

// Map phase

```
if(P_B − ε × N < 0){
   print ''Privacy limit exceeded'';
   TERMINATE
}
P_B = P_B − ε × N
For(Record r in DB){
   (k_0, v_0), ..., (k_n, v_n) = Map(r)
   For(i: 1 to n){
      if(v_i < M_min or v_i > M_max) {
        v_i = μ }
   }
   emit⟨k_0, v_0⟩ ... ⟨k_n, v_n⟩
}
```

// Reduce phase

```
count = N
Reduce(Key k, List val){
   if(--count ≤ 0) { Skip }
   V = SUM(val)
   print V + Lap(b/ε)
}
for(i: count to 0) {
   print Lap(b/ε) }
```

Figure 3: Simplified pseudo-code demonstrating differential privacy enforcement.

5.3 Mapper independence

Airavat forces all invocations of a mapper in a given MapReduce computation to be *independent*. Only a single input record is allowed to affect the key/value pairs output by the mapper. The mapper may not store the key/value pair(s) produced from an input record and use them later, when computing the key/value pair for another record. Without this restriction, estimated sensitivity used in privacy enforcement may be lower than the actual sensitivity of the mapper, resulting in a potential privacy violation. Mappers can only create additional keys for the same input record, they cannot merge information contained in different input records. We ensure mapper independence by modifying the JVM (§ 7.3).

Each mapper is permitted by the Airavat JVM to initialize itself once by overriding the `configure` function, called when the mapper is instantiated. To ensure independence, during initialization the mapper may not read any files written in this MapReduce computation.

5.4 Managing multiple outputs

A MapReduce computation may output more than one key/value pair (*e.g.*, Figure 3). The computation provider must specify the number of output keys (**N**) beforehand; otherwise, the number of outputs can become a channel through which a malicious mapper can leak information about the inputs. If a computation produces more (fewer) than the declared number of outputs, then Airavat removes (creates) outputs to match the declared value.

Range restrictions are enforced separately for each (privacy group, key) pair. Therefore, random noise is independently added to all values associated with the final output keys. Recall that Airavat never outputs a key produced by an untrusted mapper. Instead, the computation provider must specify a key or list of keys as part of the query, and Airavat will return the noisy values associated with each key in the query. For such queries, **N** can be calculated automatically.

In general, each output represents a separate release of information about the same input. Therefore, Airavat must subtract more than one ϵ from the privacy budget (see Figure 3). If different outputs are based on disjoint parts of the input, then smaller deductions from the privacy budget are needed (see below).

Example. Consider the BigShop example, where each record includes the customer's name, a list of products, and the number of items bought for each product (*e.g.*, [Joe, iPod, 1, pen, 10]). The privacy group is the customer, and each customer may have multiple records. Bob wants to compute the total number of iPods and pens sold. Bob must specify that he expects two outputs. If he specifies the keys for these outputs as part of the query (*e.g.*, "iPod" and "pen"), then the keys will be printed. Otherwise, only the values will be printed. Airavat subtracts 2ϵ from the privacy budget for this query.

Bob's mapper, after reading a record, outputs the product name and the number of sold items (*e.g.*, $\langle$iPod, 1$\rangle$, $\langle$pen, 10$\rangle$—note that more than one key/value pair is output for each record). Bob also declares the mapper range for each key, *e.g.*, $(0, 5)$ for the number of iPods bought and $(0, 25)$ for the number of pens. Airavat range enforcers automatically group the values by customer name and enforce the declared range for each item count. The final reducer adds random noise to the total item counts.

Computing on disjoint partitions of the input. When different outputs depend on disjoint parts of the input, the MapReduce computation can be decomposed into independent, parallel computations on independent datasets, and smaller deductions from the privacy budget are sufficient to ensure differential privacy. To help Airavat track and enforce the partitioning of the input, the computation provider must (1) supply the code (**PC**) that assigns input records to disjoint partitions, and (2) specify which of the

declared outputs will be based on which partition (**PM**).

The **PC** code is executed as part of the initial mapper. For each key/value pair generated by a mapper, Airavat constructs records of the form $\langle key, value, gid, pid \rangle$, where `gid` is the privacy group identifier and `pid` is the partition identifier.

The computation provider declares which partition produces which of the **N** final outputs (**PM**). Airavat uses **PM** to calculate **p**, the maximum number of final outputs that depend on any single partition. If **PC** and **PM** are not provided, Airavat sets **p** to equal **N**. Airavat charges $\epsilon \times \min(\mathbf{N}, \mathbf{p})$ from the privacy budget. For example, a computation provider may partition the BigShop data into two cities `Austin` and `Seattle` which act as the partition identifiers. He then specifies that the MapReduce computation will have 8 outputs, the first five of which are calculated from the `Austin` partition and the next three from the `Seattle` partition. In this example, $N = 8$, $p = 5$, and to run the computation, Airavat will subtract 5ϵ from the privacy budget. In Figure 3, $\epsilon \times N$ is charged to the privacy budget because the N outputs depend on the entire input, *not* on disjoint partitions.

