Secure Data Storage and Retrieval in the Cloud

Vaibhav Khadilkar, Anuj Gupta, Murat Kantarcioglu, Latifur Khan, Bhavani Thuraisingham
The University of Texas at Dallas
800 W. Campbell Road
Richardson, TX 75080

***Abstract*—With the advent of the World Wide Web and the emergence of e-commerce applications and social networks, organizations across the world generate a large amount of data daily. This data would be more useful to cooperating organizations if they were able to share their data. Two major obstacles to this process of data sharing are providing a common storage space and secure access to the shared data. In this paper we address these issues by combining cloud computing technologies such as Hive and Hadoop with XACML policy based security mechanisms that provide fine-grained access to resources. We further present a web-based application that uses this combination and allows collaborating organizations to securely store and retrieve large amounts of data.**

***Index Terms*—Cloud Computing, Hadoop, Hive, Web applications**

I. INTRODUCTION

The World Wide Web (WWW) is envisioned as a system of interlinked hypertext documents that are accessed using the Internet [1]. With the emergence of organizations that provide e-commerce such as Amazon.com[1] and social network applications such as Facebook[2] and Twitter[3] on the World Wide Web, the volume of data generated by them daily is massive [2]. It was estimated that the amount of data that would be generated by individuals in the year 2009 would be more than that generated in the entire history of mankind through 2008 [3]. The large amount of data generated by one organization could be valuable to other organizations or researchers if it can be correlated with the data that they have [4]. This is especially true for various governmental intelligence organizations. This has led to another trend of forming partnerships between business organizations and universities for research collaborations [5] and between business organizations for data sharing to create better applications [6].

The two main obstacles to this process of collaboration among organizations are arranging a large, common data storage area and providing secure access to the shared data. Organizations across the world invest a great deal of resources on minimizing storage costs and with the introduction of cloud-based services it is estimated that this cost would be reduced further [7]. Additionally, organizations spend a large amount of their yearly budget on security but this is still not sufficient to prevent security breaches [8], [9]. In this paper we present a web-based system (Hive Access Control[4]) that aims to achieve the previously stated goals by combining cloud computing techologies with policy-based security mechanisms. This idea comes in part from the recommendations of the Cloud Security Alliance for Identity and Access Management [10] and our previous work using XACML policies [11]. We have combined the Hadoop Distributed File System [12] with Hive [13] to provide a common storage area for participating organizations. Further, we have used a XACML [14] policy-based security mechanism to provide fine-grained access controls over the shared data. Users of our system are divided into groups based on the kinds of queries that they can run such as SELECT and INSERT. Our system provides a secure login feature to users based on a salted hash technique. When a user logs into our sytem, based on the group that the user belongs to he/she is provided with different options. We allow collaborating organizations to load data to the shared storage space in the form of relational tables and views. Users can also define fine-grained XACML access control policies on tables/views for groups of users. Users can then query the entire database based on the credentials that they have. We have provided some basic query rewriting rules in our system that abstract users from the query language of Hive (HiveQL). This allows them to enter regular SQL queries in the web application which are translated into HiveQL using the basic rewriting rules. Our system also allows new users to register but only a designated special user "admin" can assign these users to the appropriate groups. The contributions of this paper include:

- Mechanism to load and query shared data securely that is stored in HDFS using Hive.
- Additional layer of security above HDFS and Hive using a XACML policy-based mechanism.
- Basic query rewriting rules that abstract a user from HiveQL and allow him/her to enter SQL queries.
- Incorporation of the above mechanisms into a web-based system.

This paper is structured as follows: Section II presents the related work in the area of secure storage and retrieval of information in the cloud. In section III we present our architecture for solving the problem of secure large scale data sharing based on combining cloud computing technologies with XACML policy based security mechanisms. Further in section IV we present the details of our implementation by giving performance graphs and some sample screenshots. Finally section V presents our conclusions and future work.

[1] http://www.amazon.com/
[2] http://www.facebook.com/
[3] http://twitter.com/
[4] http://cs.utdallas.edu/secure-cloud-repository/

II. RELATED WORK

The aim of our paper is to combine cloud computing technologies with security mechanisms so that cooperating organizations can share vast amounts of data securely.

