ISSN No. 0976-5697

Volume 8, No. 5, May-June 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Survey on Security Mechanisms In NoSQL Databases

Vaishali J. Dindoliwala
Assistant Professor
C. B. Patel Computer College, Bharthana,
Surat, Gujarat, India

Dr. Rustom D. Morena
Professor, Department of Computer Science,
Veer Narmad South Gujarat University,
Surat, Gujarat, India

Abstract: Database is widely used in most of the applications to store, retrieve and manage large amount of data. Relational database work best when to handle a limited set of data. In web scale systems, relational databases degrade the performance due to the joins, locks and impedance mismatch. Thus, as a result various non-relational databases emerged known as NoSQL databases. With the development of the Internet market and new web technologies, NoSQL databases become a very popular for handling large amount of data. Object Oriented Database Management Systems come under the category of NoSQL databases as they are not queried using SQL. They store information as objects, offer advanced features such as a large number of data types and could also offer advanced features such as inheritance and polymorphism which are characteristics of object-oriented programming languages as well. With this emergence of new databases and their adoption by a large number of organizations, security becomes an important issue. In this paper, we have surveyed several NoSQL databases such as MongoDB, Cassandra GemStone, db4o and Objectivity/DB and presented security features available in these NoSQL databases and compare them.

Keywords: Relational database, NoSQL database, Object Oriented Database, Security, Authentication, Authorization, Auditing, Encryption

I. INTRODUCTION

The term NoSQL stands for "Not Only SQL" and they are not meant to replace the traditional databases such as relational databases but they are suitable to adopt big data where the traditional databases do not appropriate. They are developed by the web companies to fit their specific requirements regarding scalability, performance, maintenance and feature-set. They do not have properties of traditional relational databases and are generally not queried with SQL. Companies dealing with big unstructured data sets may benefit by migrating from a traditional relational database to a NoSQL database in terms of accommodating and processing large amount of data.

NoSQL data systems such as MongoDB, Cassandra provide schema-less modeling**,** in which the semantics of the data are embedded within a flexible connection topology and a corresponding storage model. This provides greater flexibility for managing large data sets while simultaneously reducing the dependence on the more formal database structure imposed by the relational database systems. To support big data processing, the platforms incorporate scaling in two forms of scalability - horizontal scaling and vertical scaling. In horizontal scaling the workload distributes across many servers. In this type of scalability multiple systems are added together in order to increase the throughput while in vertical scaling more processors, more memory and faster hardware are installed within a single server [1]. Unlike relational databases, NoSQL databases are designed to easily scale out as and when they grow.

In this paper, we focus on researches in security challenges in NoSQL databases. This paper describes the security mechanisms offered by various NoSQL databases like MongoDB, Cassandra, GemStone, db4o and Objectivity/DB. Section II describes various data storage models used in NoSQL databases. Section III is the literature review. Section IV describes security features offered by MongoDB, Cassandra, GemStone, db4o and Objectivity/DB. Section V is the discussion on security challenges in NoSQL databases and it also shows the comparison based on security features provided by MongoDB, Cassandra, GemStone, db4o and Objectivity/DB. Section VI is the conclusion.

II. NOSQL DATA STORAGE MODELS

A. Key-Value Store

In this data model, data objects are associated with distinct character strings called keys which are similar to hash table in data structure [6]. The key can be synthetic or auto-generated while the value can be string. The key-value type basically uses a hash table in which there exists a unique key and a pointer to a particular data item. The weakness of this model is that it does not provide any kind of traditional database capabilities such as atomicity, consistency etc. Other weakness is as the volume of data increases, maintaining unique values as keys may become more difficult. It requires the introduction of some complexity in generating character strings that will remain unique among an extremely large set of keys [3]. Some examples of NoSQL databases that uses key-value mechanism are Redis, Aerospike, OrientDB and MUMPS.

B. Document Store

A document store is similar to a key-value store in that stored objects are associated and accessed via character-string keys. The difference is that the values being stored, which are referred to as “documents,” provide some structure and encoding of the managed data [2] [3] [6]. That means these databases store records as “documents” where a document can generally be thought of as a grouping of key-value pairs. Keys are always string and values can be stored as strings, numeric, booleans arrays and other nested key-

value pairs. Values can be nested to arbitrary depths. In a document database, each document carries its own schema. Some examples of NoSQL database that uses document store mechanism are CouchDB, MongoDb, RavenDB, BaseX and eXist.

