Query Efficient Database Design for SAAS Delivery Model of Cloud

[1]Neha Kaushik, [2]Sanjeev Kumar Singh
[1]M.Tech Scholar, [2]Professor
[1]Department of Computer Science and Engineering
[1]VIET Dadri, G.B. Nagar, India

Abstract: Data management and sharing is the challenge being faced by all the IT majors today. Adds over it, is the challenge faced by the cloud service providers in terms of multi-tenancy of data and its efficient retrieval. It becomes more complex in a heterogeneous computing environment to provide cloud services. A simple, robust, query efficient, scalable and space saving multi-tenant database architecture is proposed for SaaS model of cloud architecture where organizations can collaborate to create a cloud, that doesn't harm their existence or profitability. In the proposed system we use XML field attribute to store more attribute in the same field in the form of XML file to save space and yet perform transactional operation with better efficiency. The results of the proposed multi-tenant database should show improvement for insertion, deletion and updation-queries. We further improve the select query response by caching passed record. The response of the proposed approach should be stable as compared to other system for increased number of attributes up to 50. The proposed approach should also be space efficient. Dynamically changing cloud configurations requires adaptable database and mechanism to persist and manage data and exploit resources in a multitenant design.

Index Terms **- Cloud, Multitenant Database.**

I. INTRODUCTION

Cloud computing is a computing paradigm where services and data reside in common space in scalable data centers, and these services and data are accessible via authentication. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

Some of the definitions of cloud computing given by distinguished persons in the field of cloud computing:

A style of computing in which massively scalable IT–Related capabilities are provided "as a service" using Internet technologies to multiple external customers".

1.1 Multi-Tenant Database in Cloud Computing

Cloud computing is opening markets not previously contemplated by businesses. Some software companies are now thinking of delivering their software as a service instead of the usual method of developing the software and selling it to their clients using typical distribution methods. To become software as a service (SaaS) vendor, companies need to find the right balance where resources are shared among the various tenants to reduce costs, while ensuring that customer information is kept private from other customers. In a multi-tenant environment, a SaaS company can reduce costs if it shares or reuses more of its resources. However, the more the company shares resources, the more risks it faces because an outage of a shared resource can potentially affect many customers. The more resources shared also adds to the complexity of the solution.

1.2 Motivation & Goal

Multi-Tenancy allows use a single instance of application that satisfies the requests of multiple clients. It reduces the efforts made in development, as well as cost of development. Each individual Educational organization is considered as a tenant and all such organizations collaborate to create and participate in database building process. Multi-tenancy in Education-cloud, where a cloud computing framework is harnessed to manage Information system of an Educational institution would be highly benefit in terms of space, manageability, scalability and availability. We propose to use our multi-tenant database for such a scenario, where tenants (Educational Institutions) collaborate to build the database and use it by authorization.

Various concerns come into picture when third party offers multi-tenant service in a cloud. These are:

1. How to implement Multi-Tenant data architecture so that the all tenant's requirements are met.
2. How to provide isolation among data of different tenants.

Various approaches for multi-tenancy have been proposed depending on the degree of isolation. Broadly three approaches are used to implement multi-tenant data architecture.

Separate database: In this approach, a separate database is used to store the data of an individual tenant (Educational Institution). Data, related to a tenant, is kept isolated from the data of other tenant. Each database contains some metadata that is used to relate the correct database to the specific tenant. This approach is relatively costly as it requires higher cost of maintaining the database.

Shared database, separate schema: This approach requires multiple tenants to be accommodated into a single database, though the set of tables and schema for each tenant is different. These set of tables for the tenant together make the schema for that tenant. This approach is relatively simple to implement.

Shared database, shared schema: This approach involves same database and schema to be shared by all tenants. Set of tables is shared by all the tenants. The hardware and backup costs for this approach are lowest because it supports large number of tenants per database server. The main aim is to reduce the amount of storage needed for database.

The major goal of this work is to implement Multi-Tenant database for a cloud such that storage requirements for its data-center are reduced and to provide a software architecture that can query the database in an efficient way. The sub goals of the paper are:

1. To provide an architecture that supports multi-Tenancy in shared database shared schema fashion.
2. To implement a software interface that allows tenants to choose their tables independent of real database implementations.
3. To introduce isolation among data of different tenants.