Since the birth of cloud computing technologies there has been much interest generated among researchers, business organizations and media outlets about security issues with these technologies [15], [16]. This interest has resulted in large-scale research and development efforts from business organizations [17], [18], [19]. A part of the work related to security in the cloud has been focused on implementing security at the infrastructure level. Reference [19] presents the vision for security design in Hadoop. This document presents a few security risks with Hadoop and outlines solutions to them. These solutions have been implemented in beta versions of Hadoop v0.20. This development effort is an important step towards securing cloud infrastructures but is only in its inception stage. The goal of our system is to add another layer of security above the security offered by Hadoop. Once the security offered by Hadoop becomes robust it will only strengthen the effectiveness of our system.

Amazon Web Services (AWS) is a web services infrastructure platform in the cloud [20]. Reference [18] offers an overview of security aspects that are relevant to AWS such as physical security, network security and AWS security. Our system is different from AWS in the sense that our cloud infrastructure is completely private *versus* AWS's infrastructure that is in the public domain. This distinguishing factor makes our infrastructure "trusted" over the AWS infrastructure where data must be stored in an encrypted format since AWS is in the public domain. In the future we plan to extend our work to include both, public and private clouds.

The Windows Azure platform is an Internet-scale cloud computing services platform hosted in Microsoft data centers [17]. Reference [21] provides an overview of the security challenges and recommended approaches to design and develop more secure applications for the Windows Azure platform. However according to reference [22], the Windows Azure platform is suitable for building new applications, but it is not optimal to migrate existing applications. The main reason that we did not use the Windows Azure platform is that we wanted to port our existing application to an open source system instead of writing our code from scratch as would be needed with Windows Azure. We also did not want to be tied to the Windows framework but rather allow our work to be used on any kind of system. We will be able to test our system on the Windows Azure platform once the platform supports the use of virtual machines (VM's) to run existing applications [22].

As we see from the discussions above, the research efforts identified in our study of related work have either just started or fit a completely different domain. We believe, to the best of our knowledge that our system is the first of its kind to introduce a layer of security that is independent of the security provided by current cloud infrastructures.

III. SYSTEM ARCHITECTURE

In this section we present our architecture that securely provides access to a large common storage space (the "Cloud") thus allowing cooperating organizations to share data reliably. We begin by giving an overview of the architecture followed by a discussion of each of its component layers.
Figure 1 shows the architecture of our system. Each rectangle in the figure represents a different component of our framework. The various line styles for arrows indicate the flow of control for a specific task that can be accomplished with this system. Next we present each of the component layers in the architecture.

A. The Web Application layer

The Web Application layer is the only interface provided by our system to the user to access the cloud infrastructure. We provide different functions based on the permissions assigned to a user. The web application provides a login page that can be used by any user to log into the system. We use the Java simplified encryption (JASYPT) library's [23] salted hash technique to store usernames and passwords in a file. Further, that file is stored in a secure location that is not accessible to any user. The system currently supports three types of users,

- users who can only query the existing tables/views,
- users who can create tables/views and define XACML policies on them in addition to querying all tables/views, and finally,
- a special "admin" user who in addition to the previous functions can also assign new users to either of the above categories.

B. The ZQL Parser layer

The ZQL Parser [24] layer takes as input any query submitted by a user and either proceeds to the XACML policy evaluator if the query is successfully parsed or returns an error message to the user. The ZQL Parser is a SQL parser written in Java that takes a SQL query as input and fills different Java Vectors with different parts of the query [24]. For example, given the query,

SELECT a.id, a.name FROM a WHERE a.id $>$ 5;,

the ZQL Parser parses the query and constructs different Java Vectors for every part of the query (SELECT, FROM and WHERE). In our system, the Vector of attribute names in the SELECT clause for the query above is returned to the web application layer to be used in displaying the results returned by the query. The Vector of table/view names in the FROM clause is passed to the XACML Policy Evaluator to ensure that the current user has permissions to access all tables/views specified in the query. If the evaluator determines that the current user has the required permissions, the query is processed further, else an error message is returned to the web application layer. The ZQL Parser currently supports the SQL DELETE, INSERT, SELECT and UPDATE statements.

