Fine-grained Access Control with Hive

Anuj Gupta, Mehul Vyas, Nikhil Mishra, Vaibhav Khadilkar, Dr. Murat Kantarcioglu, Dr. Latifur Khan and Dr. Bhavani Thuraisingham

19 July 2010
Fine-grained Access Control with Hive

Outline

1. Hive
   - Overview

2. Architecture
   - Overview
   - Table and View definition using Hive
   - RBAC using XACML policies
   - Hive-based Access Control

3. Future Steps
Hive is an Apache Hadoop subproject

Hive is a data warehouse infrastructure built on top of Hadoop that provides tools to enable easy data summarization, adhoc querying and analysis of large datasets data stored in Hadoop files

It provides a mechanism to put structure on this data and it also provides a simple query language called Hive QL which is based on SQL and which enables users familiar with SQL to query this data

At the same time, this language also allows traditional map/reduce programmers to be able to plug in their custom mappers and reducers to do more sophisticated analysis which may not be supported by the built-in capabilities of the language
Overview

- Our framework provides the following functions
  - Uploading data to HDFS as tables in Hive.
  - Defining or uploading XACML policies for newly created tables.
  - Defining views on existing tables in Hive.
  - Defining or uploading XACML policies on views.
  - Querying tables and views.
  - Registration of new users.
  - Assigning new users to groups. This is done by a designated user “admin”.
We have used a part of the Freebase system for this demo. Specifically we have used people, music, film, awards, sports, organizations and business datasets from within the Freebase system.

Sample commands used to load data into Hive,

- CREATE TABLE american_indian_group ( name STRING, id STRING, us_indian_reservation STRING ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘\t’ STORED AS TEXTFILE;
- LOAD DATA LOCAL INPATH ‘/home/cloud/Hive-AC/Data/people/american_indian_group.tsv’ OVERWRITE INTO TABLE american_indian_group;
Sample view definition,
CREATE VIEW view_american_indian_group ( hasName STRING, hasId STRING ) AS
SELECT DISTINCT name, id
FROM american_indian_group
ORDER BY name;
RBAC using XACML policies

- Define RBAC policies using XACML
  - Define the mapping between roles and users,
    `SELECT admin mehul anuj nikhil`
    `INSERT admin mehul nikhil`
  - Define the mapping between tables or views and roles,
    `american_indian_group SELECT`
    `view_american_indian_group SELECT INSERT`
  - Generate XACML policies using the above mappings using SUN’s XACML implementation
Hive-based Access Control

- A simple login page is provided for users. We use the salted hash technique to store passwords in a .txt file. On logging in the user is provided with options for creating/loading tables/views based on their credentials.

- Table/View definition and loading,
  - Users can create tables as well as load data into tables. Further, they can also upload XACML policies for the table they are creating. Users can also create XACML policies for tables/views.
  - Users can define views only if they have permissions for all tables specified in the query used to create the view. They can also either specify or create XACML policies for the views they are defining.
Hive-based Access Control

- **Table/View Querying**
  - A user simply enters his/her query. We use the ZQL Java parser to get the table and view names in the FROM clause.
  - Using the XACML policy engine we evaluate if the user has the permissions to execute the given query on the underlying tables and views. If the engine permits the user we run the query else we deny execution of the query.
  - On query execution, the results are displayed to the user as a series of pages. The user is also given the option of downloading the results as a file.
Future Steps

- Work resulting from using the ZQL parser,
  - Does not support the DESCRIBE, JOIN keywords or other complex queries, need to change this parser or write our own

- Work resulting from Hive,
  - Does not support multiple tables in the FROM clause, currently we replace the ‘comma’ between multiple tables in the FROM clause to the JOIN keyword so that the query is executed on Hive
  - Hive requires the keyword ‘OVERWRITE’ in an INSERT query, this is manually added by us to an INSERT query

- This framework is tested with SELECT and INSERT queries
- Implement an effective paging mechanism for results being returned
- Implement query rewriting based on permissions a user has for the underlying tables and views