Airavat enforces the partitioning declared by the computation provider. Trusted Airavat reducers use partition identifiers to ensure that only key/value pairs that have the correct `pid` are combined to generate the output. Airavat uses **PM** for computations on disjoint partitions in the same way as it uses **N** for unpartitioned data. If the number of outputs for a partition is less (more) than what is specified by **PM**, Airavat adds (deletes) outputs.

5.5 Trusted reducers and reducer composition

Trusted reducers such as `SUM` and `COUNT` are executed directly on the output of the mappers. The computation provider can combine these reducers with any untrusted mapper, and Airavat will ensure differential privacy for the reducer's output. For example, to calculate the total number of products sold by BigShop, the mapper will be responsible for the parsing logic and manipulation of the data. `COUNT` is a special case of `SUM` where the output range is $\{0, 1\}$. `MEAN` is computed by calculating the `SUM` and dividing it by the `COUNT`.

Reducers can be composed sequentially. `THRESHOLD`, `K-COUNT`, and `K-SUM` reducers are most useful when applied to the output of another reducer. `THRESHOLD` prints the outputs whose value is more than C, where C is a parameter. `K-COUNT` counts the number of records, and `K-SUM` sums the values associated with each record. For example, to count the number of distinct words occurring in a document, one can first write a MapReduce computation to group the words and then apply `K-COUNT` to calculate the number of groups. The sensitivity of `K-COUNT` is equal to the maximum number of distinct keys that a mapper can output after processing any input record.

5.6 Enforcing δ

Privacy guarantees associated with the `THRESHOLD` reducer may have non-zero δ. Intuitively, δ bounds the probability that the values generated from a given record will exceed the threshold and appear in the final output. Assuming that the mapper outputs at most **n** keys after processing a single record and the threshold value is C,

$$\delta \leq \frac{\mathbf{n}}{2} \exp\left(\epsilon \cdot (1 - \frac{C}{\Delta f})\right)$$

The proof is omitted because of space constraints and appears in a technical report [46]. When the computation provider uses the `THRESHOLD` reducer, Airavat first calculates the value of δ. If it is less than the bound specified by the data provider, then computation can proceed; otherwise it is aborted.

5.7 Mapper composition

Multiple mappers $\{M_1, \ldots, M_j\}$ can be chained one after another, followed by a final reducer R_j. Each mapper after the initial mapper propagates the partition identifier (`pid`) and privacy group (`gid`) values from the input record to output key/value pairs. Airavat enforces the declared range for the output of the final mapper M_j. Noise is added only once by the final reducer R_j.

To reduce the charge to the privacy budget, the computation provider can specify the maximum number of keys **n** that any mapper can output after reading records from a single privacy group. If provided, Airavat will enforce that maximum. If **n** is not provided, Airavat sets **n** equal to **N**. If a mapper generates more than **n** key/value pairs, Airavat will only pass **n** randomly selected pairs to the next mapper.

When charging the total cost of a composed computation to the privacy budget, Airavat uses $\epsilon \times \min(\mathbf{N}, \mathbf{p}, \mathbf{n}^j)$ where j is the number of composed mappers, **p** is the maximum number of outputs from any partition (§5.4), and **N** is the total number of output keys. If the computation provider supplies the optional arguments, then $\mathbf{N} > \mathbf{p} > \mathbf{n}^j$ results in a more economical use of the privacy budget.

MapReduce composition not supported. Airavat supports composition of mappers and composition of reducers, but not general composition of MapReduce computations (*i.e.*, reducer followed by another mapper). For many reducers, the output of a MapReduce depends on the inputs in a complex way that Airavat cannot easily represent, making sensitivity calculations difficult.

In the future, we plan to investigate MapReduce composition for reducers that do not combine information associated with different keys (*e.g.*, those corresponding to a "select" statement).

5.8 Choosing privacy parameters

Providers of sensitive data must supply privacy parameters ϵ and δ, as well as the privacy budget P_B, in order for their data to be used in an Airavat computation. These parameters are part of the differential privacy model. They control the tradeoff between accuracy and privacy. It is not possible to give a generic recommendation for setting their values because they are highly dependent on the type of the data, the purpose of the computation, privacy threats that the data provider is concerned about, *etc.*

As ϵ increases, the amount of noise added to the output decreases. Therefore, the output becomes more accurate, but there is a higher chance that it reveals the presence of a record in the input dataset. In many cases, the accuracy required determines the minimum ϵ-privacy that can be guaranteed. For example, in Section 8.5 we classify documents in a privacy-preserving fashion. Our experiments show that to achieve 95% accuracy in classification, we need to set ϵ greater than 0.6.

