Towards a Data-centric View of Cloud Security

Wenchao Zhou[♭] Micah Sherr[♮] William R. Marczak[♯] Zhuoyao Zhang[♭]
Tao Tao[♭] Boon Thau Loo[♭] Insup Lee[♭]

[♭]University of Pennsylvania [♮]Georgetown University [♯]University of California at Berkeley

{wenchaoz, zhuoyao, taot, boonloo, lee}@cis.upenn.edu
msherr@cs.georgetown.edu wrm@berkeley.edu

ABSTRACT

Cloud security issues have recently gained traction in the research community, with much of the focus primarily concentrated on securing the operating systems and virtual machines on which the services are deployed. In this paper, we take an alternative perspective and propose a *data-centric* view of cloud security. In particular, we explore the security properties of secure data sharing *between* applications hosted in the cloud. We discuss data management challenges in the areas of secure distributed query processing, system analysis and forensics, and query correctness assurance, and describe our current efforts towards meeting these challenges using our *Declarative Secure Distributed Systems* (DS2) platform.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Security, Integrity, and Protection*; C.2.4 [**Computer Systems Organization**]: Computer Communication Networks—*Distributed Systems*; H.2.4 [**Database Management**]: Systems—*Query Processing*

General Terms

Design, Management, Security

Keywords

Cloud security, Secure data processing, System analysis and forensics, Declarative networking

1. INTRODUCTION

The economics of outsourcing data and computation will likely spur a continued migration of applications to the cloud. Just as the Internet is becoming dominated by applications that require data integration and sharing, we similarly expect that applications within the cloud will become increasingly interdependent. The trend toward interoperable cloud applications will require solutions that enable the secure communication and exchange of information between the cloud's users. In this paper, *cloud users* refer to content providers or portals that have deployed their applications and/or services on the cloud in order to serve a larger group of end-host *clients*.

A framework for secure data sharing in the cloud permits more diverse and feature-rich cloud applications, including (but not limited to): (i) retail portals that exchange product, inventory, and order information with smaller merchants in the cloud; (ii) scientific computations that can be partitioned into parallel jobs and processed by cloud customers with spare computational resources; (iii) privacy-protecting social network services that allow clients to have more control over how their data are published, stored, and shared; and (iv) enterprise cloud customers who conduct business online by selling and purchasing online services.

While cloud security has recently gained traction in the research community, much of this effort has focused on securing the underlying operating systems and virtual machines that host cloud services [8, 22, 34, 36, 37, 41]. This paper argues that a comprehensive solution has to go beyond OS and VM-centric security solutions and, in particular, must provide mechanisms for securely sharing, verifying, and tracing *data* as they flow between cloud users. Although most cloud providers enforce node access control (i.e., computation and data are distributed only to centrally managed and presumably secure cloud nodes), the integration and exchange of data between potentially dishonest cloud *users* presents several unique security challenges.

This paper introduces an extensible data-driven framework called *DS2* [1] that provides secure data processing and sharing between cloud users. DS2 is designed for collaborative deployments in which (potentially untrustworthy) cloud users share and exchange data. Our data-driven approach enables us to build upon well-studied database techniques, including database access control [38], query results verification of outsourced databases [14, 31, 33, 35], distributed query engines for enforcing extensible trust management policies [28, 29, 42] in the cloud ecosystem, and declarative techniques for cloud analytics [4].

This paper makes the following contributions. We first enumerate the data management challenges that face multi-user cloud environments. To meet these challenges, we propose the DS2 platform, which provides:

- secure query processing in a multi-user cloud environment;
- seamless integration of declarative access control policies with data processing to enable secure sharing among users;

CloudDB 2010, October 30, 2010, Toronto, Ontario, Canada.

- system analysis and forensics by capturing accurate historical records of data exchanges in the form of *distributed provenance*; and
- efficient end-to-end verification of data that are partitioned across both cloud nodes and cloud users

The remainder of the paper is organized as follows. We begin by motivating the need for a secure data exchange framework via a number of examples (Section 2), and then discuss some of the high-level challenges (Section 3) facing secure cloud data management. Next, we present an overview of the DS2 platform (Section 4) and describe how it helps meet many of the identified challenges (Sections 5 through 8). Finally, we conclude with a discussion of our ongoing and future work (Section 9).

2. MOTIVATING EXAMPLES

Due to their scalability, high quality of service, cost effectiveness, and reliability, clouds present an attractive platform for a variety of applications. Although most existing cloud application deployments consist of homogeneous and isolated software, *interconnecting* such applications to share data among cloud users enables more adaptable systems. Although the following list is not intended to be comprehensive, we briefly posit some potential applications that may benefit from inter-application information exchange.

Online Marketplaces. Retail portals such as Yahoo!, Amazon, buy.com, eBay, etc. serve as storefronts for merchants. These online marketplaces collect product and inventory information from sellers, and provide potential buyers with a unified store in which they may search for and purchase goods and services.