II. LITERATURE SURVEY

In recent times, the use of multi-Tenant database systems increased multifold [1]. Multi-Tenant database system uses the SaaS model in which a data center is hosted by a service provider and the tenants subscribe to the services provided by the service provider [2]. Some of the applications, that uses multi-Tenant database, are Customer Relationship Management (CRM), Supplier Relationship Management (SRM), Business intelligence (BI) and a scenario where multiple educational institutions are sharing the same database-space (academic and administrative resources) to collaborate and benefit from each other. Various approaches have been discussed and implemented by different researchers. Some of these are as follows:

Private Table

The most basic way to support extensibility is to give each tenant their own private tables which can be extended and changed. Aulbach et al. [3], [20] state that Private Tables' technique allows each tenant to have his own private tables. Simply by renaming tables, we can transform the query from one tenant to another, and we don't need to use extra columns like "tenantid" to distinguish and isolate tenants, each tenant has different business requirements, so there exist three different user tables. In contrast, many tables are required to satisfy each tenant needs, therefore this technique can be used if there are few tenants using it, to produce sufficient database load and good performance.

Pivot Tables

In this approach, a pivot table is created for a single column [7]. This table is shared by various tenant's tables. Each pivot table consists of a Tenant column, Table column, a "col" column and a "row" column. Tenant column refers to the particular tenant. Table refers to the particular table for that tenant.

In a Pivot Table, each row field in a logical source table is given its own row. There are four columns in the Pivot Table including: tenant, table, column, and row that specify which row in the logical source table they represent. In addition, the single data type column that stores the values of the logical source table rows according to their data types in the designated pivot Table. The data column can be given a flexible type, such as VARCHAR, into which other types are converted, in which case the Pivot Table becomes a Universal Table for the Decomposed Storage Model.

Chunk Folding

A Chunk Table is similar with a Pivot Table except that it has a set of data columns of various types, with and without indexes, and the Col column is replaced by a Chunk column. This technique partitioned logical source table into groups of columns, each group assigned to a chunk ID and mapped into an appropriate Chunk Table. The Chunk Table. The "row" column is used to map a Chunk folding is a technique discussed in [5]. It vertically divides the logical tables into chunks and those are folded together into various physical tenants and are joined as needed. One table is used to store the base account information and other table is used to hold the extensions. This approach works by containing the heavily used parts of the schema into base tables and the rest part is mapped into the extensions.

Multi-Tenant shared table

In this approach, common contents from tenant information are separated [7]. This technique introduces the concept of tenants at database layer so that database engine can select an appropriate area for storage of data for that tenant. Only one schema is used for an application.

An approach that deals with scalability issue is discussed in [2]. This approach handles the increasing number of tenants without performance degradation in query performance. Two main problems are resolved in this paper. One is to resolve the sparseness of the universal table approach and second is to provide an indexing scheme for multi-Tenants. Three different approaches shared machine, shared process and shared table are discussed by Jacobs in [8]. In [9], a simulation study is done which analyzes the performance of different approaches to implement the multitenant databases. An approach for multitenant architecture supporting the SaaS model is discussed in [10]. The authors have proposed a cloudio software platform that is concerned with the flexibility of data model and managing the large data sets in the database. Different challenges in Multi tenant applications are discussed in [11] such as scalability, security, performance and zero downtime. Addressing the Application level security requirements of Multitenant applications in [12] scenarios for authentication and authorization have been discussed. A Number of conditions determine grant of resources to a particular user. Authorization is at the core of this process. Policies must be clearly specified by service provider keeping in mind that these policies are enforced by the authorization mechanisms [3]. Sometimes users that are authenticated can have access to the unauthorized resource. Kerberos [9, 12] is a widely used and fool proof authentication protocol based on symmetric key cryptography. It provides high speed of communication, a high degree of mutual authentication and transferrable degree of trust [4,14].

Improved Extension Table

In the universal table model it is a big challenge to decide the number of custom fields (columns in table). Providing less number of columns might restrict the tenants who wish to use a large number of custom fields. A large number of such fields may result in large number of NULL values in the database table [13].

Second problem is of data types of these columns because if one tenant wants to use a field as a string whereas other wish to use it as a Boolean, then resolving this issue becomes major concern [13]. This problem can be resolved by keeping the data of all tenants in a separate table and their related Meta Data in another table.