Intuitively, δ bounds the probability of producing an output which can occur only as a result of a particular input (see Section 4). Clearly, such an output immediately reveals the presence of the input in question. In many computations—for example, statistical computations where each input datapoint is a single number—δ should be set to 0. In our AOL experiment (§8.2), which outputs the search queries that occur more than a threshold number of times, δ is set to a value close to the number of unique users. This value of δ bounds the probability that a single user's privacy is breached due to the release of his search query.

The privacy budget (P_B) is finite. If data providers specify a single privacy budget for all computation providers, then one provider can exhaust more than its fair share. Data providers could specify privacy budgets for each computation provider to ensure fairness. Managing privacy budgets is an administrative issue inherent to all differential privacy mechanisms and orthogonal to the design of Airavat.

5.9 Computing with trusted mappers

While basic differential privacy only applies to computations that produce numeric outputs, it can be generalized to discrete domains (*e.g.*, discrete categories or strings) using the exponential mechanism of McSherry and Talwar [40]. In general, this requires both mappers and reducers to be trusted, because keys are an essential part of the system's output. Our prototype includes an implementation of this mechanism for simple cases, but we omit the definition and discussion for brevity.

As a case study, one of the authors of this paper ported CloudBurst, a genome mapping algorithm written for MapReduce [47], to Airavat. The CloudBurst code contains two mappers and two reducers (3,500 lines total, including library routines). The mappers are not independent and the reducers perform non-trivial computation.

The entire system was ported in a week. If a reducer was non-trivial, it was replaced by an identity reducer and its functionality was executed as the second mapper stage. This transformation was largely syntactic. Some work was required to make the mappers independent, and about 50 lines of code had to be added for differential privacy enforcement.

6 Enforcing mandatory access control

This section describes how Airavat confines MapReduce computations, preventing information leaks via system resources by using mandatory access control mechanisms. Airavat uses SELinux to execute untrusted code in a sandbox-like environment and to ensure that local and HDFS files are safeguarded from malicious users. While decentralized information flow control (DIFC) [45, 49, 51] would provide far greater flexibility for access control policies within Airavat, only prototype DIFC operating systems exist. By contrast, SELinux is a broadly deployed, mature system.

6.1 SELinux policy

Airavat's SELinux policy creates two domains, one trusted and the other untrusted. The trusted components of Airavat, such as the MapReduce framework and DFS, execute inside the trusted domain. These processes can read and write trusted files and connect to the network. Untrusted components, such as the user-provided mapper, execute in the untrusted domain and have very limited permissions.

Table 2 shows the different domains and how they are used. The `airavatT_t` type is a trusted domain used by the MapReduce framework and the distributed file system. Airavat labels executables that launch the framework and file system with the `airavatT_exec_t` type so the process executes in the trusted domain. This trusted domain reads and writes only trusted files (labeled with `airavatT_rw_t`). No other domain is allowed to read or write these files. For example, the distributed file system stores blocks of data in the underlying file system and labels files containing those blocks with `airavatT_rw_t`.

In certain cases Airavat requires the trusted domain to create configuration files that can later be read by untrusted processes for initialization. Airavat uses the `airavatT_notsec_t` domain to label configuration files which do not contain any secrets but whose integrity is guaranteed. Since MapReduce requires network communication for transferring data, our policy allows network access by the trusted domain.

Only privileged users may enter the trusted domain. To implement this restriction, Airavat creates a trusted SELinux user called `airavat_user`.

Domain	Object labeled	Remark
airavatT_t	Process	Trusted domain. Can access airavatT_*_t and common domains like sockets, networking, etc.
airavatT_exec_t	Executable	Used to transition to the airavatT_t domain.
airavatT_rw_t	File	Used to protect trusted files.
airavatT_notsec_t	File	Used to protect configuration files that contain no secrets. Can be read by untrusted code for initialization.
airavatU_t	Process	Untrusted domain. Can access only airavatT_notsec_t and can read and write to pipes of the airavatT_t domain.
airavatU_exec_t	Executable	Used to transition to the airavatU_t domain.
airavavat_user	User type	Trusted user who can transition to the airavatT_t domain.

Table 2: SELinux domains defined in Airavat and their usage.

Only `airavat_user` can execute files labeled with `airavatT_exec_t` and transition to the trusted domain. The system administrator maps a Linux user to `airavat_user`.