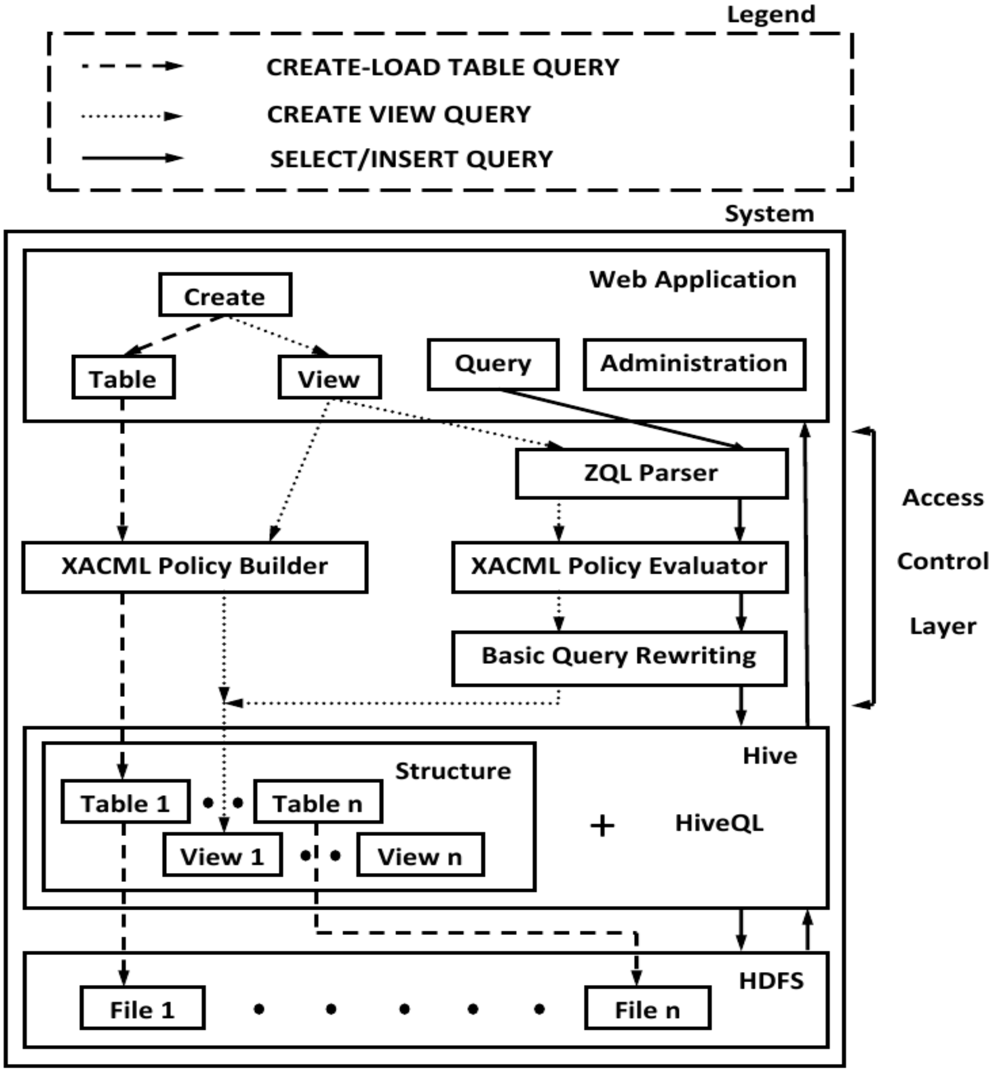


Fig. 1. System Architecture

Our future work involves adding support for other keywords such as CREATE, DROP etc.

C. The XACML Policy layer

The eXtensible Access Control Markup Language (XACML) is a XML-based language that is used to define access control policies on resources. The same language is also used to determine whether access is allowed for a particular resource based on the policy defined for that resource [14]. The following subsections explain how we have defined and used XACML policies in our framework.

1) XACML Policy Builder: In our framework the tables and views defined by users are treated as resources for building XACML policies. Further, we have defined role based access control [25], [26] policies on these resources based on the kinds of queries that are provided by our system. For every type of query supported by our framework we define a mapping between this type and all users that are allowed to run that kind of query. A sample listing of such a mapping is given below,

INSERT admin user1 user2
SELECT admin user1 user3

In our system, for every table/view that a user wants to create they are given the option of uploading their own pre-defined XACML policy or having the framework build a policy for them. If a user selects the latter option they must also specify the kinds of queries (e.g. INSERT, SELECT etc.) that will be allowed on the table/view. We then use Sun's XACML implementation [27] to build a policy for that table/view with the groups specified by that particular user.