The economics of cloud computing allow both portal operators and merchants to benefit by moving their applications into the cloud. However, although cloud platforms offer services for exchanging data between virtual nodes belonging to the *same* cloud user, there is little existing support for secure query execution between *different* cloud users. DS2 provides a unified data integration language for portal operators and merchants to exchange data – for instance, to query available inventory, gather product descriptions and prices, process orders, and track shipments. DS2 supplies an authentication layer for communicated queries and result sets which could, for example: (i) prevent competing merchants from querying each others' inventories and prices, and (ii) allow portal operators to communicate payment information with only authorized parties. Additionally, portal operators can use DS2 as an access control framework to both restrict certain portal features to trusted merchants and offer differentiated pricing models in which merchants can pay more to further distinguish their products in the store.

Social Network Service in the Cloud. Social network services allow users to interact through an online social network topology. Most existing social network services (including Facebook, LinkedIn, renren, and Cyworld) rely on centralized architectures with end-host clients' public and private information stored in the service provider's databases. Complex and frequently changing privacy policies and practices have sometimes frustrated customers [6, 44], motivating the development of less centralized and more user-controllable social network services [5, 12, 13].

One promising approach to construct a more privacy-preserving social network service is to carefully store personal data in the cloud. Social networking customers could either host their own information on cloud nodes (such a model is compatible with ongoing efforts such as Diaspora* [12] and Appleseed [5]), or distribute portions of their data across federated cloud profile hosting services.

DS2 supports such cloud-based social network services by providing secure mechanisms for query execution and data computation. For example, traversing the social graph to discover potential connections (or in the parlance of Facebook, mining for friend "suggestions") is a distributed computation when profile data are disseminated throughout the cloud. DS2 assists such computations by ensuring that data are authenticated and that access controls are properly enforced. We demonstrate in Section 5 how DS2 can be used to program a secure version of MapReduce [10] that verifies the authenticity of data received from spawned workers; such an authenticated MapReduce service can be used in the context of social network services to efficiently locate potential connections.

Outsourced Data Storage and Query. A salient feature of cloud services is their ability to amortize the cost of storing large datasets. Cloud-based *data storage providers* may sell storage solutions that extend the features offered by the underlying cloud. For example, data storage providers may strategically locate data in the cloud to optimize content delivery, sell backup solutions, or offer data mining and query services.

Data owners may opt to disseminate their data across multiple such in-cloud storage providers to improve reliability (i.e., availability) or to take advantage of different features and pricing options. DS2 provides end-to-end verifiability of partitioned data, allowing end-clients to detect unauthorized data alteration, insertion, or deletion.

3. CHALLENGES

Based on the above examples, we identify three security challenges associated with cloud data management:

Challenge 1: Secure Query Processing and Data Sharing. In a centrally administered and tightly controlled enterprise environment, queries may be safely executed in the trusted corporate network. However, in distributed and decentralized environments such as the cloud, all parties may not behave honestly.

A key challenge of deploying data-centric applications in the multi-user cloud setting is to ensure that all querying processing and data sharing are carried out securely – participating parties should be authenticated and authorized, and user-specified access control policies should be strictly enforced.

Furthermore, the access control language should be sufficiently flexible to meet the needs of a large range of applications. For example, policies may accept or reject access based on authentication tokens, or access decisions may be based on derivation histories of communicated tuples. Although domain-specific solutions have been previously proposed, we argue that a *general-purpose* secure framework better enables developers to deploy their applications in diverse cloud environments.

Challenge 2: System Analysis and Forensics. Data exchanges and interaction among multiple cloud applications complicate the dependency logic of derived tuples. A misbehaving (and potentially malicious) user's input may have profound and intricate implications on other users' applications. Non-deterministic message transmission (e.g., message re-ordering or loss) and the dynamic nature of the cloud further complicate the problem. All of these issues underscore the necessity of an effective approach to systematically perform system analysis and forensics.

Challenge 3: Query Correctness Assurance. Cloud computing represents a cost-effective means of outsourcing computation. However, naïvely relying on (potentially untrustworthy) cloud applications to accurately process queries imposes a significant security risk. We desire techniques that provide *end-to-end verification* of query results.

Given the above challenges, we now describe our data-centric architecture for multi-user cloud security.

4. DS^2 PLATFORM

We first describe the DS2 platform that provides the functionality required by our proposed data security techniques. In DS2, network protocol and security policies are specified using *Secure Network Datalog* (SeNDlog) [42], a declarative language primarily rooted in Datalog that unifies declarative networking [27, 26] and logic-based access control [3] specifications. In prior work [42], we have demonstrated the flexibility and compactness of the SeNDlog language via secure specifications of Internet routing protocols, the Chord distributed hash table [40], the PIER [20] distributed query processor, and the A^3 extensible anonymity system [39].

SeNDlog programs are disseminated and compiled at each node into individual execution plans. When executed, these execution plans both implement the specified distributed protocol as well as enforce its desired security policies. By using declarative techniques, SeNDlog requires orders of magnitude less code than imperative languages [27].

SeNDlog extends the basic declarative networking language by adding support for authenticated communication. SeNDlog integrates two commonly used constructs in distributed trust management languages (e.g. Binder [11]): (i) the notion of *context* to represent a principal in a distributed environment and (ii) a distinguished operator `says` that abstracts away the details of authentication [11, 24].

To demonstrate the key language features of SeNDlog, we present a simple distributed program that computes pairwise reachability:

```
At S:
r1 reachable(S,D) :- link(S,D).
r2 reachable(D,Z)@D :- link(S,D), W says reachable(S,Z).
```

The program consists of two rules (`r1` and `r2`) that are executed in the context of node `S`. The program computes the set of reachable nodes (`reachable`) given the set of network links (`links`) between `S` and `D`.