The untrusted domain, `airavatU_t`, has very few privileges. A process in the untrusted domain cannot connect to the network, nor read or write files. There are two exceptions to this rule. First, the untrusted domain can read configuration files of the type `airavatT_notsec_t`. Second, it can communicate with the trusted domain using pipes. All communication with the mapper happens via these pipes which are established by the trusted framework. A process can enter the untrusted domain by executing a file of the type `airavatU_exec_t`. In our implementation, the framework transitions to the untrusted domain by executing the JVM that runs the mapper code.

Each data provider labels its input files (**DB**) with a domain specific to that provider. Only the trusted `airavatT_t` domain can read files from all providers. The output of a computation is stored in a file labeled with the trusted domain `airavatT_rw_t`. Data providers may set their declassify flag if they agree to declassify the result when Airavat guarantees differential privacy. If all data providers agree to declassify, then the trusted domain label is removed from the result when differential privacy holds. If only a subset of the data providers agree to declassify, then the result is labeled by a new domain, restricted to entities that have permission from all providers who chose to retain their label. Since creating domains in SELinux is a cumbersome process, our current prototype only supports full declassification. DIFC makes this ad hoc sharing among domains easy.

6.2 Timing channels

A malicious mapper may leak data using timing channels. MapReduce is a batch-oriented programming style where most programs do not rely on time. The bandwidth of covert timing channels is reduced by making clocks noisy and low-resolution [25]. Airavat currently denies untrusted mappers access to the high-resolution processor cycle counter (TSC), which is accessed via Java APIs. A recent timing attack requires the high-definition processor counter to create a channel with 0.2 bits per second capacity [44]. Without the TSC, the data rate drops three orders of magnitude.

We are working to eliminate all obvious time-based APIs from the Airavat JVM for untrusted mappers, including `System.currentTimeMillis`. We assume an environment like Amazon's elastic MapReduce, where the only interface to the system is the MapReduce programming interface and untrusted mappers are the only untrusted code on the system. Untrusted mappers cannot create files, so they cannot use file metadata to measure time. Airavat eliminates the API through which programs are notified about garbage collection (GC), so untrusted code has only indirect evidence about GC through the execution of finalizers, weak, soft, and phantom references (no Java native interface calls are allowed). Channels related to GC are inherently noisy and are controlled by trusted software whose implementation can be changed if it is found to leak too much timing information.

Airavat does not block timing channels caused by infinite loops (non-termination). Such channels have low bandwidth, leaking one bit per execution. Cloud providers send their users billing information (including execution time) which may be exploited as a timing channel. Quantizing billing units (*e.g.*, billing in multiples of $10) and aggregating billing over long time periods (*e.g.*, monthly) greatly reduce the data rate of this channel. A computer system cannot completely close all time-based channels, but a batch-oriented system like MapReduce where mappers may not access the network can decrease the utility of timing channels for the attacker to a point where another attack vector would appear preferable.

7 Implementation

The Airavat implementation includes modifications to the Hadoop MapReduce framework and Hadoop file system (HDFS), a custom JVM for running user-supplied mappers, trusted reducers, and an SELinux policy file. In our prototype, we modified $2,000$ lines of code in the MapReduce framework, $3,000$ lines in HDFS, and 500 lines of code in the JVM. The SELinux policy is approximately 450 lines that include the type enforcement

rules and interface declarations. This section describes the changes to the HDFS, implementation details of the range enforcers, and JVM modifications.

7.1 HDFS modifications

An HDFS cluster consists of a single *NameNode* server that manages the file system namespace and a number of *DataNode* servers that store file contents. HDFS currently supports file and directory permissions that are similar to the discretionary access control of the POSIX model. Airavat modifies HDFS to support MAC labels, by placing them in the file `inode` structure. Inodes are stored in the NameNode server. Any request for a file operation by a client is validated against the inode label. In the DataNodes, Airavat adds the HDFS label of the file to the block information structure.

7.2 Enforcing sensitivity

As described in Section 5.1, each mapper has an associated range enforcer. The range enforcer determines the group for each input record and tags the output produced by the mapper with the `gid`. In the degenerate case when each input belongs to a group of its own, each output by the mapper is given a unique identifier as its `gid`. The range enforcer also determines and tags the outputs with the partition identifier, `pid`. The default is to tag each record as belonging to the same partition.

During the reduce phase, each reducer fetches the sorted key/value pairs produced by the mappers. The reducer then uses the `gid` tag to group together the output values. Any value that falls outside the range declared by the computation provider ($M_{min} \ldots M_{max}$) is replaced by a value inside the range. Such a substitution (if it happens) prioritizes privacy over accuracy (§ 5.1). The reducer also enforces that only key/value pairs with the correct `pid` are combined to generate the final output.