2) XACML Policy Evaluator: Our system uses Sun's XACML implementation [27] to evaluate if the current user has access to all tables/views that are defined in any user query. If permission is granted for all tables/views then the query is processed further, else an error message is returned to the user. The policy evaluator is used both, during regular user query execution as well as during view creation since the only way to create a view in Hive is by specifying a SELECT query on the existing tables/views. The current user must have access to all tables/views specified in this SELECT query before the view can be created.

D. The Basic Query Rewriting layer

This layer enables us to add another layer of abstraction between the user and HiveQL by allowing users to enter SQL queries that are rewritten according to HiveQL's syntax. In our current system we provide two basic rewriting rules for user-specified SQL queries,

- HiveQL does not allow multiple tables in the FROM clause of a query, but rather expects this kind of query to be given as a sequence of JOIN statements. The user is abstracted from this fact by allowing him/her to enter a regular SQL query with multiple tables in the FROM clause that we transform to a sequence of JOIN statements in conformance with HiveQL's syntax. As an example,
  SELECT a.id, b.age FROM a, b;
  ⇒ SELECT a.id, b.age FROM a JOIN b;
- HiveQL uses a modified version of SQL's INSERT-SELECT statement, INSERT OVERWRITE TABLE <tablename> SELECT rather than INSERT INTO <tablename> SELECT. Again we abstract this from the user by allowing him/her to enter the traditional INSERT INTO <tablename> SELECT which we then rewrite into HiveQL's INSERT OVERWRITE TABLE <tablename> SELECT. As an example,
  INSERT INTO a SELECT * FROM b;
  ⇒ INSERT OVERWRITE TABLE a SELECT * FROM b;

As part of our future work we plan to extend these basic rewriting rules with more complicated rules in a complete query rewriting engine.

E. The Hive layer

Hive is a data warehouse infrastructure built on top of Hadoop [13]. Hive provides the ability to structure the data in the underlying HDFS as well as to query this data. The arrows in Figure 1 between the tables in this layer and the files in the HDFS layer indicate that each table in Hive is stored as a file in the HDFS. These files contain the data that this table represents. There are no arrows between the views in this layer and the files in the HDFS layer since a view is only a logical concept in Hive that is created with a SELECT query. In our framework, Hive is used to structure the data that will be shared by collaborating organizations. Further we use Hive's SQL-like query language, HiveQL, to enable access to this data. The advantage of using Hive in our system is that users can query the data using a familiar SQL-like syntax.

F. The Hadoop Distributed File System (HDFS) layer

The Hadoop Distributed File System (HDFS) is a distributed file system that is designed to run on basic hardware [12]. The HDFS layer in our framework stores the data files corresponding to tables that are created in Hive [28]. Our security assumption is that these files can neither be accessed using Hadoop's [29] web interface nor Hadoop's command line interface but only by using our system.

IV. IMPLEMENTATION DETAILS AND RESULTS

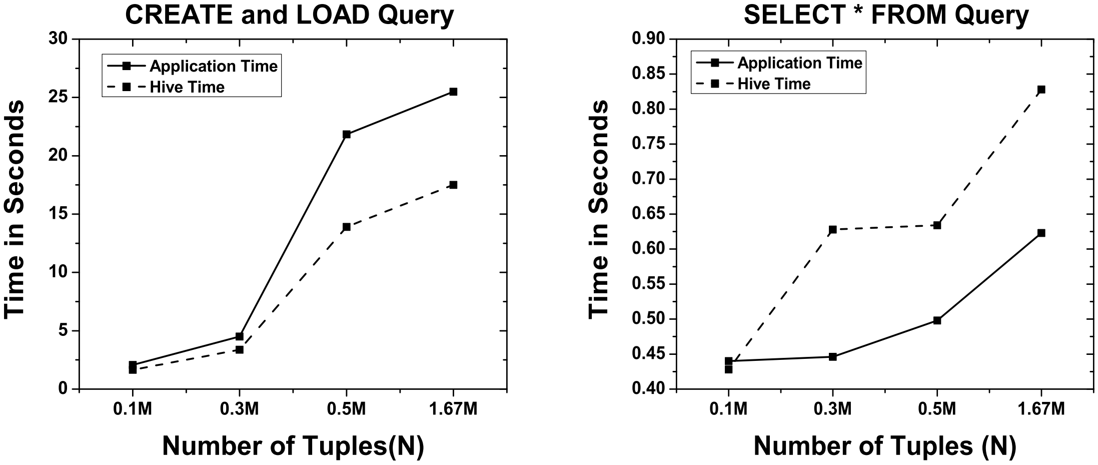
In this section we present the implementation details of our system by providing performance graphs for the insert and query processes for tables with different sizes. We begin by giving a brief description of our implementation setup followed by the implementation details.