Rule `r1` takes `link(S,D)` tuples, and computes single-hop reachability `reachable(S,D)`. Rule `r2` recursively computes multi-hop reachability, based on derived reachability facts asserted by node `W`. The `says` primitive in rule `r2` specifies that the authenticity of the received `reachable` tuple should be checked, ensuring that it originated from `W`. Finally, the location specifier `@D` in rule `r2` indicates that the evaluation result should be communicated to node `D`.

We highlight two language features of SeNDlog of relevance to cloud security, with additional details in [42]:

Communication Context. Due to the distributed nature of network queries, a principal does not have control over rule execution at other nodes. As described in more detail in Section 5, SeNDlog achieves secure distributed query processing by allowing programs to interoperate correctly and securely via the export and import of rules and derived tuples across contexts.

In the above example, the rules are in the context of `S`, where `S` is a variable assigned upon rule installation. In the multi-user cloud environment, `S` represents a cloud user or group of cloud users.

Import and Export Predicates. The SeNDlog language allows different contexts to communicate by importing and exporting tuples. The communication serves two purposes: (i) to disseminate maintenance messages, and (ii) to distribute the derivation of security decisions.

During the evaluation of SeNDlog rules, derived tuples can be communicated among contexts via the use of *import predicates* and *export predicates*. An import predicate is of the form "`N says p`", indicating that principal `N` asserts the predicate `p`.

The use of export predicates provides confidentiality by exporting tuples only to specified principals. An export predicate is of the form "`N says p@X`", where principal `N` exports the predicate `p` to principal `X`. In rule `r2`, node `S` exports `reachable` tuples to node `D` (as a shorthand, "`S says`" is omitted as `S` is where the rule resides).

The `says` operator implements an authentication scheme [24] that allows the receiver of the tuple to verify its source. The implementation of `says` depends on the system and its context. For example, `says` may require digital signatures. In a different cloud environment in which all applications are trusted, `says` may simply append a cleartext principal header to a message. Alternatively, cryptographic signatures may be applied only to certain important messages or when communicating with specific principals. DS2 provides sufficient flexibility to support a variety of cryptographic authentication primitives.

In addition to authentication, DS2 also provides mechanisms that optionally encrypt transmitted messages to protect the *confidentiality* of communication.

5. SECURE QUERY PROCESSING

DS2 aims to enable developers to specify cloud computations, data integration, and trust policies using a common declarative framework. Using SeNDlog as its basis, DS2 allows cloud users to both seamlessly integrate their services without exposing their confidential information as well as verify the authenticity of received data. To highlight these possibilities, we show an example based on an authenticated implementation of MapReduce written and implemented using SeNDlog.

5.1 Example: Authenticated MapReduce

MapReduce has been used extensively in cloud applications for efficiently utilizing parallel resources to solve certain classes of distributed problems. Generally, a MapRe-

duce job consists of two steps: *map* and *reduce*. At the map step, a master node splits the job into small sub-problems and distributes them to map workers. The results produced by the map workers are then collected at the reduce step, and are combined into a solution to the original problem.

In an untrusted environment, the map and reduce workers may be executed on untrusted nodes. In order to ensure correct execution of a MapReduce program, the nodes participating in the computation need to be authenticated. As an example, we consider the WordCount program in which MapReduce is used to count the occurrences of words in webpages. The following rules (`m1-m2` for map steps and `r1-r2` for reduce steps) demonstrate an *authenticated* implementation of MapReduce written in SeNDlog and executed via DS2:

```
At MW:
 m1 map(ID,Content) :- file(MW,ID,Content).
 m2 emits(MW,Word,Num,Offset)@RW :-
        word(Word,Num,Offset),
        reduceWorker(RID,RW), RID=f_SHA1(Word).

At RW:
 r1 reduceTuple(Word,a_LIST<Num>) :-
        MW says emits(MW,Word,Num,Offset).
 r2 reduce(Word,List) :- reduceTuple(Word,List),
        Master says rBegin(RW).
```

In the program shown above, rules `m1` and `m2` are within the context of a map worker `MW`, and rules `r1` and `r2` are in the context of a reduce worker `RW`.

Map Operation. Rule `m1` takes as input a `file` predicate and passes the `ID` and the `Content` of the file to the instances of the user-defined `Map` functions.

Upon receiving an (`ID`, `Content`) pair, each `Map` instance splits the content of the document into separate words, and generates a (WORD,1) pair (stored in `word` tuples) for each word, denoting that the occurrence count of the word should be increased by one. The `word` tuples, tagged with the `Offset` of each word in the document, are then sent back to `MW` as the result of the `map` function.

Rule `m2` takes as input `word` tuples and distributes them (in the form of `emits` tuples) to reduce workers. To enable authentication, a signature is included within each `emits` tuple using the `says` primitive.

Reduce Operation. Reduce workers receive and authenticate (e.g., via digital signatures) `emits` tuples from map workers. The tuples are then grouped by the key field `Word` (in rule `r1`). The `a_LIST` aggregate operator maintains the occurrences of each word in a list structure.