7.3 Ensuring mapper independence

To add the proper amount of noise to ensure differential privacy, the result of the mapper on each input record must not depend on any other input record (§ 5.3). A mapper is stateful if it writes a value to storage during an invocation and then uses this value in a later invocation. Airavat ensures that mapper invocations are not stateful by executing them in an untrusted domain that cannot write to files or the network. The MAC OS enforces the limitation that mappers cannot write to system resources.

For memory objects, Airavat adds access checks to two types of data: *objects*, which reside on the heap, and *statics*, which reside in the global pool. Airavat modifies the Java virtual machine to enforce these checks. Our prototype uses 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. Airavat modifies 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 (which is maintained by the trusted framework). 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 the JVM throws an exception. After this exception, the current map invocation is re-executed and the final output of the MapReduce operation is not differentially private and must be protected using MAC (without declassification).

Jikes RVM is not mature enough to run code as large and complex as the Hadoop framework. We therefore use Hadoop's streaming feature to ensure that mappers run on Jikes and that most of the framework executes on Sun's 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 pipes. This communication is secured by SELinux.

8 Evaluation

This section empirically makes the case that Airavat can be used to efficiently compute a wide variety of algorithms in a privacy-preserving manner with acceptable accuracy loss. Table 3 provides an overview of the case studies. Our experiments show that computations in Airavat incur approximately 32% overhead compared to those running on unmodified Hadoop and Linux. In all experiments except the one with the AOL queries, the mappers are untrusted. The AOL experiment outputs keys, so we trust the mapper not to encode information in the key.

8.1 Airavat overheads

We ran all experiments on Amazon's EC2 service on a cluster of 100 machines. We use the large EC2 instances, each with two cores of 1.0–1.2 GHz Opteron or Xeon, 7.5 GB memory, 850 GB hard disk, and running SELinux-enabled Fedora 8. The numbers reported are the average of 5 runs, and the variance is less than 8%. K-Means and Naive Bayes use the public implementations from Apache Mahout.[3]

Figure 4 breaks down the execution time for each benchmark. The values are normalized to the execution time of the applications running on unmodified Hadoop and unmodified Linux. The graph depicts the percentage

[2] www.jikesrvm.org

[3] http://lucene.apache.org/mahout/

Benchmark	Privacy grouping	Reducer primitive	#MapReduce computations	Accuracy metric
AOL queries	Users	`THRESHOLD, SUM`	Multiple	% Queries released
kNN recommender	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

Table 3: Details of the benchmarks, including the grouping of data, type of reducer used, number of MapReduce phases, and the accuracy metric.

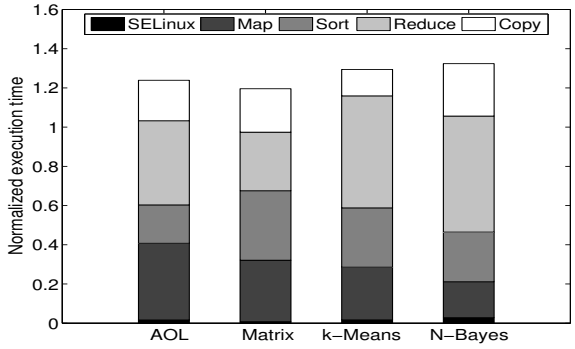

Figure 4: Normalized execution time of benchmarks when running on Airavat, compared to execution on Hadoop. Lower is better.

Benchmark	JVM overhead	Total overhead	Time (sec)	
AOL	36.3%	23.9%	228	±3
Cov. Matrix	43.2%	19.6%	1080	±6
k-Means	28.5%	29.4%	154	±7
Naive Bayes	37.4%	32.3%	94	±2

Table 4: Performance details.

of the total time spent in different phases, such as map, sort, and reduce. The category *Copy* 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 33%.

Table 4 measures the performance overhead of enforcing differential privacy. The JVM instrumentation, to ensure mapper independence, adds up to 44% overhead in the map phase.

8.2 Queries on AOL dataset

Recently, Korolova *et al.* showed how to release search queries while preserving privacy [28]. They first find the frequency of each query and then output the noisy count of those that exceed a certain threshold. Intuitively, the threshold suppresses uncommon, low-frequency queries, since such queries are likely to breach privacy.

We demonstrate how Airavat can perform similar computations on the AOL dataset, while ensuring differential privacy. Airavat does not output non-numeric values if the mapper is untrusted because non-numeric values can leak information (§5). The outputs of this experiment are search queries (which are non-numeric) and their frequencies, so we assume that the mapper is trusted. We use `SUM` and `THRESHOLD` as reducers to generate the frequency of distinct queries and then output those that exceed the threshold. The privacy group is the *user*, and M is the maximum number of search queries made by any single user. The mapper range is $(0, M)$. We vary M in our experiments.