A. Implementation Setup

Our implementation was done on a 19 node cluster with a mix of two different configurations for nodes. Further, all nodes are in the same rack. Out of the 19 nodes 11 nodes ran Ubuntu v10.04 Lucid Lynx, on an Intel Pentium 4, 3.2GHz CPU with 4GB SDRAM 400 MHz memory and a 40GB Western Digital WDC WD400BB-75FJ SATA hard drive as the primary drive and a 250GB Western Digital WD2500AAJB-0 SATA hard drive as the secondary drive. The other 8 nodes ran Ubuntu v9.04 Jaunty Jackalope, on an Intel Pentium 4, 2.8 GHz CPU with 4GB SDRAM 333MHz memory and two 40GB Western Digital WDC WD400BB-75FJ SATA hard drives. We used the Java version JRE v1.6.0_18 for our implementation. For the cloud infrastructure we used Hadoop version v0.19.1, with a 1000MB heap space and Hive version v0.5, with the default heap space. We also used Apache Tomcat v7.0.0 as the web server for our application with a 2GB heap space. We also used default values for all parameters that are provided by Hadoop and Hive. We understand that we will have a performance gain when we set optimal values for these parameters. But, since this is preliminary work we have chosen not to focus on these parameters which will be done in the future.

B. Experimental Datasets

We have used two different datasets to test the performance of our system *versus* Hive. The first dataset is the Freebase



(a) Data Loading Time

(b) "SELECT * FROM" Query Time

Fig. 2. Experimental comparison between our application and Hive for the Freebase dataset. The number of tuples, N = ⌊actual table size⌋

[30] system which is an open repository of structured data that has approximately 12 million topics or entities. An entity is a person, place or thing with a unique identifier. We wanted to simulate an environment of cooperating organizations by using the people, business, film, sports, organization and awards datasets from the Freebase system. We assume that each dataset is loaded into our system by a different organization and further, users can run various queries across these datasets based on their permissions. The queries we have used to test our implementation were created by us based on the datasets of the Freebase system.

The second dataset we have used to test our system is the well known TPC-H benchmark [31]. The TPC-H benchmark is a decision support benchmark that consists of a schema that is typical to any business organization. The benchmark contains 8 tables and provides 22 queries with a high degree of complexity. We have used this benchmark to test the performance of our system *versus* Hive in performing complex queries. The TPC-H benchmark provides a tool for data generation (DBGEN) and a tool for query generation (QGEN). We have used DBGEN to generate datasets with varying Scale Factor's (SF) from 1 to 1000 as specified in the benchmark document. The reader should note that a scale factor of 1 is approximately 1GB of data. Thus we have tested our system with data sizes varying from 1GB to 1000GB. The smaller of these datasets (SF=1, SF=10, and SF=30) are used to test the loading performance of our system *versus* Hive. On the other hand, the larger datasets (SF=100, SF=300, and SF=1000) are used to run a few of the benchmark queries. We have used queries Q1, Q3, Q6 and Q13 of the TPC-H benchmark to test our system. These queries were randomly selected after applying the following criterion to the study given in [32]. The original query does not need to be divided into subqueries manually since our web application does not support this feature. We also think that the results obtained by running the queries selected above is indicative of the performance of all the other TPC-H benchmark queries that can be run on our system.

C. Implementation Results

We have tested our web-based system for performance metrics such as data loading and querying times. Further, we have compared these metrics with the Hive command line interface (CLI). All query times that are used in performance graphs and result tables in this subsection are averaged over three separate runs. We ran two sets of experiments, one using the Freebase system and the other using the TPC-H benchmark.

