Secure Data Storage and Retrieval in the Cloud

Agenda

- Motivating Example
- Current work in related areas
- Our approach
  - Contributions of this paper
  - System architecture
- Experimental Results
- Conclusions and Future Work

Motivating Example

- ***Current Trend:*** Large volume of data generated by Twitter, Amazon.com and Facebook
- ***Current Trend:*** This data would be useful if it can be correlated to form business partnerships and research collaborations
- ***Challenges due to Current Trend:*** Two obstacles to this process of data sharing
  - Arranging a large common storage area
  - Providing secure access to the shared data

Motivating Example

- ***Addressing these challenges:***
  - Cloud computing technologies such as Hadoop HDFS provide a good platform for creating a large, common storage area
  - A data warehouse infrastructure such as Hive provides a mechanism to structure the data in HDFS files. It also allows adhoc querying and analysis of this data
  - Policy languages such as XACML allow us to specify access controls over data
  - This paper proposes an architecture that combines Hadoop HDFS, Hive and XACML to provide fine-grained access controls over shared data

Current Work

- Work has been done on security issues with cloud computing technologies
  - Hadoop v0.20 proposes solutions to current security problems with Hadoop
  - This work is in its inception stage and proposes simple access control list (ACL) based security mechanism
- Our system adds another layer of security above this security
- As the proposed Hadoop security becomes robust it will only strengthen our system

Current Work

- Amazon Web Services (AWS) provide a web services infrastructure platform in the cloud
- To use AWS we would need to store data in an encrypted format since the AWS infrastructure is in the public domain
- Our system is “trusted” since the entire infrastructure is in the private domain

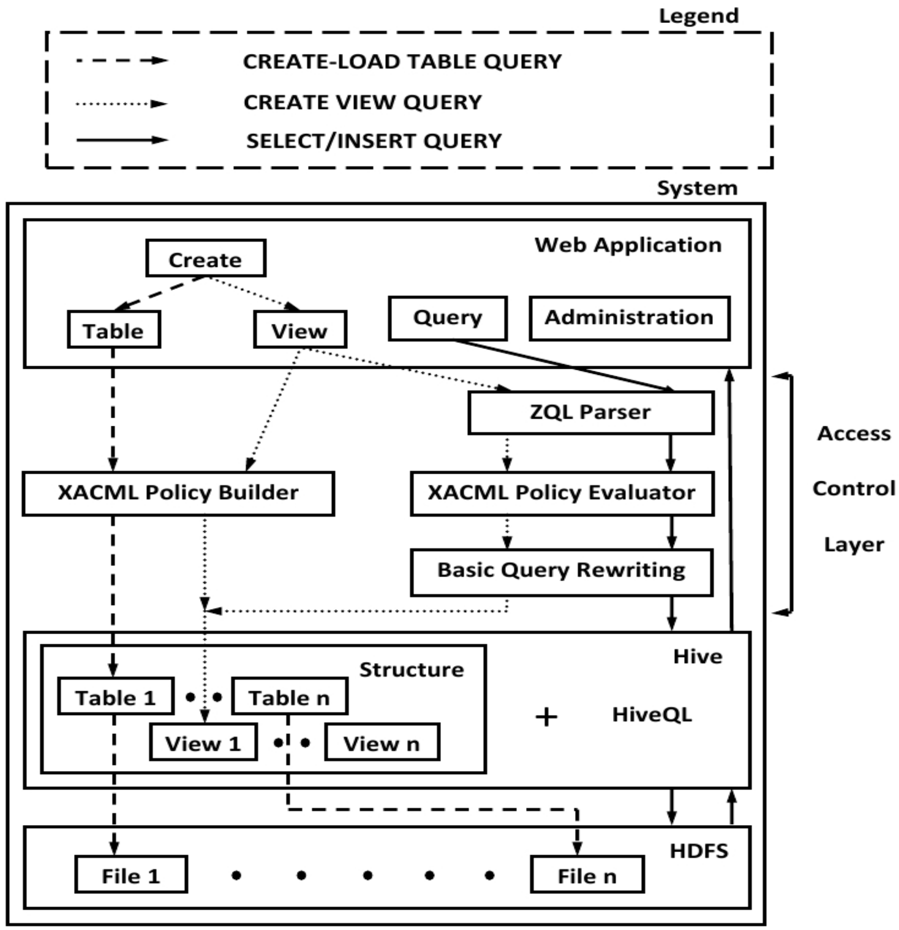
Current Work

- The Windows Azure platform is an Internet-scale cloud computing services platform
- This platform is suitable for building new applications but not to migrate existing applications
- We did not use this platform since we wanted to port our existing application to an open source environment
- We also did not want to be tied to the Windows framework but allow this system to be used on any platform