Our experiments use the AOL data for the first week of April 2006 (253K queries). Since we use the threshold function, Airavat needs a non-zero δ as input. We chose $\delta = 10^{-5}$ based on the number of unique users for this week, 24,861. Fixing the value of ϵ and δ also determines the minimum threshold to ensure privacy. The exact threshold value can be calculated from the formula in section 5.5: $C = M(1 - \frac{ln(\frac{2\delta}{M})}{\epsilon})$.

It is possible that a single user may perform an uncommon search multiple times (*e.g.*, if he searches for his name or address). Releasing such search queries can compromise the user's privacy. The probability of such a release can be reduced by increasing M and/or setting a low value of δ. A large value of M implies that the release threshold C is also large, thus reducing the chance that an uncommon query will be released.

In our experiments, we show the effect of different parameters on the number of queries that get published. First, we vary M, the maximum number of search queries that belong to any one user. Figure 5(a) shows that as we increase the value of M, the threshold value also increases, resulting in a smaller number of distinct queries being released. Second, we vary the privacy parameter ϵ. As we increase ϵ, *i.e.*, decrease the privacy restrictions, more queries can be released. Note that fewer than 1% of total unique queries (109K) are released. The reason is that most queries are issued very few times and hence cannot be released without jeopardizing the privacy of users who issued them.

8.3 Covariance matrices

Covariance matrices find use in many machine-learning computations. For example, McSherry and Mironov recently showed how to build a recommender system that preserves individual privacy [39]. The main idea is to construct a covariance matrix in a privacy-preserving fashion and then use a recommender algorithm such as

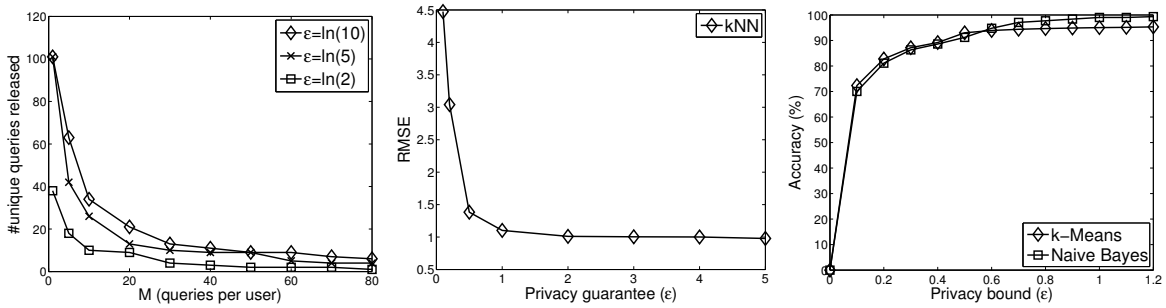

Figure 5: Effect of privacy parameter on the (a) number of released AOL search queries, (b) accuracy in RMSE in kNN recommender system (lower is better) and (c) accuracy in k-Means and Naive Bayes (higher is better).

k-nearest neighbor (kNN) on the matrix.

We picked $1,000$ movies from the Netflix prize dataset and generated a covariance matrix using Airavat. The computation protects the privacy of any individual Netflix user. We cannot calculate the complete matrix in one computation using the Airavat primitives. Instead, we fill the matrix cell by individual cell. The disadvantage of this approach is that the privacy budget is expended very quickly. For example, if the matrix has M^2 cells, then we subtract ϵM^2 from the privacy budget (equivalently, we achieve ϵM^2-differential privacy).

Because each movie rating is between 1 and 5 and an entry of the covariance matrix is a product of two such ratings, the mapper range is $(0, 25)$. Figure 5(b) plots the root mean squared error (RMSE) of the kNN algorithm when executed on the covariance matrix generated by Airavat. The x-axis corresponds to the privacy guarantee for the complete covariance matrix. Our results show that with the guarantee of 5-differential privacy, the RMSE of kNN is approximately 0.97. For comparison, Netflix's own algorithm, called Cinematch, has a RMSE of 0.95 when applied on the complete Netflix dataset.

8.4 Clustering Algorithm: k-Means

The k-Means algorithm clusters input vectors into k partitions. The partitioning aims to minimize intra-cluster variances. We use Lloyd's iterative heuristic to compute k-Means. The algorithm proceeds in two steps [6]. In the first step, the cardinality of each cluster is calculated. In the second step, all points in the new cluster are added up and then divided by the cardinality derived in the previous step, producing new cluster centers. The input dataset consists of 600 examples of control charts.[4] Control charts are used to assess whether a process is functioning properly. Machine learning techniques are often applied to such charts to detect anomaly patterns.