Figure 2 shows a comparison of the data loading and querying time of our application *versus* Hive for the Freebase datasets. As we see from figure 2(a) the data loading time for our application is almost the same as Hive's time for small tables (0.1 and 0.3 million tuples). As the number of tuples increases to 0.5 million and then to 1.67 million tuples our system gets slower than Hive at loading tuples. This is primarily because of the overhead associated with establishing a connection with the Hive database as well as the time associated with building a XACML policy for the table being loaded.

Figure 2(b) shows a comparison of the running time for a simple "SELECT * FROM" query between our application and Hive. We have run the query on the same tables that were used for data loading in figure 2(a) but we restricted our results by using the LIMIT clause to only the first 100 tuples. This was done to avoid the large time difference that would occur between our application and Hive's CLI since we have implemented a paging mechanism on the results whereas Hive's CLI would display all results on the screen. We see that

TABLE I
COMPARISON OF RUNNING TIMES FOR VARIOUS QUERIES ON OUR SYSTEM *versus* HIVE FOR THE FREEBASE SYSTEM

Query Type	Query	System Time (sec)	Hive Time (sec)
Query Table	SELECT name, id FROM Person LIMIT 100;	27.1	28.4

Query Table	SELECT id FROM Person;
	WHERE name = ’Frank Mann’ LIMIT 100;
		30.2	30.5

Create View	CREATE VIEW Person_View AS
	SELECT name, id FROM Person;
		0.19	0.114

Query View	SELECT name, id FROM Person_View LIMIT 100;	22.5	23.2

Query View	SELECT id FROM Person_View
	WHERE name = ’Sachin’ LIMIT 100;
		30.1	29.3

Join Query	SELECT p.name, d.place_of_death
	FROM Person p JOIN Deceased_people d
	ON (p.id = d.id AND p.name = d.name)
	LIMIT 100;
		70	68.6

Aggregate Query	SELECT p.name, COUNT(DISTINCT p.id)
	FROM Person p GROUP BY p.name LIMIT 100;
		64	64.3

Insert Query	INSERT INTO Person_Insert
	SELECT name,id,profession FROM Person;
		29.2	27.77

our application times are slightly faster than the running time for the query on the Hive CLI. This difference is because of the time taken by the Hive CLI to display the results of the query on the screen. Both running times are fast because Hive does not need to run a Map-Reduce [33] job for this query but simply needs to return the whole file for the corresponding table from the HDFS.

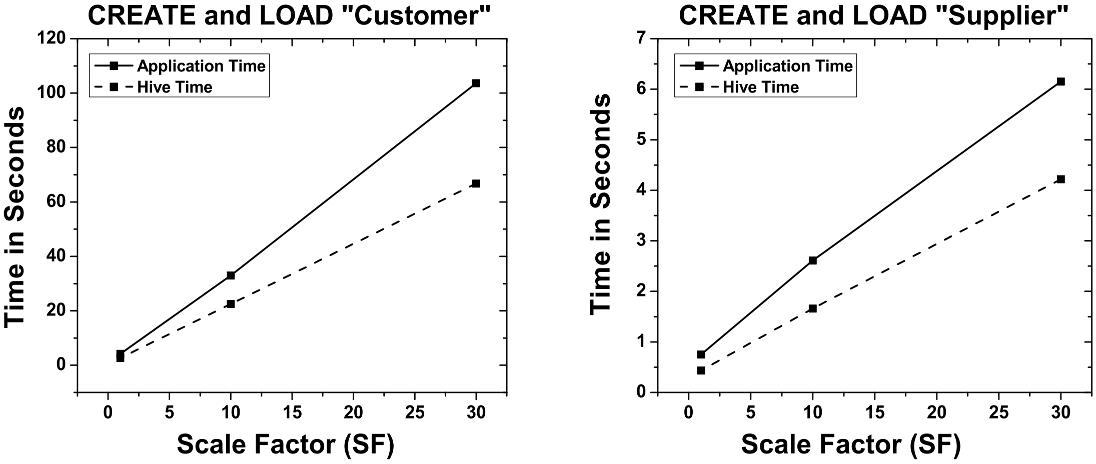
We have also run a number of other queries and compared the running times of these queries on our system *versus* Hive for the Freebase system. These queries test the performance of our system in creating and querying views/tables *versus* Hive. We have tested a variety of queries including insert, create, select, aggregate and join queries. We present each query and the average running time on our system and on Hive in table I.