Contributions of this paper

- Create an open source application that combines existing open source technologies such as Hadoop and Hive with a policy language such as XACML to provide fine-grained access control over data
- Ensure that the new system does not create a performance hit when compared to using Hadoop and Hive directly

System Architecture



System Architecture - Web Application Layer

- This layer is the only interface provided by our system to the user
- Provides different functions based on a user's permissions
  - users who can query the existing tables/views
  - users who can create tables/views and define policies on them in addition to being able to query
  - an "admin" user who in addition to the above can also assign new users to either of the above categories
- We use the salted hash technique to store usernames/passwords in a secure location

System Architecture - ZQL Parser Layer

- ZQL is a Java based SQL parser
- The Parser layer takes as input a user query and continues to the Policy layer if the query is successfully parsed or returns an error message
- The variables in the SELECT clause are returned to the Web application layer to be used in the results
- The tables/views in the FROM clause are passed to the Policy evaluator
- The parser currently supports SQL DELETE, INSERT, SELECT and UPDATE statements

System Architecture - XACML Policy Layer

- XACML Policy Builder
  - Tables/Views are treated as resources for building policies
  - We use a table/view to query-type mapping

    table1 SELECT INSERT

    view1 SELECT

    to create policies using Sun's XACML implementation
  - Since a view is constructed from one or more tables, this allows us to define fine-grained access controls over the data
  - A user can upload their own pre-defined policies or have the system build the policy for them at the time of table/view creation

System Architecture - XACML Policy Layer

- XACML Policy Evaluator
  - Use the query-type to user mapping

    SELECT user1 user2

    INSERT user1 user3

    to extract the kinds of queries that a user can execute
  - Use Sun's implementation to verify if a given query-type can be executed on all tables/views that are defined in any user query
  - If permission is granted for all tables/views, the query is processed further, else an error is returned
  - The policy evaluator is used during query execution as well as during table/view creation

System Architecture - Basic Query Rewriting Layer

- Adds another layer of abstraction between a user and HiveQL
- Allows a user to enter SQL queries that are rewritten according to HiveQL's syntax
- Two simple rewriting rules in our system:
  - SELECT a.id, b.age FROM a, b;

    $\Rightarrow$ SELECT a.id, b.age FROM a JOIN b;
  - INSERT INTO a SELECT * FROM b;

    $\Rightarrow$ INSERT OVERWRITE TABLE a SELECT * FROM b;

System Architecture - Hive Layer

- Hive is a data warehouse infrastructure built on top of Hadoop
- Hive allows us to put structure on files stored in the underlying HDFS as tables/views
- Tables in Hive are defined using data in HDFS files while a view is only a logical concept in Hive
- HiveQL is used to query the data in these tables/views

System Architecture - HDFS Layer

- The HDFS is a distributed file system designed to run on basic hardware
- In our framework, the HDFS layer stores the data files corresponding to tables created in Hive
- ***Security Assumption***
  – Files in HDFS can neither be accessed using Hadoop's web interface nor Hadoop's command line interface but only using our system