Figure 5(c) plots the accuracy of the k-Means algorithm as we change the privacy parameter ϵ. We assume that each point belongs to a different user whose privacy must be guaranteed. The mapper range of the computation that calculates the cluster size is $(0, 1)$. The mapper range for calculating the actual cluster centers is bounded by the maximum value of any coordinate over all points, which is 36 for the current dataset. We measure the accuracy of the algorithm by computing the intra-cluster variance. With $\epsilon > 0.5$, the accuracy of the clustering algorithm exceeds 90%.

8.5 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 previously unseen vectors.

Figure 5(c) plots the accuracy against the privacy parameter ϵ. We used the 20newsgroup dataset,[5] 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 value of ϵ affects the noise which is added to the model in the training phase. We measure the accuracy of the classifier by looking at the number of misclassified articles. An article contributes at most $1,000$ to a category of words, so the range for mapper outputs is $(0, 1000)$. Our results show that, for this particular dataset, we require $\epsilon > 0.6$ to achieve 95% accuracy.

9 Related work

Differential privacy guarantees are somewhat similar to robust or secure statistical estimation, which provides statistical computations with low sensitivity to any single input (*e.g.*, see [21, 23, 24]). While robust estimators do not by themselves guarantee privacy, they can serve as the basis for differentially private estimators [16].

[4] http://archive.ics.uci.edu/ml/databases/synthetic_control

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

In its current version, Airavat requires computation providers to provide an upper bound on the sensitivity of their code by declaring the range of its possible outputs in advance. An alternative is to have the enforcement system estimate local, input-specific sensitivity of the function computed by the code—either by re-running it on perturbed inputs, or by sampling from the input space [43]. Local sensitivity measures how much the output of the function varies on neighboring inputs from a subset of the function's domain. It often requires less noise to be added to the output in order to achieve the same differential privacy guarantee. We plan to investigate this approach in future work.

PINQ. Privacy Integrated Queries (PINQ) is a declarative system for computing on sensitive data [38] which ensures differential privacy for the outputs of the computation. Airavat mappers are Java bytecode, with restrictions on the programming model enforced at runtime. Mapper independence is an example of a restriction enforced by the language runtime which is absent from PINQ. PINQ provides a restricted programming language with a small number of trusted, primitive data operations in the LINQ framework. PINQ employs a request/reply model, which avoids adding noise to the intermediate results of the computation by keeping them on a trusted data server or an abstraction of a trusted data server provided by a distributed system.

Airavat's privacy enforcement mechanisms provide end-to-end guarantees, while PINQ provides language-level guarantees. Airavat's enforcement mechanisms include all software in the MapReduce framework, including language runtimes, the distributed file system, and the operating system. Enforcing privacy throughout the software stack allows Airavat computations to be securely distributed across multiple nodes, achieving the scalability that is the hallmark of the MapReduce framework. While the PINQ API can be supported in a similar setting (*e.g.*, DryadLINQ), PINQ's security would then depend on the security of Microsoft's common language runtime (CLR), the Cosmos distributed file system, the Dryad framework, and the operating system. Securing the levels below the language layer would require the same security guarantees as provided by Airavat.

Alternative definitions of privacy. Differential privacy is a relative notion: it assures the owner of any individual data item that the same privacy violations, if any, will occur whether this item is included in the aggregate computation or not. Therefore, no additional privacy risk arises from participating in the computation. While this may seem like a relatively weak guarantee, stronger properties *cannot* be achieved without making unjustified assumptions about the adversary [11, 12]. Superficially plausible but unachievable definitions include "the adversary does not learn anything about the data that he did not know before" [8] and "the adversary's posterior distribution of possible data values after observing the result of the computation is close to his prior distribution."

Secure multi-party computation [20] ensures that a distributed protocol leaks no more information about the inputs than is revealed by the output of the computation. The goal is to keep the intermediate steps of the computation secret. This technique is not appropriate in our setting, where the goal is to ensure that the output itself does not leak too much information about the inputs.

While differential privacy mechanisms often employ output perturbation (adding random noise to the result of a computation), several approaches to privacy-preserving data mining add random noise to inputs instead. Privacy guarantees are usually average-case and do not imply anything about the privacy of individual inputs. For example, the algorithm of Agrawal and Srikant [4] fails to hide individual inputs [3]. In turn, Evfimievski *et al.* show that the definitions of [3] are too weak to provide individual privacy [18].