C. Column Store

In this type of data store, data are stored in cells grouped in columns of data rather than as rows of data. Relational databases store a single row as a continuous disk entry. Different rows are stored in different places on disk while columnar databases store all the cells corresponding to a column as a continuous disk entry which makes the search or access faster [2][7]. Some examples of NoSQL databases that use column store mechanism are Cassandra, Vertica and HBase.

D. Object Store

Object data stores are essentially a hybrid approach to data storage and management. Object data stores and object databases seem to bridge the worlds of schema-less data management and the traditional relational models. On the one hand, approaches to object databases can be similar to document stores except that while the document stores explicitly serialize the object so the data values are stored as strings, object databases maintain the object structures [3]. Object data store offers all the features of object oriented programming such as data encapsulation, polymorphism and inheritance.

E. Graph Databases

Graph databases consist of edges between nodes. Both nodes and their edges can store additional properties such as key-value pairs [14]. The strength of a graph database is in traversing the edges between the nodes. But they generally require all data to fit on one machine that limits its scalability. Examples of such type of NoSQL databases are OrientDB, Allegro and Stardog.

III. LITERATURE REVIEW

In their survey, J. Ahmed and R. Gulmeher [2], described that because of its heterogeneous nature than in homogeneous environments, the protection of integrity in NoSQL database system is difficult. It is difficult to implement integrity constraints because of its schema-less nature and absence of central control. They have also discussed when there is no central management security point, protecting the name servers, nodes and those clients become difficult. As heterogeneous data is stored together in one database as opposed to relational models, it becomes difficult to handle Role Based Access Control (RBAC).

L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes and J. Abramov [5] have discussed security issues in two of the most popular NoSQL databases Cassandra and MongoDB. They had outlined their security features and problems. According to them, the main problems with Cassandra and MongoDB are the lack of encryption support for data files, weak authentication between clients and servers, simple authentication, vulnerability to SQL injection and DOS attack. It is also mentioned that both of them do not support RBAC and fine-grained authorization.

A. K. Zaki [7] also has discussed various security challenges on NoSQL databases such as transactional integrity, authentication methods and susceptibility to injection attacks, lack of consistency and insider attacks. According to him, NoSQL databases make use of many distributed commodity servers, it does not assure consistent results at all time, as all participating commodity servers may not entirely synchronized with other servers holding latest information. If a single commodity server gets fail, it results in load imbalance among other commodity servers. NoSQL databases incorporate the weak authentication mechanism and weak password storage techniques. Some NoSQL databases enforce authentication mechanism at local node level but fail to enforce authentication across all commodity servers. Complex integrity constraints cannot be added in NoSQL database architecture because it results in failure to meet the NoSQL's main objective of attaining better performance and scalability.

IV. SECURITY FEATURES OF NOSQL DATABASES

NoSQL databases increasingly being used to handle big data challenges, are still very new in the field with respect to their feature set, especially with respect to security, so fine-grained permissions or access control in these systems are yet to be provided.

In this section, we have presented various security features like authentication, authorization, data encryption and auditing capabilities provided by NoSQL databases such as MondoDB, Cassandra, GemStone, db4o and Objectivity/DB.

A. MongoDB

MongoDB is a schema-free document database. A MongoDB database contains one or more collections of documents. A collection is equivalent term for a table but it has no pre-defined schema. Document is the unit of storing data in a MongoDB database. They are analogous to the records of relational databases. It is a set of fields. A document can contain complex structures such as lists or even other documents. Once the first document is inserted into a database, a collection is created automatically and the inserted document is added to this collection. Every document has an id [5]. Insert, update, and delete operations can be performed on a collection [11]. Each document can match the data fields of the represented entity, even if the data has substantial variation. In practice, however, the documents in a collection share a similar structure. [12]

MongoDB has its own ad-hoc query language named Mongo Query Language. To retrieve certain documents from a database collection, a query document is created containing the field conditions to match and then returns a cursor. The cursor is iterated to access the resulting document. [11][12]

As MongoDB databases do not have strictly defined database schemas, using JavaScript for query syntax allows developers to write arbitrarily complex queries against irrelevant document structures. NoSQL database engines that process JavaScript containing user-specified parameters can be vulnerable to injection attacks. MongoDB, for

example, supports the use of JavaScript functions for query specifications and map/reduce operations. Most of the internal commands are actually short JavaScript routines [12]. JavaScript functions, stored in db.system.js collection are also available to the database users.