Figure 3 shows a comparison of the data loading time of our application *versus* Hive for the “Customer” and “Supplier” tables for SF = 1, 10 and 30 from the TPC-H benchmark. Our system currently allows users to upload data files that are at most 1GB in size. The TPC-H benchmark’s DBGEN tool generates files for the “Customer” and “Supplier” tables for SF = 1, 10, and 30 that are less than 1GB in size. These are the reasons why we have selected the “Customer” and “Supplier” tables with SF = 1, 10, and 30 to compare the data loading performance of our system *versus* Hive. Figure 3 shows us results that are similar to the results obtained from figure 2(a). Our system performs similar to Hive at the smallest SF of 1 and as the SF increases our system gets slower than Hive for data loading. Again, this difference in execution performance is because of the overhead associated with the Hive database connection and XACML policy generation. The trend for both, our system and Hive is linear as expected, since the size of these tables increases linearly with the SF.

Table II shows a comparison of the performance of four TPC-H benchmark queries on our system *versus* Hive. We see that our system performs as well as the Hive command line interface for the selected queries. Based on the query performance times for both, our system and Hive, we see that as the size of the tables increases the time for benchmark query execution also increases, as expected. In a production environment such queries would not be performed at runtime on large datasets. We would rather run these queries offline and could store the results in Hive as views. We could then use a query rewriting mechanism to return these results efficiently.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a system that allows cooperating organizations to securely share large amounts of data. We have ensured that the organizations have a large common storage area by using Hadoop. Further, we have used Hive to present users of our system with a structured view of the data and to also enable them to query the data with a SQL-like language. We have used a simple salted hash mechanism



(a) “Customer” Table Loading Time

(b) “Supplier” Table Loading Time

Fig. 3. Comparison of Loading and Querying times between our application and Hive for the TPC-H benchmark.

TABLE II
COMPARISON OF RUNNING TIMES FOR TPC-H BENCHMARK QUERIES ON OUR SYSTEM *versus* HIVE

Query	Scale Factor (SF)	System Time (sec)	Hive Time (sec)
Q6	100	605.24	590.663
	300	1815.454	1806.4
	1000	6240.33	6249.679
Q13	100	870.704	847.52
	300	1936.351	1910.186
	1000	7322.541	7304.392
Q1	100	1210.04	1209.786
	300	5407.14	5411.616
	1000	42780.67	42768.83
Q3	100	1675.19	1670.765
	300	7532.23	7511.523
	1000	61411.21	61390.71

to authenticate users in the current version of our system. We plan to implement a more sophisticated technique for authentication in future versions of our system. In this paper we have used the ZQL parser to parse any SQL query that is input by the user. We plan to extend this parser with support for keywords such as DESCRIBE, JOIN etc that are currently not supported in ZQL. We have abstracted the user from the use of Hive by implementing some basic query rewriting rules. A part of our future work is to implement materialized views in Hive and extend the basic query rewriting rules into a complete engine for Hive that takes into account all existing tables/materialized views and the XACML policies defined on them. We have provided fine-grained access control on the shared data using XACML policies. We have also incorporated role based access control in our framework based on the kinds of queries that users will submit to our system. In the current version of our system we only provide support for two types of keywords, INSERT and SELECT, as groups for XACML policies. In the future we plan to extend our system to include other keyword based groups such as DELETE, UPDATE etc. We also plan to test the impact of using different values for parameters provided by Hadoop and Hive on query execution performance. Lastly, the current system is implemented in a private cloud which will be extended to include public clouds such as Amazon Web Services and Amazon Simple Storage Services in future versions.

Acknowledgements

The work carried out in this paper was funded by the Air Force Office of Scientific Research under contract FA9550-08-1-0260 on secure cloud computing. We thank Dr. Robert Herklotz for his support and encouragement of our work. We are also thankful to Mehul Vyas and Nikhil Mishra for helping with the design and implementation of our framework. We would also like to thank Arindam Khaled for providing us with the source code for the XACML policy builder and evaluator.

References

[1] World Wide Web. http://en.wikipedia.org/wiki/World_Wide_Web.

[2] Samuel Axon. Facebook Will Celebrate 500 Million Users Next Week. http://news.yahoo.com/s/mashable/20100717/tc_mashable/facebook_will_celebrate_500_million_users_next_week, July 2010.