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, such as ZIP code and date of birth [7, 48]. k-anonymity is achieved by syntactic generalization and suppression of these attributes (*e.g.*, [31]). 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 [34, 35]. It does not hide whether a particular individual is in the dataset [42], nor the sensitive attributes associated with any individual [32, 34]. Multiple releases of the same dataset or mere knowledge of the k-anonymization algorithm may completely break the protection [19, 52]. Variants, such as l-diversity [34] and m-invariance [50], suffer from many of the same flaws.

Program analysis techniques can be used to estimate how much information is leaked by a program [36]. Privacy in MapReduce computations, however, is difficult if not impossible to express as a quantitative information flow problem. The flow bound cannot be set at 0 bits because the output depends on every single input. But even a 1-bit leakage may be sufficient to reveal, for example, whether a given person's record was present in the input dataset or not, violating privacy. By contrast, differential privacy guarantees that the information revealed by the computation cannot be specific to any given input.

10 Conclusion

Airavat is the first system that integrates mandatory access control with differential privacy, enabling many privacy-preserving MapReduce computations without the need to audit untrusted code. We demonstrate the practicality of Airavat by evaluating it on a variety of case studies.

Acknowledgements

We are grateful to Frank McSherry for several insightful discussions and for helping us understand PINQ. We thank the anonymous referees and our shepherd Michael J. Freedman for the constructive feedback. We also thank Hany Ramadan for his help with the initial implementation. This research is supported by NSF grants CNS-0746888 and CNS-0905602, "Collaborative Policies and Assured Information Sharing" MURI, and a Google research award.

References

[1] *AppArmor*. `https://help.ubuntu.com/8.04/serverguide/C/apparmor.html`.

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

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

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

[5] 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.

[6] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the SuLQ framework. In *PODS*, 2005.

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

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

[9] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1), 2008.

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

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

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

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

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

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

[16] C. Dwork and J. Lei. Differential privacy and robust statistics. In *STOC*, 2009.

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

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

[19] S. Ganta, S. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *KDD*, 2008.

[20] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[21] F. Hampel, E. Ronchetti, P. Rousseeuw, and W. Stahel. *Robust Statistics - The Approach Based on Influence Functions*. Wiley, 1986.

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

[23] J. Heitzig. The "jackknife" method: confidentiality protection for complex statistical analyses. Joint UNECE/Eurostat work session on statistical data confidentiality, 2005.

[24] J. Hellerstein. Quantitative data cleaning for large databases. `http://db.cs.berkeley.edu/jmh/papers/cleaning-unece.pdf`, February 2008.

[25] W-M. Hu. Reducing timing channels with fuzzy time. In *S&P*, 1987.

[26] P. Karger, M.E. Zurko, D. Bonin, A. Mason, and C. Kahn. A retrospective on the VAX VMM security kernel. *IEEE Trans. Softw. Eng.*, 17(11), 1991.

[27] M. Kearns. Efficient noise-tolerant learning from statistical queries. *J. ACM*, 45(6), 1998.

[28] A. Korolova, K. Kenthapadi, N. Mishra, and A. Ntoulas. Releasing search queries and clicks privately. In *WWW*, 2009.

[29] 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.

[30] B. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10), 1973.

[31] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *SIGMOD*, 2005.

[32] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and ℓ-diversity. In *ICDE*, 2007.

[33] S. Lipner. A comment on the confinement problem. In *SOSP*, 1975.

[34] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.

[35] D. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Halpern. Worst-case background knowledge for privacy-preserving data publishing. In *ICDE*, 2007.

[36] S. McCamant and M. Ernst. Quantitative information flow as network flow capacity. In *PLDI*, 2008.

[37] B. McCarty. *SELinux: NSA's Open Source Security Enhanced Linux*. O'Reilly Media, 2004.

[38] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, 2009.

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

[40] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, 2007.

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

[42] M. Nergiz, M. Atzori, and C. Clifton. Hiding the presence of individuals from shared database. In *SIGMOD*, 2007.

[43] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *STOC*, 2007.

[44] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud! Exploring information leakage in third-party compute clouds. In *CCS*, 2009.

[45] I. Roy, D. Porter, M. Bond, K. Mckinley, and E. Witchel. Laminar: Practical fine-grained decentralized information flow control. In *PLDI*, 2009.

[46] I. Roy, S. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for MapReduce. Technical Report TR-10-09, UT-Austin, 2010.

[47] M. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 2009.

[48] L. Sweeney. k-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5), 2002.

[49] 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.

[50] X. Xiao and T. Tao. m-invariance: Towards privacy preserving re-publication of dynamic datasets. In *SIGMOD*, 2007.

[51] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in HiStar. In *OSDI*, 2006.

[52] L. Zhang, S. Jajodia, and A. Brodsky. Information disclosure under realistic assumptions: Privacy versus optimality. In *CCS*, 2007.