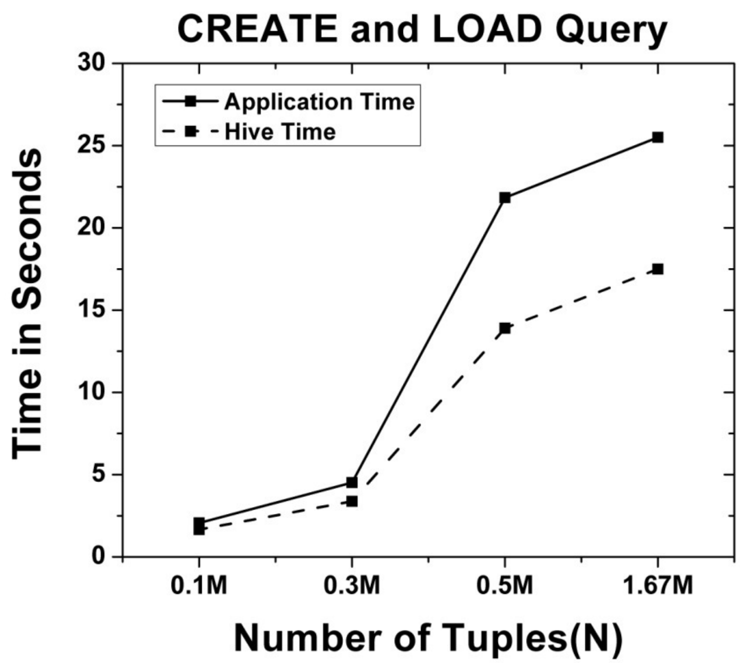
Experiments and Results

- Two datasets
  - Freebase system - an open repository of structured data that has approximately 12 million topics
  - TPC-H benchmark - a decision support benchmark that consists of a typical business organization schema
- For Freebase we constructed our own queries while for TPC-H we used Q1, Q3, Q6 and Q13 from the 22 benchmark queries
- Tested table loading times and querying times for both datasets

Experiments and Results

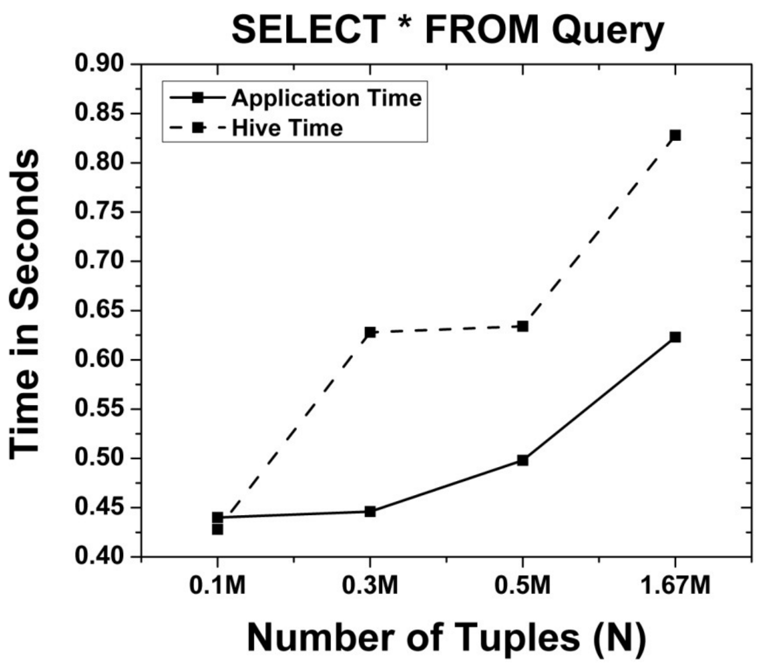
- Our system currently allows a user to upload files that are at most 1GB in size
- All loading times are therefore restricted by the above condition
- For querying times with larger datasets we manually added the data in the HDFS
- For all experiments XACML policies were created in such a way that the querying user was able to access all the necessary tables and views

Experiments and Results - Freebase



- Loading time of our system *versus* Hive is similar for small sized tables
- As the number of tuples increases our system gets slower
- This time difference is attributed to data transfer through a Hive JDBC connection to Hadoop