MongoDB provides various security features such as authentication, access control and authorization, encryption and auditing.

1) ***Authentication:*** In MongoDB, authentication of users can be managed from within the database itself or via integration with an external mechanism i.e. LDAP, x.509 PKI certificates or a Kerberos service. [13]

MongoDB authenticates entities on a per-database level using the SCRAM-SHA-1 IETF standard. Users are authenticated via the authentication command while database nodes can be authenticated to the MongoDB cluster via keyfiles. SCRAM-SHA-1 is the default authentication mechanism for MongoDB. It is an IETF standard, RFC 5802 that defines best practice methods for implementation of challenge-response mechanisms for authenticating users with passwords. SCRAM-SHA-1 verifies the supplied user credentials against the user's name, password and authentication database. The authentication database is the database where the user was created and together with the user's name, serves to identify the user [4].

With LDAP integration, MongoDB can authenticate and authorize users directly against corporate LDAP infrastructure to enforce centralized access policies. [4]

With MongoDB Enterprise Advanced, authentication using a Kerberos service is supported. Kerberos is an industry standard authentication protocol for large client/server systems, allowing both the client and server to verify each other's identity. Before users can authenticate to MongoDB using Kerberos, they must first be created and granted privileges within MongoDB [4].

2) ***Authorization:*** In MongoDB, administrators can define the specific permissions for an application or user has and they can decide what data an application or user can see when querying the database. The authorization mechanism in MongoDB includes roles, LDAP authorization and Field-Level Security with Read-Only Views.

Authorization privileges can be based on the specific functionality a user needs in their role. Privileges are assigned to roles and roles are in turn assigned to users. MongoDB provides a number of built-in roles that administrators can use to control access to a MongoDB system. MongoDB uses the combination of the database name and the role name to uniquely define a role. Roles are granted to the users. Each user is identified by unique id. Outside of role assignments, the user has no access to the system. A privilege consists of a specified resource and the actions permitted on the resource. A resource is a database, collection, set of collections or the cluster. Privileges are set for each resource according to the requirement. The MongoDB supports a simple role-based authentication system that allows you to control who has access to each database and the level of access they are granted. For normal users to have access on to two databases, their credentials and rights must be added to both databases. [8]

MongoDB also support authorization via LDAP. This enables existing user privileges stored in the LDAP server to be mapped to MongoDB roles without users having to be recreated in MongoDB itself. [4]

DBAs can define non-materialized views that expose only a subset of data from an underlying MongoDB collection. As a result, risks of data exposure are dramatically reduced [4]. DBAs can define a view of a collection that is generated from an aggregation over another collections or view. Permissions granted against the view are specified separately from permissions granted to the underlying collections. As views are non-materialized, the view data is generated dynamically by reading from the underlying collections when a user queries the view. This reduces data duplication in the database and eliminates inconsistencies between the base data and view. Views are defined using the standard MongoDB Query Language and aggregation pipeline.

3) ***Auditing:***

The auditing framework captures administrative actions such as schema operations as well as authentication and authorization activities along with read and writes operations to the database [4]. Administrators can construct and filter audit trails for any operation against MongoDB. For example, it is possible to log and audit the identities of users who accessed specific documents and any changes they made to the database during their session. Administrators can configure MongoDB to log all actions or apply filters to capture only specific events, users or roles. MongoDB Enterprise Advanced also supports role-based auditing. It is possible to log and report activities by specific role.

4) ***Data Encryption:*** Data can be encrypted over the network and at rest in permanent storage and backups. MongoDB Enterprise Advanced supports FIPS 140-2 encryption if run in FIPS Mode with a FIPS validated Cryptographic module. There are multiple ways to encrypt data at rest with MongoDB. Encryption can be implemented at the application level or via external file system and disk encryption solutions. The storage engine encrypts each database with a separate key. The key-wrapping scheme in MongoDB wraps all of the individual internal database keys with one external master key for each server [4].

