LIMITING DISCLOSURE FOR DATA STREAMS IN THE CLOUD

Wee Siong Ng, Huayu Wu, Wei Wu and Shili Xiang
*Institute for Infocomm Research, A*STAR, Singapore, Singapore*

Keywords: Access Control, Limited Disclosure, Cloud Computing, Stream Data Management.

Abstract: The rapidly increasing number of sensors and surveillance devices as well as the coming of age of pervasive and cloud computing are fostering applications driven by real-time stream data management. As more and more data stream processing engines (and services) will be deployed in the cloud, we feel it is critical to enable the data stream owners to control who can access their data streams for what purposes under what conditions. Therefore, we design an architecture to support data stream management in the cloud with privacy-preserving capabilities. In this paper, we focus on one of the premier principles of data privacy, limited disclosure. We design an access control framework, to define privacy policies and efficiently enforce these privacy policy rules from stream-level to tuple-level granularity.

1 INTRODUCTION

The rapidly increasing number of sensors and surveillance devices are generating tremendous amount of streaming data to transform our physical world into computing platform. On the other hand, there is growing demand for applications from a large variety of domains to consume these real-time data. This fuels the need for real-time stream management. However, data streaming environments are highly-demanding environments, where issues such as the sheer size of data, distributed and heterogeneous nature of data, and dynamic rates of data make it challenging for data providers and consumers to cope with (Golab and Özsu, 2003).

We believe that using cloud service (Vaquero et al., 2008) to manage and process stream data is promising for many existing and foreseeable applications thanks to cloud computing's elasticity. For example, multiple stream processing engine instances can be deployed in multiple virtual machines to make the stream engine's capacity dynamically scale with data stream rate (Knauth and Fetzer, 2011).

However, increasing privacy and confidentiality requirements impose a major obstacle. Very often, stream data convey information related to individuals or information which are critical to organizations. While companies collecting and processing sensitive stream data (in the cloud) from users have realized that they have to cope with privacy issues, what they do is only showing a privacy policy statement to the users and promise that they will only use the collected information according to the privacy policy. The users do not have real control on how their data are used.

In our project, we endeavor to provide an architecture that supports data stream management in the cloud with privacy-preserving capabilities. One focus of our system is to provide data owners with the ability to control how their data is managed and used by a particular organization or individual. Our system is called HipCloudS, which extends the functionality of traditional data stream management to satisfy Hippocratic principles (Agrawal et al., 2002).

In the process of realizing the HipCloudS, we focus on providing limited disclosure. Out of the Hippocratic principles, *Limited Disclosure* is one of the most premier principles of data privacy. The limited disclosure principle is based on the premise that a stream owner should be given total control over who is accessing his/her data, under which circumstances and for what purpose. That is, the system is not allowed to release any information to a user without the owner's approval. Limiting disclosure is particularly critical to ease data owners' reluctance to provide their data.

In our system, data stream owner specifies privacy policies, and these policies are thereafter enforced with respect to all data access. Privacy policies are expressed in high-level privacy specification languages. Essentially, a privacy policy is comprised of a set of rules that describe which part of data may be disclosed to who (the users) and how the data may

Ng W., Wu H., Wu W. and Xiang S..
LIMITING DISCLOSURE FOR DATA STREAMS IN THE CLOUD.
DOI: 10.5220/0003927005740579
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 574-579
ISBN: 978-989-8565-05-1

be used (the purposes). Additional conditions may be specified to govern disclosure as well.

We adopt an approach similar to EPAL (Enterprise Privacy Authorization Language) (Ashley et al., 2003) to design an access control model. However, our access control model is different from EPAL in that our model is specifically designed for an open platform (e.g., cloud) where there are many data stream owners and many data stream consumers while EPAL is designed for an individual enterprise. Each data stream owner manages his/her own access control policies for his/her own data stream. This work also differs from EPAL in that we elaborate on the management of the components in our access control model and present our strategy of enforcing the access control policies in a data stream query processing environment.

2 RELATED WORK

Lindner and Meier (Lindner and Meier, 2006) design a filtering operator and apply it to the stream query processing results to filter the output based on relevant access control policies. Similar to many other post-processing approaches, performance becomes the major problem of this attempt, because a large set of useless intermediate result may be generated.

Nehme et al. (Nehme et al., 2008) embed security policies into data stream, by security punctuations (SPs). Their query processor analyzes the SPs in each data stream, and enforces the policies during query processing. This framework is further improved by supporting dynamic access authorization of query issuers (Nehme et al., 2010; Nehme et al., 2009). Compared to the filtering approach, the punctuation-based framework has better performance, as useless intermediate result can be avoided.