Experiments and Results - Freebase

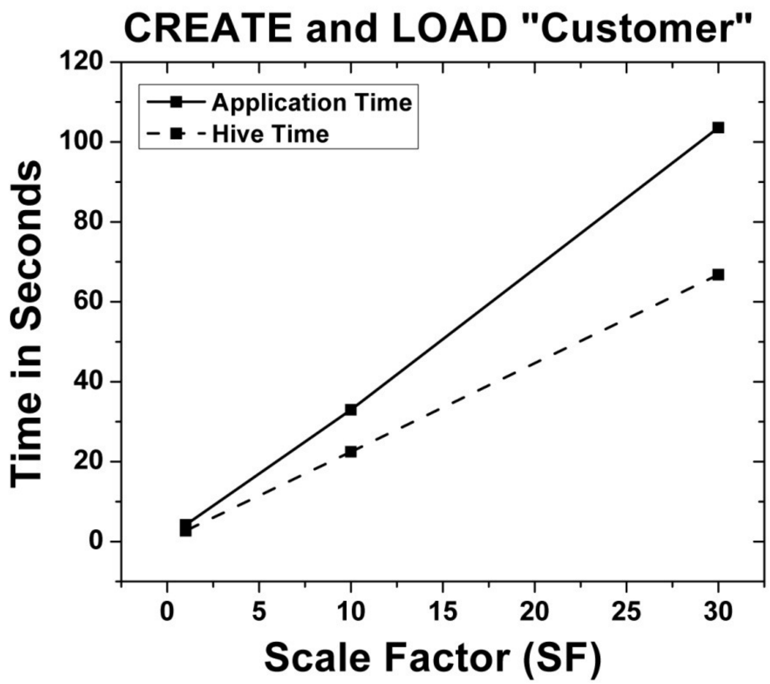


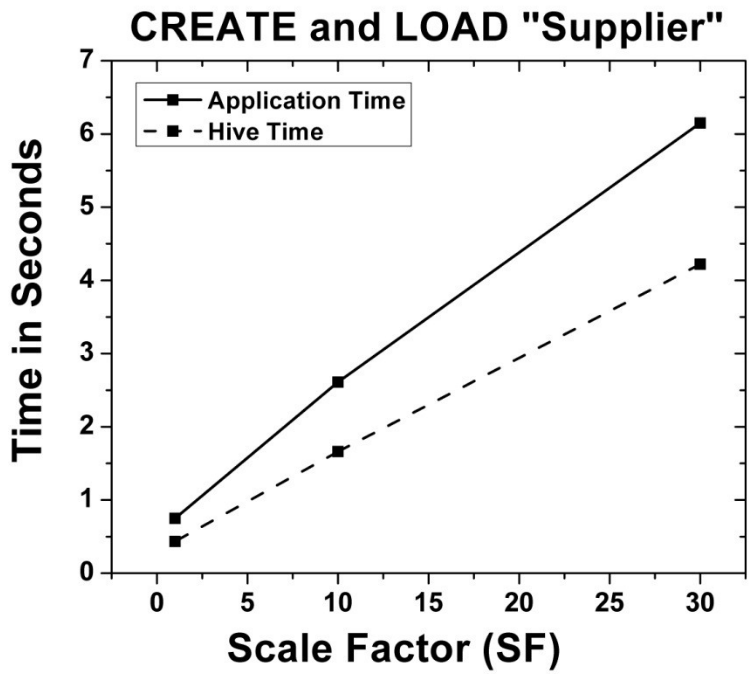
- Our running times are slightly faster than Hive
- This is because of the time taken by Hive to display results on the screen
- Both running times are fast because Hive does not need a Map-Reduce job for this query, but simply returns the entire table

Experiments and Results - Freebase

Query	System Time (sec)	Hive Time (sec)
SELECT name, id FROM Person LIMIT 100;	27.1	28.4
SELECT id FROM Person WHERE name='Frank Mann' LIMIT 100;	30.2	30.5
CREATE VIEW Person_View AS SELECT name, id FROM Person;	0.19	0.11

Experiments and Results - TPC-H





- Similar to the Freebase results, our system gets slower as the number of tuples increases
- The trend is linear since the tables sizes increase linearly with the Scale Factor

Experiments and Results - TPC-H

Query	Scale Factor (SF)	System Time (sec)	Hive Time (sec)
Q6	100	605.24	590.66
	300	1815.45	1806.4
	1000	6240.33	6249.68
Q3	100	1675.19	1670.77
	300	7532.23	7511.52
	1000	61411.21	61390.71

Experiments and Results - TPC-H

Query	Scale Factor (SF)	System Time (sec)	Hive Time (sec)
Q13	100	870.70	847.52
	300	1936.35	1910.19
	1000	7322.54	7304.39
Q1	100	1210.04	1209.79
	300	5407.14	5411.62
	1000	42780.67	42768.83

Conclusions

- A system was presented that allows secure sharing of large amounts of information
- The system was designed using Hadoop and Hive to allow scalability
- XACML was used to provide fine-grained access control to the underlying tables/views
- We have combined existing open source technologies in a unique way to provide fine-grained access control over data
- We have ensured that our system does not create a performance hit

Future Work

- Extend the ZQL parser with support for more SQL keywords
- Extend the basic query rewriting engine into a more sophisticated engine
- Implement materialized views in Hive and extend HiveQL with support for these views
- Extend the simple security mechanism with more query types such as CREATE and DELETE
- Extend this work to include public clouds such as Amazon Simple Storage Services