After the map workers complete their job, a master node sends a `rBegin` tuple to each reduce worker, signaling the start of the reduce job. `reduce` tuples are sent to the user-defined `Reduce` instances, each of which contains a word and the list of its occurrences. Based on these lists, the `Reduce` instances generate and emit the final results – the total occurrence counts of words.

Authentication. The above program enforces authentication using group signatures: reduce workers process an `emits` tuple as long as it is signed by *any* legitimate map worker. Alternatively, authentication can occur at finer granularity. For instance, a reduce worker may be configured to only accept tuples from cloud user `merchant123` by changing the clause "`MW says emits`" in rule `r1` to "`merchant123 says emits`".

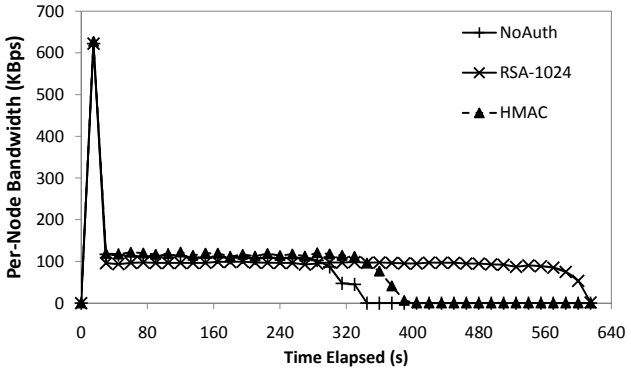

Figure 1: Per-node bandwidth (KBps) utilization

5.2 Preliminary Evaluation

We developed a prototype of WordCount using the RapidNet [32] declarative networking system. Our preliminary evaluation is intended (i) to experimentally validate DS2's ability to implement secure cloud applications using the MapReduce paradigm, and (ii) to study the overhead incurred by adding authentication features to MapReduce. As a workload, we use a two-phase version of our WordCount program. In the *Filtering Phase*, the master node distributes 6,400 randomly selected webpages from the Stanford WebBase project [2] to map workers. Map workers filter out all HTML tags and partition the results to reduce workers. Reduce workers then distribute the webpage fragments to the map workers of the *Counting Phase*. The Counting Phase uses the techniques described above to determine the occurrence counts of words in the webpages.

We perform the experiments within a local cluster of 16 quad-core machines. We deploy 16 map workers and 16 reduce workers for the Filtering Phase and 32 map workers and 128 reduce workers for the Counting Phase. Each physical machine runs a total of 12 MapReduce worker instances. To avoid packet-drops due to congestion, we rate-limit the number of packets sent per second.

To evaluate the overhead incurred by performing authentications, we constructed three versions of WordCount: *NoAuth*, *RSA-1024* and *HMAC*. In *NoAuth*, MapReduce workers transmit tuples without the sender's signature; in *RSA-1024* and *HMAC*, the communication between different map and reduce workers are authenticated using 1024-bit RSA signatures and SHA-1 HMACs, respectively.

Figure 1 shows the per-node bandwidth usage over time. All three versions of WordCount incur spikes in their bandwidth utilization in the first 30 seconds. The spikes are mainly attributed to the cross-node communication during the Filtering Phase. The MapReduce operation at this stage is computationally inexpensive but network intensive, as the tuples transmitted in this phase consist of relatively large chunks of documents (as compared to `word` tuples transmitted in the Counting Phase).

We observe that *NoAuth* finishes the computation in 350 seconds, whereas *HMAC* and *RSA-1024* incur an additional 17.4% (60s) and 78.3% (270s) overhead in query completion latency, respectively. The increase in query completion time is due primarily to the computation incurred by signature generation and verification. The respective aggregated communication overheads of *HMAC* and *RSA-1024* are 18.4%

(7.5MB) and 53.3% (21.8MB) higher than *NoAuth*. However, due to the use of network throttles in our evaluation, the per-node bandwidth utilization for the three versions are similar.

Note that while our example focuses on authentication, prior work [42, 28] demonstrate that the `says` construct is itself customizable, not only via different authentication schemes, but also encryption schemes for confidentiality and anonymity. This suggests that one can further extend our example here to implement MapReduce customized with other secure communication properties. We plan to explore this as part of our future work.

6. ACCESS CONTROL

Data sharing and integration are integral to many applications that operate in a multi-user cloud environment. The enforcement of access control policies can become complicated by the fact that users may frequently enter or leave the cloud, requiring policies to be rapidly updated. Real-world trust policies are complex and may require functionality beyond that offered by traditional tuple-level access control policies. Devising a scheme to respond to the dynamics and complexity of cloud environments represents a significant challenge.

In this section, we demonstrate how DS2 supports complex and dynamic data access control policies through query rewriting and other techniques. Our approach builds upon traditional database- and logic-based mechanisms for reasoning about trust policies.