B. ***Cassandra***

Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure. The basic components in Cassandra are – nodes, data centers and clusters. Logically, the data in a cluster is organized into keyspaces (databases), which contain tables. Tables contain rows, and rows have columns. Cassandra does not force individual rows to have all the columns. Keyspace is the container for data. A column family is a container for

collection of rows and each row contains columns. Fig. 1 shows the view of a keyspace while Fig. 2 shows the basic structure of a column family. [14]

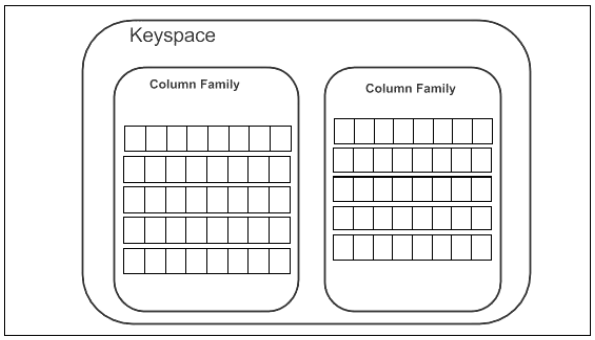

Figure 1. Schematic view of a keyspace

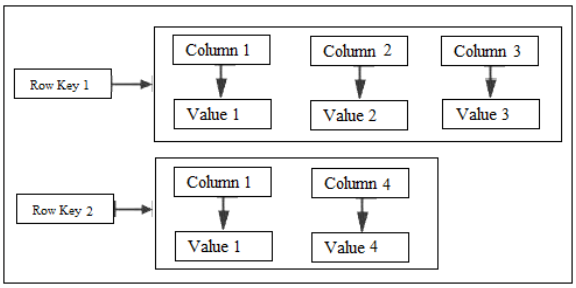

Figure 2. Structure of column family

Physically, the content of a table row is always stored on the hard drive of at least one node in the cluster and depending on how the keyspace has been defined upon creation, this row content is replicated to 0 or more other nodes in the cluster. Cassandra uses a query language called Cassandra Query Language (CQL) which is something like SQL. Following are the security features offered by Cassandra:

1) ***Authentication:*** Cassandra supports only internal authentication mechanism. External authentication mechanism such as through 3rd party authentication/authorization tool e.g. LDAP is not supported yet. Cassandra 3.0 authentication is roles-based and stored internally in Cassandra system tables. Administrators can create, alter, drop or list roles using CQL commands with an associated password. The internal authentication is used to access Cassandra keyspaces and tables and by cqlsh and DevCenter to authenticate connections to Cassandra clusters and sstableloader to load SSTables. [16]

2) ***Authorization:*** Cassandra uses the GRANT/REVOKE security paradigm to manage object permissions as a part of authorization. In Cassandra, permissions on database resources are granted to roles. Roles-based access control is available in Cassandra 2.2 and later. Roles enable authorization management on a larger scale than security per user can provide. A role is created and may be granted to other roles. Hierarchical sets of permissions can be created [17]. There is various permission/resource combinations currently set. The various resources are keyspace, roles, tables, index and functions. And the permissions that can be applied on these resources are : [18]

a) Create - keyspace, table, function, role, index
b) Alter - keyspace, table, function, role
c) Drop - keyspace, table, function, role, index
d) Select - keyspace, table
e) Modify (Insert, Update, Delete, Truncate) - keyspace, table
f) Authorize (Grant and Revoke Permissions) - keyspace, table, function, role, MBean (in Cassandra 3.6 and later)
g) Describe – list roles
h) Execute – functions

3) ***Data Encryption:*** There is a support for at-rest data encryption through Transparent Data Encryption (TDE) from version 3.2. [16]

4) ***Auditing:*** Auditing is available in Enterprise Cassandra as a log4j-based integration and a per node basis. To get the maximum audit information, turning on auditing on every node is recommended. Filters are available for logging using a combination of the following categories – ADMIN, ALL, AUTH, DML, DCL and QUERY. [16]

C. ***GemStone***

GemStone is an Object-Oriented Database Management System. It supports multiuser environment. Multiple user sessions can be active at the same time and each user may have multiple sessions open. In GemStone, security is provided at several levels, from login authorization to object access privileges. Following are the security features offered by GemStone:

1) ***Authentication :*** In GemStone, each user is identified by unique userID and password. It supports its own authentication protocol as well as the Kerberos scheme. A user is represented by an instance of class UserProfile. A UserProfile contains userID, password, default authorization information, privileges and group membership about a user. Only users who have a UserProfile can log on to the system. [15]

2) ***Authorization :*** Authorization exists within GemStone and controls individual object access. It is implemented at the lowest level of basic object access to prevent users from circumventing the authorization checking. Objects can't be accessed from any language without proper authorization.