[3] Andreas Weigend. The Social Data Revolution(s). http://blogs.hbr.org/now-new-next/2009/05/the-social-data-revolution.html, May 2009.

[4] Ganesh Variar. Data Explosion: A guide to connecting the dots. http://advice.cio.com/ganesh_variar/data_explosion_a_guide_to_connecting_the_dots?source=rss_Blogs_and_Discussion_All, January 2010.

[5] Nokia Research Center - Open Innovation. http://research.nokia.com/openinnovation.

[6] Salesforce.com and Facebook Create New Opportunities for Enterprise Applications to Serve Facebook's 120 Million Users. http://www.facebook.com/press/releases.php?p=63948, November 2008.

[7] Ann All. Cloud Data Storage: It'll Still Cost You, So Give It Some Thought. http://www.itbusinessedge.com/cm/blogs/all/cloud-data-storage-itll-still-cost-you-so-give-it-some-thought/?cs=38733, January 2010.

[8] John Sawyer. Tech Insight: How To Cut Security Costs Without A Lot Of Pain. http://www.darkreading.com/smb-security/security/management/showArticle.jhtml?articleID=226200159, July 2010.

[9] First Annual Cost of Cyber Crime Security. Technical report, Ponemon Institute, July 2010.

[10] Subra Kumaraswamy, Sitaraman Lakshminarayanan, Michael Reiter Joseph Stein, and Yvonne Wilson. Domain 12: Guidance for Identity & Access Management V2.1, April 2010.

[11] Pranav Parikh. Secured Information Integration with a Semantic Web-based Framework. Master's thesis, The University of Texas at Dallas, December 2009.

[12] Dhruba Borthakur. HDFS Architecture. http://hadoop.apache.org/common/docs/current/hdfs_design.html, 2010.

[13] Apache Hive. http://wiki.apache.org/hadoop/Hive.

[14] Tim Moses. eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, February 2005.

[15] David Talbot. How Secure Is Cloud Computing? http://www.technologyreview.com/computing/23951/, November 2009.

[16] Robert L. Mitchell. Cloud storage triggers security worries. http://www.computerworld.com/s/article/340438/Confidence_in_the_Cloud?, July 2009.

[17] Windows Azure Platform - Whitepapers. http://www.microsoft.com/windowsazure/whitepapers/.

[18] Amazon Web Services: Overview of Security Processes. http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf, November 2009.

[19] Owen O'Malley, Kan Zhang, Sanjay Radia, Ram Marti, and Christopher Harrell. Hadoop Security Design. http://bit.ly/75011o, October 2009.

[20] Amazon Web Services. http://aws.amazon.com/.

[21] Andrew Marshall, Michael Howard, Grant Bugher, and Brian Harden. Security Best Practices For Developing Windows Azure Applications. http://download.microsoft.com/download/7/3/E/73E4EE93-559F-4D0F-A6FC-7FEC5F1542D1/SecurityBestPracticesWindowsAzureApps.docx.

[22] Jon Brodkin. Microsoft Windows Azure and Amazon EC2 on collision course. http://www.networkworld.com/news/2010/062510-microsoft-azure-amazon-ec2.html, June 2010.

[23] JASYPT - Java simplified encryption. http://www.jasypt.org/index.html.

[24] Zql: a Java SQL parser. http://www.gibello.com/code/zql/.

[25] David F. Ferraiolo and D. Richard Kuhn. Role-Based Access Controls. In *National Computer Security Conference*, pages 554–563, 1992.

[26] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[27] Sun XACML Implemetation. http://sunxacml.sourceforge.net/.

[28] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - A Warehousing Solution Over a Map-Reduce Framework. *PVLDB*, 2(2):1626–1629, 2009.

[29] Apache Hadoop. http://hadoop.apache.org/.

[30] Freebase. http://www.freebase.com/.

[31] TPC BENCHMARK H. http://tpc.org/tpch/spec/tpch2.11.0.pdf.

[32] Running the TPC-H Benchmark on Hive. https://issues.apache.org/jira/secure/attachment/12416257/TPC-H_on_Hive_2009-08-11.pdf.

[33] Apache MapReduce. http://hadoop.apache.org/mapreduce/.