Carminati et al. (Carminati et al., 2010) propose another framework to enforce access control policies for stream query processing. They model continuous queries as graph of algebraic operators, and focus on query rewriting to incorporate policies into query graphs. Finally, the rewritten query graph can be translated into different query languages according to different stream query processors (Cao et al., 2009). The difference between (Carminati et al., 2010) and our system is that we use a different model for access control policies, in which user roles, data categories and query purposes are all modeled as hierarchies. Our model is more comprehensive to cope with most general cases in applications.

Recently, Raman and Thomas (Adaikkalavan and Perez, 2011) argue that enforcing security checking during continuous query processing, such as query plan re-generation in (Nehme et al., 2008) and query rewriting in (Carminati et al., 2010), may affect the sharing of similar query execution. However, they only offer a simple solution for sharing exactly the same query, rather than sharing the same operator across different queries. The latter is more general for multi-query processing.

This work differs from existing work in that our system focus on access control on data streams contributed by many stream owners in an open (i.e., cloud) environment. In our access control model each data stream owner can manage his/her own data category and set access control policies at stream level, attribute level, and tuple level. The way we enforce access control policies is also unique: we check stream level and attribute level access control conditions before a query is admitted to the system while enforcing tuple-level conditions during query processing.

3 SYSTEM OVERVIEW

The design paradigm of the HipCloudS system is based on Cloud Computing. Our system deploys the SaaS (Software as a Service) paradigm and runs on top of existing cloud provider's PaaS (Platform as a Service) or IaaS (Infrastructure as a Service) service offerings. This approach enables the development of Hippocratic data stream applications by providing a set of Hippocratic Data Stream Service APIs to allow data to be queried or analytic algorithms to be executed (as indicated in Figure 1).

Due to space limit, we will only briefly describe the components that are closely related to access control in our system.

The *Stream Producer* and *Stream Consumer* communicate with the HipCloudS through Web Service interfaces: Stream Producer Service and Stream Consumer Service. While invoking a Stream Producer Service, a user is allowed to use the Stream Definition Language (SDL) statements to define the schema of a new stream along with the privacy preferences of the stream owner. On the other hand, a stream consumer can invoke a Stream Consumer Service to submit queries in the Stream Manipulation Language (SML), e.g., an SQL query.

The *Privacy Controller* is the core of the privacy policy enforcement unit to assure the compliance of limited disclosure, and to enforce access control. Basically, it examines the set of queries for each purpose in order to determine if any information is being disclosed violating the stream providers' privacy policy.

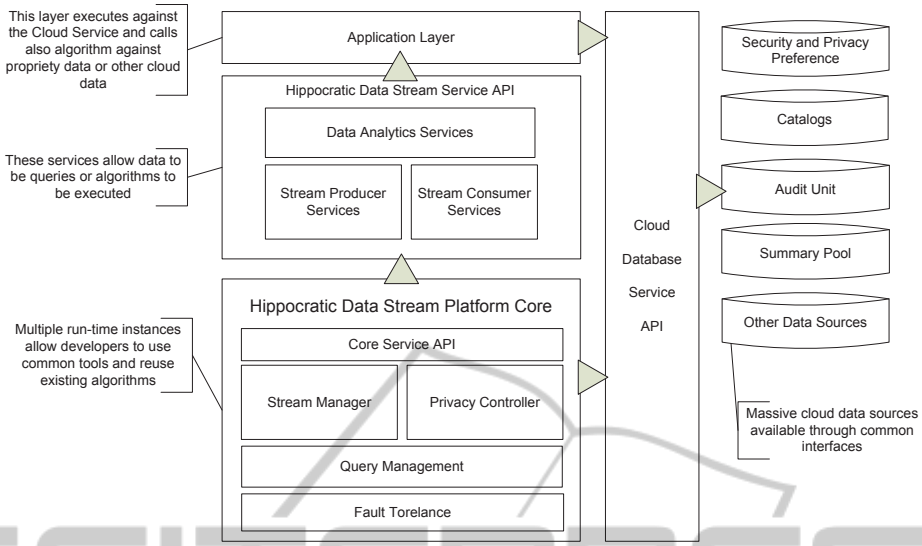

Figure 1: HipCloudS – A logical design.

4 PRIVACY CONTROLLER

In this section we describe the privacy controller in our HipCloudS. In particular, we present the access control model that we designed for *limited disclosure* and *privacy protection* in HipCloudS, and briefly present how the access control policies are enforced during query processing.