6.1 View-based Access Control

Traditional databases typically utilize *view-based* access control in which views are expressed in SQL, with access controls to these views enforced via explicit permissions granted to authorized users. Such views are easily expressible in DS2. For example, if a user `alice` owns an `employees` predicate that stores the names, departments, and salaries of employees, then she may create a security view for some other user, `Bob`:

```
At alice:
 sv1 employee_sv_bob(Name,Dept) :-
        employee(Name,Dept,Salary), Salary < 5000.
 sv2 predsecview("employee","employee_sv_bob",bob).
```

In this example, `alice` allows `bob` to view only those employees in any department who have salaries less than $5000. The `employee_sv_bob` predicate represents both a horizontal (salary<5000) and vertical (salary column omitted) partition of the `employee` predicate. Note that `employee_sv_bob` (defined by rule `sv1`) is dynamically applied as the predicates in the rule body update, and hence its contents change with the contents of the `employee` predicate. Rule `sv2` maintains an additional predicate, `predsecview`, that associates security views with the protected predicate.

DS2 can easily express security views that apply to entities other than single users. For example, `alice` may delegate access to the users whom a certificate authority believes are `good` by replacing rule `sv2` with the following rule:

```
 sv2: predsecview("employee","employee_sv_good",U) :-
        cert_authority says good(U).
```

In addition to supporting simple horizontal and vertical slicing, DS2 permits more complex partitioning of access control. For example, access rights may be based on provenance information (e.g., Alice allows Bob to view only those employee tuples derived using data from Bob). In general, SeNDlog's flexibility enables DS2 to support a wide variety of access control policies.

Enforcement. We have thus far left unspecified how security views are enforced. Standard Datalog cannot prevent users from submitting a query that references predicates directly, nor can unmodified Datalog dynamically rewrite queries to refer to the security views. To enable these capabilities, we extend Datalog to represent the currently executing queries in tables accessible to the program – a concept we call the *DS2 meta-model*. Our approach is similar to recently proposed Datalog-based meta-compilers [9, 28].

Preventing queries from directly reading predicates (e.g., `employee`) is made possible by adding schema constraints called *meta-constraints* [29, 28] to the meta-model. Meta-constraints restrict the set of allowable queries. DS2 uses meta-constraints to express that users can only insert queries that refer to security views:

```
says(U,R), body(R,A), functor(A,P) -> predsecview(_,P,U).
```

The above example introduces the schema constraint format. Note that instead of `:-`, the head and body are separated by a right-arrow (`->`). The logical meaning of this format is that if, for any assignment of the variables, the left hand side (LHS) is true, then the right hand side (RHS) must also be true. If, on the other hand, the LHS is true but the RHS is false, then evaluation of the query terminates with an error.

In the above constraint, `says` associates a user `U` with a query `R`, and `body` and `functor` are meta-model predicates that examine the structure of the rule. The constraint expresses that every predicate mentioned in the body must be the security view of some other predicate (since the variable for this other predicate is unimportant in the constraint, we represent it with an underscore).

Code Generation. Unfortunately, constructing queries that abide by meta-constraints requires careful diligence by the programmer. To ease this burden, DS2 uses *code generation* to automatically rewrite queries to refer to security views.

```
-rulebody(R,B), +rulebody(R,C), +atom(C),
+atompred(C,S), +atomargs(C,N,V) :-
  P says R, rulebody(R,B), atompred(B,P),
  predsecurityview(P,S), predargs(S,N,V).
```

For example, the above *meta-rule* updates the body of queries by *retracting* (-) the body atom representing the original predicate and *asserting* (+) the body atom representing the security view. (The logical meaning of a conjunction of atoms in the head of a rule `q1,q2,...,qm :- p1,p2,...,pn` is the set of m rules: `qi :- p1,p2,...,pn.`)

Meta-rules provide programmers with a flexible access control framework. Developers may specify meta-rules to cause DS2 to transparently transform queries to adhere to various security policies, without requiring any changes to the program's data or control logics. In addition to access control, we have previously demonstrated [28] that meta-programming capabilities are also useful for customizing various forms of security mechanisms for authentication, encryption, and even anonymity.

6.2 Multi-user Multi-stage MapReduce

In Section 5, we introduced an authenticated version of MapReduce. Given that both access control policies and the MapReduce program are specified within SeNDlog, it is easy to integrate access control policies as additional rules in the program. The programmer needs only to supply a set of policy rules. DS2 automatically performs the query rewrites to generate the appropriate rules for executing the MapReduce program.

In practice, given that MapReduce operations can span multiple stages (i.e., compose several MapReduce operations), access control can be enforced across stages. For instance, if `alice` starts a MapReduce operation and the resulting output is used by `bob` for his subsequent MapReduce operation, one needs to ensure that `bob` accesses only data that he is authorized to view. One naïve solution is to enforce access control only at the boundaries of each MapReduce operation – i.e., on the resulting output between jobs. A more sophisticated solution that we are currently exploring is to "push-down" selection predicates dynamically into each MapReduce execution plan. This ensures that proper filtering is applied early so that only authorized data flows downstream to later stages of the MapReduce operations. In this example, the trust relationships between `alice` and `bob` may result in filtering of input data as early as `alice`'s MapReduce operation, causing only the data that `bob` is authorized to view to be processed. We are currently investigating methods to adapt the dynamic rewrite capabilities presented in [29] to handle more complex sharing, particularly across users and multiple stages of MapReduce computations.