When objects are created, they are assigned to a default segment if not specified. Segment is used to control ownership of and access to objects. With segment, one can abstractly group objects, specify who owns the objects, who can read them and who can write them. Each repository in GemStone is composed of segments. All objects assigned to a segment also have exactly the same protection that is, if you can read or write one object assigned to a certain segment; you can read or write them all. Each segment is owned by a single user and all objects assigned to the same segment have the same owner. Groups of users can have read, write or no access to a segment. Likewise, any authorized GemStone user can have read, write or no access to a segment. Whenever an application tries to access an object, GemStone compares the object's authorization

attributes in the segment associated with the object with those of the user whose application is attempting access. If the user is appropriately authorized, the operation proceeds. If not, GemStone returns an error notification. The user that owns the segment controls what access other users has to it. By default, the owner has write access to the objects in a segment. [15]

3) ***Auditing:*** Administrator can also configure a GemStone system to monitor failures to log in and he can also configure system to disable a user account after a certain number of failed attempts to log into the system through that account. [15]

D. *db4o*

db4o is only an object database and not an object database management system. It is not standalone that means everything can be done through an application. Therefore the application can fully control authentication and authorization as and when required. On the level of db4o – in embedded mode – there are no integrated security mechanisms. There is no user management. Access to the data files should be controlled by using the mechanisms of the underlying platform. In client-server mode of db4o, user has to specify userID and password credentials to connect and access to the server. [10]

There exists very limited authorization capability. One can switch files where objects can be stored in different files and credentials can be granted based on a file level which seems to be a very limited approach and is therefore not considered. It provides the eXtended Tiny Encryption Algorithm (XTEA). [9]

E. *Objectivity/DB*

Objectivity/DB is another object database. Since the Objectivity/DB schema is created from internal class definitions from the programming language, it is easier to maintain changes to the schema.

Objectivity/DB provides access control to prevent certain applications from updating or changing the schema. Objectivity/DB also provides a feature for supporting changes to the schema. Ideally changes should be phased in and not adjusted at runtime to ensure consistency across the application and other applications using different programming languages. However this feature in addition to object conversion allows the schema to be changed quite easily and automatically update the previously stored objects to be compatible with the new schema. [9]

Access to database can be controlled by implementing security based on a user name, a password or both. Objectivity/DB relies completely on the operating system and file systems for access control. There is no role-based or discretionary access control in place. It also does not appear to provide any inbuilt encryption so it is another feature that would need to be built into a middle layer. [9]

V. Discussion on Security Challenges in NoSQL Databases

The main focus of NoSQL databases is handling the new data sets with less priority on security. Each NoSQL database has different API and different query systems, requiring a full learning curve for developers every time a new NoSQL database is introduced. For example Cassandra uses CQL, MongoDB uses mongo query language etc. Therefore it becomes difficult for the user to switch from one NoSQL database provider to another. The lack of common query language, consistency support and transaction system puts limitations towards the adoption of NoSQL databases in certain business applications like banking systems. Also schema-free structure does not allow fine grained access control. NoSQL databases do not provide any feature of embedding security in the database itself. Developers need to impose security in the middleware.

Role-based access control is also difficult to enforce because of schema-free structure of some of the NoSQL databases. Different data are stored in one huge database in this type of database. As heterogeneous data is stored together in one database as opposed to relational models this becomes a challenge. NoSQL databases like MongoDB and Cassandra are failed to ensure ACID properties. They are generally resides on the concept of BASE (Basically available, Soft state, Eventual consistency). The focus of BASE is the permanent availability rather than consistency.

In GemStone database, object level privileges are given by authorizing the Segment. So Administrator has to plan the segment according to the privileges given on various objects to the users. So, in GemStone security is easier to implement if it is built into the application design at the beginning, not added later. Apart from this, the other limitation of GemStone includes the maintenance of segment. Many objects can refer to any segment. If the segment has to be removed then we must first decide whether we wish to remove all the objects assigned to it or whether we wish to reassign some or all of them to another segment. As long as an object that you or another user can refer to within your application is assigned to the segment, GemStone cannot reclaim the storage used by the segment. [15]

The Table 1 summarizes and compares various security features provided by NoSQL databases.