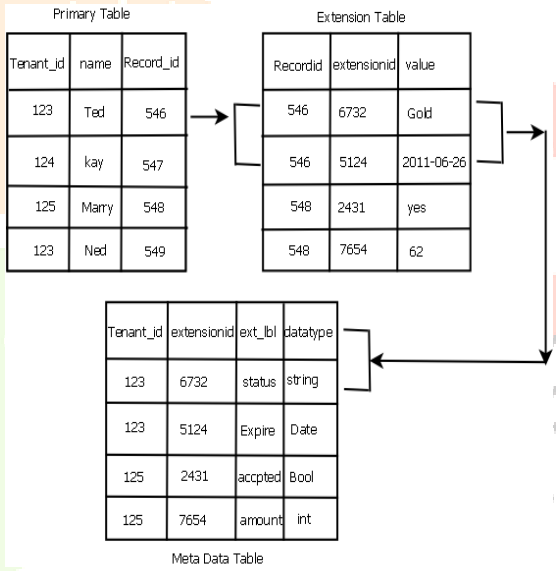


Figure 1. Use of extension table

Fig.1 shows three tables used in the basic approach, that makes use of extension table. The primary table keeps the tenant_id and record_id and some other fields. record_id field uniquely identifies the transaction made by a particular tenant. By extracting the value of record_id field, one can extract the values from the extension table. For a single record_id, there are number of rows in the extension table.

The number of rows for a particular record_id is the number of columns in the logical table of that tenant. The Meta Data table tells about the data types of these fields. Whenever a tenant inserts data into its table, Meta Data table is accessed to match the given values against the data types of the Meta data table. An ext_id in extension table is associated with an extensionid field of Meta Data table. This extensionid is unique for each column and is used to know the data type and external label of that field in the logical table for that tenant.

An extension table contains the actual data for all the tenants. In case of universal table structure, columns, which are not used by a particular tenant, contain the NULL value, this results in wastage of space. Extension table concept overcomes this problem. Figure shows an extension table and a Meta Data table.

In the basic approach of extension table discussed above, following drawbacks can be observed:

1. Extension table contains a lot of information for Meta Data i.e. for a single row of table of a tenant that consists of four columns, The record_id and extensionid are repeated four times this information introduces a kind of redundancy.
2. Whenever a query for insertion, deletion or update is performed three tables are accessed which increases the query processing time.

III. PROPOSED SYSTEM

The Proposed approach is designed over the shared database shared schema technique discussed in Introduction section. A single database and a table is shared amongst all the tenants. Our proposed approach makes use of extension table. An extension table contains the data for all the tenants (Institutions). To differentiate one tenant's data from other, some mechanism is used.

In our proposed approach following concepts are introduced and Implemented:

1. Concept of XML object into a database is used that helps to reduce the size of extension table as well as eliminates the need of a primary table as shown in figure below.
2. An approach that achieves multiple table creation for a tenant is proposed and successfully implemented.
3. Kerberos authentication protocol is used for authentication and authorization.

Fig 3.1. shows the proposed approach where extension table consists of a tenant_id, a record_id , an XML attribute and a table_id . Tenant_id and record_id uniquely identify a particular record. A record_id is used to associate each transaction with a unique record number. XML object contains the data for an entire row of a Tenant's logical table. Tags in a single XML object refers to the name of a particular field in the corresponding table. Table_id field represents the id of the table in which a particular record is inserted for the specified Tenant. This XML document is dynamically created and depends upon the table structure of a particular Tenant's logical table.

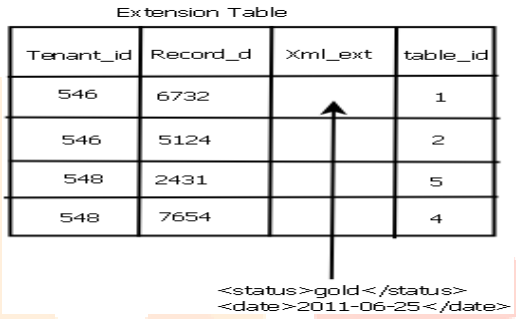


Figure 2. Modified Extension Table

A tenant is given full liberty of creating any number of tables and any number of columns in that table. Tenant specifies the name of the table and our proposed system generates unique id for that table for that tenant. A table that maintains the information about all the tables of all tenants is created. This table maps the table_id field of the extension table to the name of the table which a tenant is referring to.