7. DISTRIBUTED PROVENANCE

In an integrated cloud environment where multiple applications share information, it is useful to track the *source* of communicated and derived data. For example, if a cloud user `merchant123` is trusted at time t but later discovered to be deceitful at time t' ($t' > t$), it is critical that `merchant123`'s effects on other users' application states be determined and potentially rolled back. DS2 addresses this need by providing a mechanism for identifying where a tuple originated, which nodes it traversed, how it was derived, and which parties were involved in its derivation.

The database literature typically refers to such derivation information as *provenance* [7]. DS2's declarative rule-based derivation framework captures information flow as distributed queries, enabling natural support for acquiring, maintaining, and querying provenance in the cloud.

7.1 Usage of Provenance in the Cloud

The ability to identify the source of derived data enables several important functionalities in the cloud. The following list highlights some capabilities relevant to our multi-application cloud setting:

Diagnosis and Forensics. Provenance information is useful for debugging and error detection in the cloud. For example, tracing backwards in a network-level provenance graph may yield the discovery of the (possibly malicious) causes of suspicious query results. Provenance is also useful for *root cause analysis* (e.g., pinpointing the bottleneck in a multi-staged MapReduce execution to a slow machine or malicious user). In all scenarios, the querying of provenance can be automatically *triggered* by an anomalous behavior (e.g., a spike in traffic) that is detectable using a continuous query over existing network state.

Mitigation. Once incorrect or untrusted data have been identified, provenance enables developers to efficiently propagate corrections by targeting only the affected applications, reducing the number of expensive updates and rollbacks [25].

Provenance-based Trust Management. Provenance has been used in the context of p2p data integration systems as a basis to accept or reject access [21, 17]. Similarly, cloud application developers can specify access control policies that process or discard an incoming tuple based on its derivation. For instance, in a multi-staged MapReduce execution, a map worker may make the decision based on its trust relationship with the tuple's original producer. Provenance also allows the adoption of a *quantitative* approach for trust management: a derived tuple is assigned a trust value evaluated from its provenance, based on which a decision is made. Interestingly, the quantitative approach is computable and customizable by representing provenance in an algebraic form [16].

7.2 Computing Distributed Provenance

In general, provenance can be modeled as a directed graph that encodes the dependency relationships between base and derived tuples. The provenance graph consists of vertices that represent base tuples or intermediate evaluation results, and edges indicate the information workflow [15, 43]. This graph structure can be stored as relational tables in a straightforward manner.

In the context of the cloud, data are distributed across nodes and applications, and hence the provenance information is itself distributed. We have previously explored the design space of *distributed provenance* in declarative networks, and demonstrated that incrementally updateable and bandwidth efficient implementations of distributed provenance are achievable at Internet-scale [43].

In a nutshell, given a declarative networking protocol, by using an automatic rewrite process, one can compute and maintain distributed provenance as distributed views over computed network state, thus allowing provenance maintenance to be automatically handled by the incremental view maintenance mechanism[1] provided in DS2.

To query provenance, a distributed recursive query is issued over the distributed view, and this results in a traversal of the provenance graph. To customize each query, each user can supply additional annotation schemes schemes [16, 23] in the form of user-defined functions. This allows the query results to be returned in a desired format (e.g., contributing base tuples, traversed cloud peers, and confidentiality levels), to match to the requirement of various applications. See [43] for more details.

Maintaining and querying distributed provenance in a cloud setting raises open research challenges that we intend to explore. First, our previous work focused primarily on enabling distributed provenance support at the network layer. Since fault analysis is typically a cross-cutting concern, en-

[1]In our DS2 system, incremental view maintenance is leveraged by *Pipelined Semi-naïve* (PSN) evaluation, which is essentially an asynchronous version of classic semi-naïve evaluation used for Datalog programs.

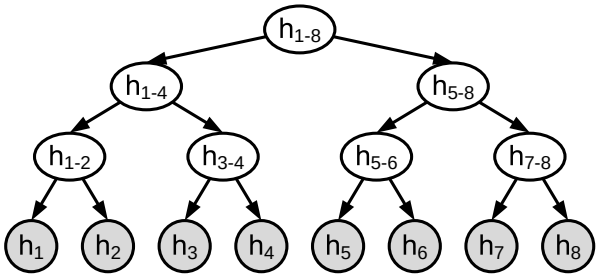

Figure 2: MHT for table X

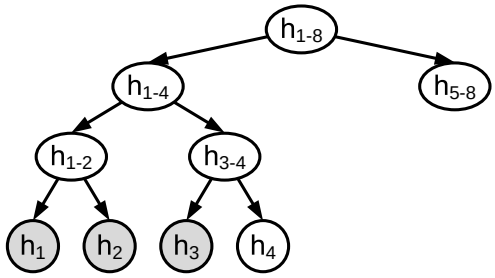

Figure 3: P-MHT for table X_1

abling useful provenance information in the cloud may require integrating system state across various layers of the cloud computing stack (application, middleware, network, OS, etc.), several of which may run in legacy code outside of DS2.

Second, since malicious nodes may intentionally manipulate provenance information, the *integrity* of provenance data must be protected. Fortunately, existing cryptographic approaches [19] that ensure that unauthorized modification of provenance data will be eventually detected are applicable here. Provenance data should also remain *confidential* and not be revealed to unauthorized parties. Cryptography can be used to hide provenance information based on roles, security levels, or other customized requirements. A more challenging security problem is detecting when malicious nodes purposefully *withhold* provenance information. We are currently exploring the use of fault-detection [18] techniques to address this particular threat.