We will use the following taxi data stream in our writing as a user case. CompanyX is a taxi service provider. The taxis of this company are all equipped with GPS devices. They periodically send their status information including time, longitude, latitude, speed, and occupancy-status to our system as a data stream. In this case, the stream is "taxi", and we use "t", "x", "y", "v", and "s" to refer to the the stream attributes. The taxi company wants to share the taxi data stream (since it is very useful in many applications) but at the same time limit the disclosure of data to avoid potential privacy problems. For example, they want to make the data stream only available to traffic management authority and to researchers in a specific department of a specific research institute.

4.1 Access Control Model

In our access control model, users and data are organized in hierarchical structures. The hierarchical structures of users is called the *user-category*. The hierarchical structures of data, including streams and stream attributes, is called the *data-category*. Basically, the access control model enables a data stream owner to specify who can access which part of his/her data stream.

The other two important concepts in our data access model are *purpose-category* and *condition*s. The purpose-category is a classification of data access purposes. A stream owner can specify that a user can access the data stream only for a specific purpose. For example, the owner of the taxi stream can specify that users from TrafficAuthority can access this data stream only for Traffic-Management purpose. Conditions are provided for data stream owners to set tuple-level access control which is discussed below.

Our access control model supports three levels of access control, namely *stream-level*, *attribute-level*, and *tuple-level* access control. The stream-level access control enables a stream provider (a.k.a. stream owner) to grant the access to the stream to a group of users or a specific user. The attribute-level access control empowers a stream owner to control the access to stream attributes. The tuple-level access control lets a stream provider grant access to specific stream tuples satisfying certain conditions to a group of users or a specific user. For example, the owner of the taxi data stream can specify that user A can access the whole taxi data stream, user B can access the latitude, longitude and speed attributes of the taxi stream, and user C can access the tuples in the taxi stream if the status of the taxi is FREE. They are examples of stream-level, attribute-level, and tuple-level access control respectively.

4.1.1 Access Control Policies

In our access control model, a data stream owner con-

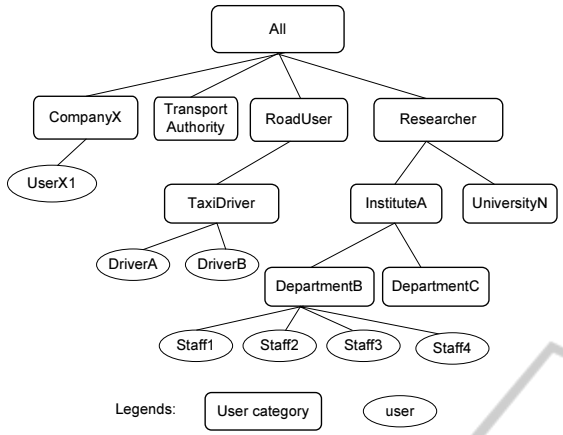

Figure 2: An example of a User-Category hierarchy.

trols the access to his/her stream data by setting access control policies.

An access control policy is represented as:

User-category, data-category, purpose-category, conditions

It means the users under the specified *user-category* can access the data under the specified *data-category* for the specified *purpose-category* if the specified *conditions* on stream tuples are satisfied.

While the user-category, data-category, and purpose-category are necessary for an access control policy, the conditions are optional. In our current data access model, we require that the conditions (if given) to be represented in SQL selection conditions syntax, i.e., the expressions used in SQL WHERE clause. For example, if the owner of taxi stream wants to set a condition to only allow a user to access tuples in which the speed of the taxi is less than 80km per hour, he/she would write "taxi.v $<$ 80" as the condition in his/her access control policy.

Now we describe the user-category, data-category, and purpose-category hierarchical structures. We will also elaborate on who can manage these hierarchical structures in our HipCloudS.

4.1.2 User Category

User category is modeled as a tree.

- Nodes of the tree are categories.
- Leaves of the tree are users.
- The root of the user-category tree is "All".

Figure 2 shows an example of the user-category hierarchical structure. Note that users are the leaves of the tree. A user has the access privileges of all its ancestors. For example, if a stream owner specifies that

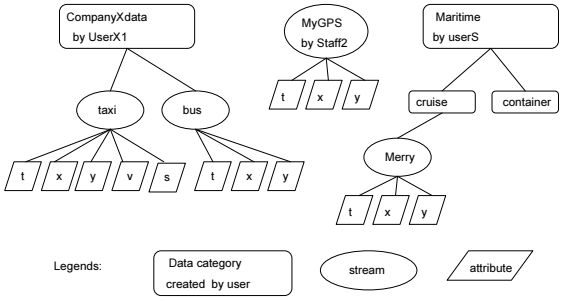

Figure 3: An example of Data-Category hierarchies.