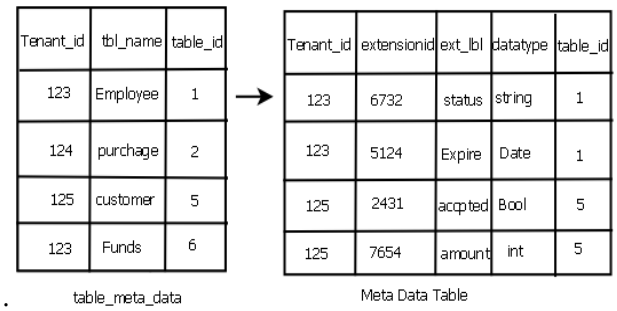
Creating the Tables

A tenant is free to use any number of custom fields assuming that service provider has created sufficient number of fields in the main database schema. A tenant is free to create any number of tables and use any data type (supported by that DBMS) for its fields. Whenever a tenant specifies a new table name, this name is stored in the table_meta_data table. This table associates a unique id to that name for that particular tenant. Tenant_id field in this table is used to extract the table names that are owned by that tenant. Figure 3 shows the structure of table_meta_data table.

Tenant_id	tbl_name	table_id
123	Employee	1
124	purchage	2
125	customer	5
123	Funds	6

Figure 3. Table table_meta_data

Each table name is given a unique table id that is used to keep track of information inserted in that table. Meta data information is also maintained in Meta_data_table table. This table contains the information like data type and external label for a column that a tenant is using. Each column is associated with a ext_id. Figure 4 shows these two tables.



table_meta_data

Tenant_id	tbl_name	table_id
123	Employee	1
124	purchage	2
125	customer	5
123	Funds	6

Meta Data Table

Tenant_id	extensionid	ext_lbl	datatype	table_id
123	6732	status	string	1
123	5124	Expire	Date	1
125	2431	accpted	Bool	5
125	7654	amount	int	5

Figure 4. Tables involved in Creation of new table.

IV. IMPLEMENTATION AND PERFORMANCE EVALUATION

To implement the proposed approach, Python with MySQL with native XML parsing support for the XML attribute is used. The table tbl_usr contains the information about the tenants. When a new tenant registers with the service provider, the registration procedure is called to check if this is a new user by traversing the list of its existing users and its IP address. If this is an existing user it simply discards the user request otherwise it provides the registration agreement to the user and demands for user name and password. It also assigns a Tenant id after registration process. All this information is kept in tbl_usr table. This tenant id is used for identifying a particular tenant and to identify the particular tables that are owned by Tenant.

Comparison with extension table approach:

The number of rows in the original extension table depends upon the number of fields in a tenants table. But in our approach it contains only a single entry for a row. Therefore a lot of space savings and also solves the NULL value problem with the extension table approach. In figure 8 it can be seen that except for select query all other query outperform the extension table approach. As shown in figure 5 Statistically for insertion, deletion and updation query there is a gain in number of query execution per second of 20%, 230% and 236% respectively and there is drop by 27% for selection query, which is slow due to parsing of XML file. The gain is due to less number of joins involved in the schema.

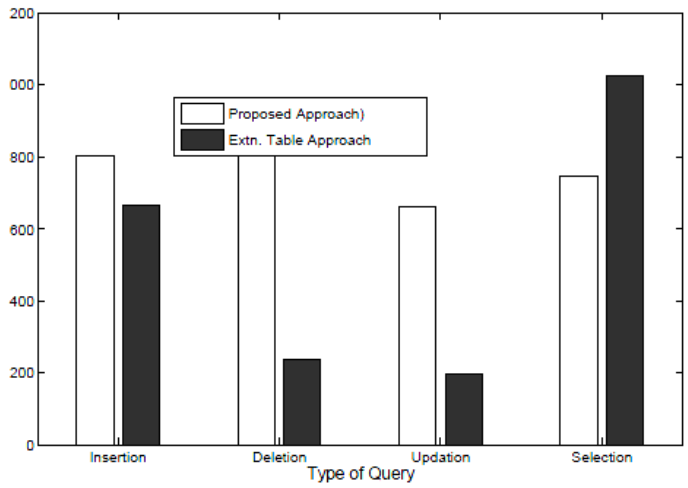


Figure 5. Showing no of queries executed per sec.

The average response time for query execution for added attributes is almost constant for our approach as shown in Figure 6 but for extension table approach it increases exponentially. Figure 4.2 shows there is exponential rise in the response time with increase in number of attributes in the extension table approach where as our approach yields a constant response time approximately. The response time in the extension table approach degrades varying from 2.45% to 384.5% in five consecutive increases in step size of 10 attributes. The increase in attributes involves creation of new tables therefore more number of joins are required to satisfy a query. Therefore in the extension table approach the response time increases with added attributes. Since in the proposed approach due to the use of XML attribute the number of tables created would be less therefore resulting in lesser number of joins required for query execution. The better performance in response time in our approach is due to the use of XML Field, which accommodates and adjusts extra added attributes in the XML field itself to an extent.