Table 1: Comparison among MongoDB, Cassandra, GemStone, db4o and Objectivity/DB

Sr. No.	*Features*	*MongoDB*	*Cassandra*	*GemStone*	*db4o*	*Objectivity/DB*
1	Data Store Category	Document store	Column store	Object store	Object store	Object store
2	Authentication	Provides internal as well as external authentication	Provides only internal authentication mechanism.	Provides its own authentication protocol as well	Provides only internal authentication mechanism.	Provides Kerberos authentication mechanism by

		mechanism.		as Kerberos		default. Also uses Advanced Multithreaded Server (AMS) protocol.
3	Authorization	Provided by roles, LDAP authorization and Field-Level Security with Read-Only Views	permission/resource combinations are set according to the roles for the user	Provides Object level authorization	Offers file level authorization.	Relies on the operating system and file systems for access control
4	RBAC	Supported	Supported	Not Supported	Not Supported	Not Supported
5	Encryption	Supported	Supported	Not provided	Provides the eXtended Tiny Encryption Algorithm	Doesn't provide any inbuilt encryption
6	Auditing	Supported	Supported with limited capability	Supported with limited capability	No auditing capabilities	No auditing capabilities

VI. CONCLUSION

NoSQL databases are designed to provide real-time performance while managing large amount of data. They do not use SQL as the primary query language, also they do not typically require fixed table schemas. Each NoSQL databases are built to handle different challenges and hence security is never part of the model of its design stage. Developers have to embed security in the middleware to overcome the security issues in NoSQL databases. However, clustering aspect of NoSQL databases poses additional challenges to the robustness of such security practices. As opposed to relational databases, they trade consistency and security for performance and scalability.

VII. REFERENCES

[1] E. Sahafizadeh and M. A. Nematbakhsh, "A survey on security issues in Big Data and NoSQL", Advances in Computer Science: an International Journal, vol. 4, issue 4, no.16 , July 2015, pp. 68-72, ISSN : 2322-5157.

[2] J. Ahmed and R. Gulmeher, "NoSQL databases: New trend of databases, emerging reasons, classification and security issues", International Journal of Engineering Sciences and Research Technology, vol. 4(6), June, 2015, pp. 176-184, ISSN: 2277-9655.

[3] P. Badlani.,"NoSQL in action - A new pathway to database", International Journal of Science and Research, vol. 5, issue 6, June 2016, pp. 872-877.

[4] A MongoDB White Paper, "MongoDB Security Architecture", Dec. 2016.

[5] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes and J. Abramov.,"Security issues in NoSQL databases", International Joint Conference of IEEE TrustCom, 2011, pp. 541-547, ISBN**:** 978-1-4577-2135-9.

[6] Dadapeer and N. M. Indravasan., G. Adarsh," A survey on security of NoSQL databases", International Journal of Innovative Research in Computer and Communication Engineering, vol. 4, issue 4, Apr. 2016, pp. 5249 – 5254, ISSN : 2320-9798.

[7] A. K. Zaki, "NoSQL databases: new millennium database for big data, big users, cloud computing and its security challenges", International Journal of Research in Engineering and Technology, vol. 03, issue 03, May 2014, pp. 403 – 409, ISSN: 2321-7308.

[8] E. Plugge, D. Hows, P. Membrey and T. Hawkins,"The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB", Third Edition, Apress.

[9] T. Muirhead,"Object databases and object persistence for openEHR", Thesis of Bachelor of Information Technology, Computer and Information Science, UniSA, 2009.

[10] P. Hauser, "Review of db4o from db4objects", University of Applied Sciences Rapperswil, Switzerland.

[11] MongoDB Documentation Release 2.6.9, MongoDB Documentation Project, March 27, 2015.

[12] K. Chodorow, "MongoDB: The Definitive Guide", second edition, O'Reilly, May 2013.

[13] A MongoDB White Paper, MongoDB Architecture Guide, MongoDB 3.2.

[14] E. Hewitt, "Cassandra : The Definitive Guide", O'Reilly, 2010, ISBN: 978-1-449-39041-9.

[15] GemStone Programming Guide, GemStone Systems Inc., GemStone Version 5.0, July 1996.

[16] Fidels Cybersecurity, "Current data security issues of NoSQL databases", January 2014.

[17] https://docs.datastax.com/en/cql/3.3/cql/cql_using/useSecureRoles.html

[18] https://docs.datastax.com/en/cql/3.3/cql/cql_using/useSecurePermission.html