8. END-TO-END QUERY VERIFICATION

Our final research direction explores verification mechanisms that ensure that the results of distributed queries can be verified. Here, we adapt a threat model in which the owner of the data is trustworthy, but some fraction of the cloud applications that host the data (and respond to queries) is malicious.

Our initial approach adopts techniques used in verifying the correctness of *outsourced databases* [14, 31, 35]. At a high level, end-to-end verification works by requiring cloud applications to transmit *verification objects* (VOs) along with query results. These VOs are then assessed by end-host clients to verify the correctness of the tuples in the returned result sets.

For example, previous work [35] has applied Merkle Trees (MHTs) [30] to verify the results of range queries processed by online servers. MHTs are constructed over presorted data, with leaves corresponding to the hashes of database tuples in a table and non-leaf nodes corresponding to the hash of the concatenation of the children. The MHT is built recursively to the root of the tree. The VO that is attached to the result of a range query consists of (i) a signature over the MHT's root node, signed by the table's owner; (ii) the tuple with the greatest value that is lower than the minimum specified in the query range (if such a tuple exists); (iii) the tuple with the least value that is greater than the maximum specified in the query range (if such a tuple exists); and (iv) the smallest set of internal nodes (excluding the root) that is required for the client to independently compute the root of the MHT (additional details are provided in [35]). The VO contains sufficient information to guarantee that no data have been omitted or inserted to/from the query results, and that no entry in the returned result sets has been modified.

To illustrate, consider table $X = \{x_1, x_2, x_3, ..., x_8\}$. The MHT of X is shown in Figure 2, where h_i is the digest of tuple x_i. The VO for a query that returns $\{x_1, x_2\}$ is $\{x_3, h_4, h_{5-8}, \texttt{SIG}(h_{1-8})\}$, where $\texttt{SIG}(h_{1-8})$ denotes a copy of the root node that has been signed by the table's owner. When combined with $\{x_1, x_2\}$, the VO is sufficient for the client to compute the root of the MHT and verify the signature. Note that the VO authenticates the resultset since a modified, inserted, or omitted tuple will cause the computed root's digest to differ from that of the signed copy.

MHTs in the Cloud. We propose an adaptation of the above MHT-based verification scheme for multi-user cloud environments. Our approach, which we call *partitioned MHT* (P-MHT), operates over partitioned data. P-MHTs enable clients to verify the correctness of range queries executed across multiple cloud applications.

When data are distributed across different cloud applications, tables may be horizontally partitioned such that each application maintains a range of tuples in a sorted table. Suppose table X (see Figure 2) is partitioned into three sub-tables — $X_1 = \{x_1, x_2, x_3\}$, $X_2 = \{x_4, x_5, x_6\}$, and $X_3 = \{x_7, x_8\}$ — each of which is hosted by a separate (and potentially malicious) cloud application. In addition to storing the tuples, each application maintains a P-MHT that embeds sufficient information for the *local* generation of the VO for any tuple in the sub-table. That is, a P-MHT is an MHT over a partition of data.

Figure 3 shows the P-MHT for X_1. Although the P-MHT is a partial MHT, it may still generate the VO for any tuple maintained in X_1. For example, the P-MHT maintains sufficient information to generate the VO for the query that returns $\{x_1, x_2\}$ (i.e. the VO is $\{x_3, h_4, h_{5-8}, \texttt{SIG}(h_{1-8})\}$), thus allowing a client to verify the query result, provided that he has obtained $\texttt{SIG}(h_{1-8})$ from the (trusted) data owner.

Our approach works for tables that are ordered and range partitioned across nodes. Verifying query results across hash-based partitions is an area of future work.

9. CONCLUSION AND FUTURE WORK

This paper outlines the challenges of secure cloud data management and proposes several security mechanisms based on the DS2 framework that move us closer toward meeting these challenges. As a basis for further exploration, we have implemented an initial DS2 prototype, developed using the

RapidNet declarative networking engine [32]. The SeNDlog language/compiler and distributed provenance implementation is available for download at http://netdb.cis.upenn.edu/rapidnet/downloads.html.

Our future work is proceeding in several directions. First, we are exploring use-cases of DS2 for securing a range of cloud applications, ranging from large-scale data processing to data sharing and analysis. Our paper has presented one example based on MapReduce, but the applicability of DS2 is sufficiently broad to enable a variety of data sharing applications on the cloud, particularly those related to social networking and cloud analytics [4]. To bootstrap the usage of DS2, we are exploring utilizing DS2 as a separate extensible trust management policy engine incorporated into existing distributed computing software (e.g. Hadoop), and potentially integrating our system with existing cloud analytics tools.

A security solution for secure cloud data management is not complete unless there are mechanisms for cloud providers and users to debug, analyze, and diagnose distributed queries executed in cloud environments. We believe that distributed provenance is an important step toward realizing a secure cloud data management infrastructure. We have recently added provenance support to the DS2 platform, and are actively developing techniques for ensuring the integrity and confidentiality of distributed provenance in the cloud.

10. ACKNOWLEDGMENTS

This work is supported in part by NSF IIS-0812270, NSF CNS-0831376, NSF CNS-0834524, ONR MURI N00014-07-1-0907, and OSD/AFOSR MURI on Collaborative Policies and Assured Information Sharing.