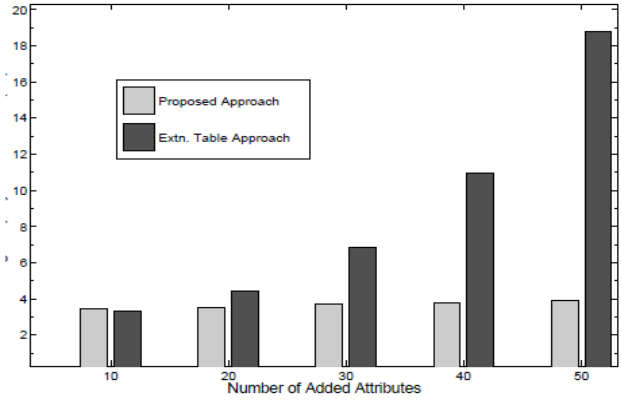


Figure 6: Average performance for added attributes

Figure 7 show the increase in space requirements in the extension table approach, with the increase in no of attributes, whereas in our approach the space requirement increases linearly. This is again due to the adjustment of group of attributes into one Field in for of XML file.

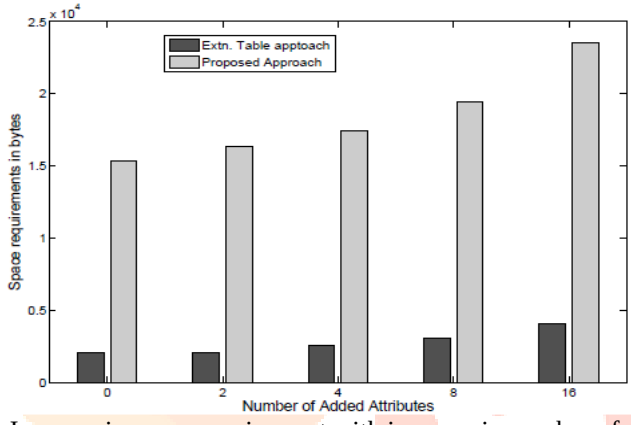


Figure 7: Increase in space requirement with increase in number of attributes.

We consider in the implementation model that a total of 10 common attributes are present in the multi-tenant database along with this 20 tenant specific attributes are there. The size of a field is 512 bytes. Maximum allowed attributes in an XML file is 4. The efficiency gained in space on an average is 86%. It could be greater if more than 4 attributes could be allowed in a field. In our approach we are only limited by the size of XML file that can fit in field as constraint by the native database technology used.

Selection with caching assuming 20% hit ratio yield a better performance as shown in the graph below.

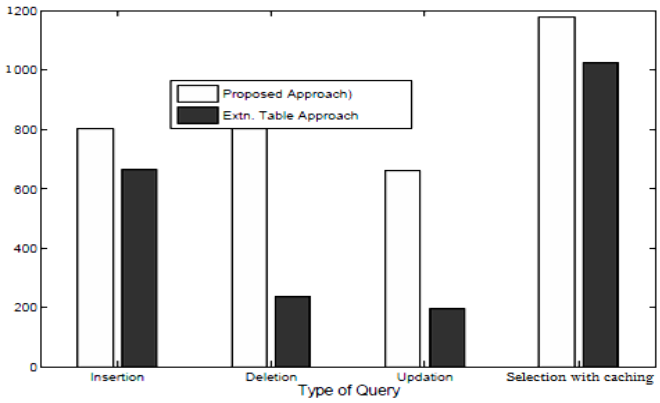


Figure 8: No of queries executed per second vs type of query

V. CONCLUSION

In this work attempt has been made to implement the Multi-tenant database for SaaS model of a cloud that offers operational advantages over the existing ones. It fits very well in scenarios where SaaS cloud services are to be delivered between multiple clients(organizations). The proposed multi-tenant database accommodates larger number of tenants because a single database instance is used to store the data of multiple tenants. Another advantage of the proposed work is that the tenants are allowed to create multiple tables which add flexibility in terms of having varied set of attributes as specifically required for its application. It is evident from the result that our approach performs much better in terms of space saving in terms of solving the NULL value problem as compared to other multi-tenant approaches. With increase the number of attribute in the table the query performance

drops with the extension table approach as compared to our approach, which is due to more number of attribute and more number of joins required to execute query. The multi-tenant database architecture proposed is highly efficient in terms of query execution, space saving and change in number of attributes. The performance is moderate and comparable with extension table approach for concurrent requests. The results of the proposed work show 20% to 230% improvement for insertion, deletion and updation-queries. The response of the proposed approach is stable as compared to other system which degrades in terms of response time by 384% for increased number of attributes up to 50. The proposed approach is also space efficient by average of 86% for 2 to 16 more added attributes. Also by using caching in case of selection query the performance is found to be improved.