"Researcher" category can access his/her stream, then user "Staff1" is allowed to access the data stream because "Staff1" is under the "Research" category. If a stream owner wants to make her stream open to all the users of the system, she can set "All" as user-category in her access control policy.

The user-category is managed by HipCloudS *system administrator*. When a user registers in the system, its category is "All". A user can request to join a specific user-category. A user can also request the system administrator to create a new category (e.g., "CompanyX"). The system administrator can accept/reject the request. The system administrator can assign a user to a specific category.

4.1.3 Data Category

Data-categories are modeled as a **forest** where the stream owners can manage their own trees in the forest.

- A tree in the forest is a data-category.
- A stream itself can form a data-category, i.e., a tree.
- A leaf of a tree is a stream attribute.
- Parent of a leaf is a stream.
- Parent of a stream, if applicable, is a category.

Data categories are created and managed by stream providers. A stream owner can only set access control policies for her own data-categories.

Figure 3 shows an example of data-categories. In this example, three stream providers "UserX1", "Staff2", and "userS" have created their own data-categories. UserX1 created data categories for her company's data streams including taxi data stream and bus data stream. The attributes in the taxi stream include time (t), longitude (x), latitude (y), and speed (v), and status (s).

Before a stream provider registers a stream in the system, she can create data-categories and sub data-categories. A stream provider can also choose not to

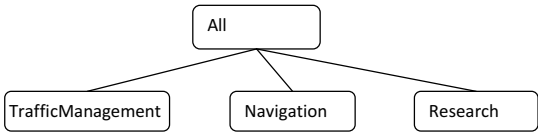

Figure 4: An example of Purpose-Category hierarchy.

create data categories for her streams. In this case, his/her streams are independent trees (in the forest) and each tree forms its own data-category.

A data access policy specified on a node in a data-category is applicable to all the nodes and leaves in its subtree. For example, if user "UserX1" specifies that Researcher can access "CompanyXdata", then Researcher can access all the data under the "CompanyXdata" category including taxi and bus data streams and their attributes.

4.1.4 Purpose Category

The main objective of having the purpose category is to record the purpose of each data access query so that this information can be logged and audited when necessary.

Purpose category is modeled as a Tree.

- Nodes of the tree are categories.
- The root of the purpose-category tree is "All".

The purpose-category is managed by system administrator. Figure 4 shows an example of a purpose-category.

4.1.5 Specifying Access Control Policy

Access control policies are managed by stream providers. A stream provider can only specify access control policies for her own data-categories, streams, and stream-fields. For example, user UserX1 (and owner of data category CompanyXdata in Figure 3) can specify the following access control policies:

```
DepartmentB, taxi, research, taxi.v <
80

TransportAuthority, CompanyXdata,
traffic-management
```

The first access control policy means that users under the "DepartmentB" user category can access the tuples in the "taxi" stream where speed (attribute v) is less than 80km per hour for "research" purpose. The second access control policy means that users under the "TransportAuthority" can access all the data under "CompanyXdata" category for "traffic-management" purpose.

As another example, user "Staff2" (i.e., owner of data category MyGPS) can specify the following policies:

```
All, MyGPS, All, (hour(t)>8 AND
hour(t)<18)
```

This access control policy means that all users can access the "MyGPS" stream for all purposes if the time in data stream tuple if between 8am to 6pm. Note that this condition is specified with the condition expression (hour(t)>8 AND hour(t)<18).

4.2 Enforcing the Access Control Policies

Our system will enforce all the access control policies set by the data stream owners. Here we briefly present our design for access control policies enforcement.

We differentiate *stream-level* access control and *attribute-level* access control from *tuple-level* access control. We enforce stream-level and attribute-level conditions when a query is submitted to our system for running. For tuple-level access conditions, we combine the conditions with the original query through query rewriting.

When a query (together with its purpose) is submitted to the system by a user for processing, a query parser will first check syntax and extract the stream and stream attributes that the query will access. These information together with the query will be passed on to do stream-level (and attribute level) checking. This phase of checking looks at the category of the user who submitted the query, the data the query wants to access, and the purpose of the query (given by the query owner) and checks whether there is an access control policy (specified by the stream owner) that allows the query to be executed. If so, the query passes the stream-level and attribute level checking.