11. REFERENCES

[1] DS2: Declarative Secure Distributed Systems. http://netdb.cis.upenn.edu/ds2/.

[2] Stanford WebBase. http://diglib.stanford.edu:8091/~testbed/doc2/WebBase/.

[3] M. Abadi. Logic in Access Control. In *Proc. LICS*, 2003.

[4] P. Alvaro, T. Condie, N. Conway, K. Elmeleegy, J. Hellerstein, and R. Sears. BOOM Analytics: Exploring Data-Centric, Declarative Programming for the Cloud. In *Proc. EuroSys*, 2010.

[5] The Appleseed Project. http://opensource.appleseedproject.org/.

[6] N. Bilton. Price of Facebook Privacy? Start Clicking. *The New York Times*, 12 May 2010.

[7] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Proc. ICDT*, 2001.

[8] M. Christodorescu, R. Sailer, D. L. Schales, D. Sgandurra, and D. Zamboni. Cloud Security is not (just) Virtualization Security. In *Proc. CCSW*, 2009.

[9] T. Condie, D. Chu, J. M. Hellerstein, and P. Maniatis. Evita raced: Metacompilation for declarative networks. In *Proc. VLDB*, 2008.

[10] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. OSDI*, 2004.

[11] J. DeTreville. Binder: A logic-based security language. In *Proc. IEEE S&P*, 2002.

[12] Diaspora*. http://www.joindiaspora.com.

[13] J. Dwyer. Four Nerds and a Cry to Arms Against Facebook. *The New York Times*, 11 May 2010.

[14] M. H. Feifei Li and G. Kollios. Dynamic authenticated index structures for outsourced databases. In *Proc. SIGMOD*, 2006.

[15] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *Proc. VLDB*, 2007.

[16] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. PODS*, 2007.

[17] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. ORCHESTRA: Facilitating Collaborative Data Sharing. In *Proc. SIGMOD*, 2007.

[18] A. Haeberlen, P. Kuznetsov, and P. Druschel. PeerReview: Practical Accountability for Distributed Systems. In *Proc. SOSP*, 2007.

[19] R. Hasan, R. Sion, and M. Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *Proc. FAST*, 2009.

[20] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proc. VLDB*, 2003.

[21] Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *Proc. CIDR*, 2005.

[22] M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono. On Technical Security Issues in Cloud Computing. In *Proc. CLOUD*, 2009.

[23] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *Proc. SIGMOD*, 2010.

[24] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed Systems: Theory and Practice. *ACM TOCS*, 1992.

[25] M. Liu, N. Taylor, W. Zhou, Z. Ives, and B. T. Loo. Recursive computation of regions and connectivity in networks. In *Proc. ICDE*, 2009.

[26] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing Declarative Overlays. In *Proc. SOSP*, 2005.

[27] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative Routing: Extensible Routing with Declarative Queries. In *Proc. SIGCOMM*, 2005.

[28] W. R. Marczak, S. S. Huang, M. Bravenboer, M. Sherr, B. T. Loo, and M. Aref. SecureBlox: Customizable Secure Distributed Data Processing. In *Proc. SIGMOD*, 2010.

[29] W. R. Marczak, D. Zook, W. Zhou, M. Aref, and B. T. Loo. Declarative reconfigurable trust management. In *Proc. CIDR*, 2009.

[30] R. C. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford University, 1979.

[31] K. Mouratidis, D. Sacharidis, and H. Pang. Partially materialized digest scheme: an efficient verification method for outsourced databases. *VLDB Journal*, 2009.

[32] S. C. Muthukumar, X. Li, C. Liu, J. B. Kopena, M. Oprea, and B. T. Loo. Declarative toolkit for rapid network protocol simulation and experimentation. In *SIGCOMM (demo)*, 2009.

[33] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *Proc. NDSS*, 2004.

[34] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *Proc. CCGRID*, 2009.

[35] H. Pang and K.-L. Tan. Verifying Completeness of Relational Query Answers from Online Servers. *ACM TISSEC*, 2008.

[36] R. Perez, L. van Doorn, and R. Sailer. Virtualization and Hardware-Based Security. In *Proc. IEEE S&P*, 2008.

[37] H. Raj, R. Nathuji, A. Singh, and P. England. Resource Management for Isolation Enhanced Cloud Services. In *Proc. CCSW*, 2009.

[38] S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proc. SIGMOD*, 2004.

[39] M. Sherr, A. Mao, W. R. Marczak, W. Zhou, B. T. Loo, and M. Blaze. A3: An Extensible Platform for Application-Aware Anonymity. In *Proc. NDSS*, 2010.

[40] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. SIGCOMM*, 2001.

[41] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning. Managing Security of Virtual Machine Images in a Cloud Environment. In *Proc. CCSW*, 2009.

[42] W. Zhou, Y. Mao, B. T. Loo, and M. Abadi. Unified Declarative Platform for Secure Networked Information Systems. In *Proc. ICDE*, 2009.

[43] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In *Proc. SIGMOD*, 2010.

[44] M. Zuckerberg. From Facebook, Answering Privacy Concerns with New Settings. *The Washington Post*, 24 May 2010.