VI. REFERENCES

[1] H. Hacigumus, B. Iyer, S. Mehrotra, "Providing database as a service", In 18th International Conference on Data Engineering, pp. 29-38, 2002.

[2] Mei Hui, Dawei Jiang, Guoliang Li, Yuan Zhou, "Supporting Database Applications as a service", In IEEE 25th International Conference on Data Engineering, pp. 832-843, 2009.

[3] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, Jan Rittinger, "Multi-Tenant Databases for Software as a Service:Schema-Mapping Techniques", In the proc. of International Conference on Management of Data - SIGMOD, pp. 1195-1206, 2008.

[4] D. Maier and J. D. Ullman, "Maximal objects and the semantics of universal relation databases", In ACM Trans. Database Syst., pp.114, 1983.

[5] G. P. Copeland , S. N. Khoshafian. "A decomposition storage model", In the Proc. of 2005 ACM SIGMOD International conference on Management of data, pages 268279. ACM, 2005.

[6] R. Agrawal, A. Somani, and Y. Xu, "Storage and querying of e-commerce data", In the Proc. of the 27th International Conference on Very Large Data Bases, pp.149158. Morgan Kaufmann Publishers Inc., 2001.

[7] M. Grund, M. Schapranow J. Kruege, J. Schaffner, A. Bog. In IEEE Symposium on Advanced Management of Information for Globalized Enterprises,pp 1-5, 2008.

[8] Dean Jacobs,Stefan Aulbach, "Ruminations on Multi-Tenant Databases", Datenbanksysteme in Bro, Technik undWissenschaft(German Database Conference) – BTW , pp. 514-521, 2007.

[9] Zhi Hu Wang, Chang Jie Guo, Bo Gao, Wei Sun, Zhen Zhang, Wen Hao, "A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing", In IEEE International Conference on e-Business Engineering, pp 94-101, 2008.

[10] E.J.Domingo, J.T.Nino, A.L.Lemos, M.L.Lemos, R.C. Palacios, J.M.G. Berbis, "A Cloud Computing-oriented Multi-Tenant Architecture for Business Information Systems", In IEEE 3rd International Conference on Cloud Computing, pp. 532-533, 2010.

[11] Cor-Paul Bezemer, Andy Zaidman, "Challenges of Reengineering into Multi-Tenant SaaS Applications", Report TUD-SERG-2010-012, Delft University of Technology Software Engineering Research Group Technical Report Series.

[12] http:www:ibm:com/developerworks/library/arsaassec/index.html. Last accessed on June 4, 2011.

[13] Peter C. Chapin, Christian Skalka and X. SeanWang, "Authorization in trust Management: Features and Foundations", In ACM Computing Surveys, Vol. 40, Issue 3, pp. 1-48,August 2008.

[14] Neuman C. RFC 1510, "The Kerberos network authentication service (V5) [S]", 1993.

[15] Phillip L. Hellewell, Timothy W. van der Horst and Kent E. Seamon, "Extensible Preauthentication in Kerberos", In 23rd Annual Computer Security Applications Conference, pp 201 - 210, 2007.

[16] Ching Lin and Vijay Varadharajan, "Trust based risk management for distributed system security- a new approach, In the Proc. of the IEEE First International Conference on Availability, Reliability and Security, 2006.

[17] Ping Liu, Rui Zong and Sizuo Liu, "A new model for Authentication and Authorization across Heterogeneous Trust-Domai", In the proc. of International Conference on Computer and Software Engineering, Volume 03, IEEE Computer Society, pages 789-792, 2008.

[18] W. W. W. Consortium. XQuery: The W3C query language for XML W3C working draft. Available at http : /www:w3:org/TR/xquery/; 2001.

[19] http://msdn:microsoft:com/en-us/library/aa479086:aspx Last accessed on June 4, 2011.

[20] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and M. Seibold, "A Comparison of Flexible Schemas for Software as a Service", SIGMOD , 2009, pp. 881-888.