The query rewriting function will check whether the applicable access control policy has tuple-level conditions. If so, they will be combined with the original query to form a new query. Since the rewritten query contains the tuple-level access control conditions, the query engine will enforce these conditions during query processing.

For example, user "Staff2" may submit the following query to the system and select "Research" as the purpose.

```
SELECT t, x, y FROM taxi WHERE
x>103.81 AND x<103.86
```

Our access control policy enforcement component will find out the the query wants to access stream

"taxi" and its attributes "t", "x", and "y". During stream-level and attribute-level checking, our system will find out that the access control policy "DepartmentB, taxi, research, taxi.v $<$ 80" allows the query to be executed, because user "Staff2" is under user-category "DepartmentB", and the stream and attributes are under data-category "taxi", and the purpose is also allowed. The query rewriting function then will transform the query to the following query for execution.

```
SELECT t, x, y FROM taxi WHERE
x>103.81 AND x<103.86 AND v<80
```

Note that the condition "taxi.v $<$ 80" in the access control policy is attached to the query as another selection condition. This makes the query processing engine to help enforce the tuple-level access control during query processing time.

The overhead on policy enforcement in our system is acceptable. Based on our evaluation, the time of stream/attribute-level policy checking, i.e., searching user, data and purpose categories, is linear to the number policies and is constant to the size (fanout and height) of each category. For the tuple-level access control enforcement, i.e., query rewriting to incorporate policy constraints, if the selectivity of original query is high, policy checking will not bring in any obvious overhead; if the query selectivity is not high, the overhead of checking access control policy could be either positive or negative, depending on the selectivity of the policy. Due to space limit, we do not describe our prototype implementation and detailed performance evaluation result in this paper.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a framework to provide limited disclosure to data stream management systems deployed in cloud. The key concept of our design is to incorporate a privacy policy enforcement component, which is called privacy controller, into the normal stream system. In our access control model, users, data, and data access purposes are organized into hierarchies. The access control model enables each stream owner to specify who can access what data for what purpose under what condition. All stream level, attribute level, and tuple level access controls are possible. The privacy controller enforces stream-level and attribute-level access control immediately when a query is registered into the system. By this way, unauthorized access to particular streams and attributes can be prevented at the very first time. Then tuple-level access control is achieved by query rewriting, i.e., adding tuple-level access control constraints to original queries.

Another focus of our project is to make our Hippocratic data stream system cloud friendly, i.e., make it able to scale with elastic computing resources. To this end, we are investigating the problem of deploying multiple instances of our Hippocratic data stream system on multiple (virtual or physical) machines and making them collaborate with each other to scale with the dynamic stream rate and query processing load.

ACKNOWLEDGEMENTS

This work was supported by A*STAR Grant No. 102 158 0037.

REFERENCES

Adaikkalavan, R. and Perez, T. (2011). Secure shared continuous query processing. In *SAC*, pages 1000–1005.

Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. (2002). Hippocratic databases. In *VLDB*, pages 143–154.

Ashley, P., Hada, S., Karjoth, G., Powers, C., and Schunter, M. (2003). Enterprise Privacy Authorization Language (EPAL 1.2). Technical report, IBM.

Cao, J., Carminati, B., Ferrari, E., and Tan, K.-L. (2009). ACStream: Enforcing access control over data streams. In *ICDE*, pages 1495–1498.

Carminati, B., Ferrari, E., Cao, J., and Tan, K. L. (2010). A framework to enforce access control over data streams. *ACM Trans. Inf. Syst. Secur.*, 13:28:1–28:31.

Golab, L. and Özsu, M. T. (2003). Issues in data stream management. *SIGMOD Record*, 32(2):5–14.

Knauth, T. and Fetzer, C. (2011). Scaling non-elastic applications using virtual machines. In *IEEE CLOUD*, pages 468–475.

Lindner, W. and Meier, J. (2006). Securing the borealis data stream engine. In *IDEAS*, pages 137–147.

Nehme, R. V., Lim, H.-S., and Bertino, E. (2010). FENCE: Continuous access control enforcement in dynamic data stream environments. In *ICDE*, pages 940–943.

Nehme, R. V., Lim, H.-S., Bertino, E., and Rundensteiner, E. A. (2009). StreamShield: a stream-centric approach towards security and privacy in data stream environments. In *SIGMOD*, pages 1027–1030.

Nehme, R. V., Rundensteiner, E. A., and Bertino, E. (2008). A security punctuation framework for enforcing access control on streaming data. In *ICDE*, pages 406–415.

Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39:50–55.