NIST Internal Report
NIST IR 8504

Access Control on NoSQL Databases

Vincent C. Hu

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8504

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Internal Report
NIST IR 8504

Access Control on NoSQL Databases

Vincent C. Hu
Computer Security Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8504

May 2024

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Publication History
Approved by the NIST Editorial Review Board on 2024-05-01

How to Cite this NIST Technical Series Publication:
Hu VC (2024) Access Control on NoSQL Databases. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency or Internal Report (IR) NIST IR 8504. https://doi.org/10.6028/NIST.IR.8504

Author ORCID iDs
Vincent C. Hu: 0000-0002-1648-0584

Contact Information
ir8504-comments@nist.gov

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930

Additional Information
Additional information about this publication is available at https://csrc.nist.gov/pubs/ir/8504/final, including related content, potential updates, and document history.

Abstract

NoSQL database systems and data stores often outperform traditional RDBMS in various aspects, such as data analysis efficiency, system performance, ease of deployment, flexibility/scalability of data management, and users' availability. However, with an increasing number of people storing sensitive data in NoSQL databases, security issues have become critical concerns. NoSQL databases suffer from vulnerabilities, particularly due to the lack of effective support for data protection, including weak authorization mechanisms. As access control is a fundamental data protection requirement of any database management system DBMS, this document focuses on access control on NoSQL database systems.

Keywords

access control; attribute-based access control; authorization; database systems; No-SQL; SQL.

Reports on Computer Systems Technology

Table of Contents

List of Tables

List of Figures

Acknowledgments

The author would like to express his thanks to Isabel Van Wyk and Jim Foti of NIST for their detailed editorial review of both the public comment version and the final publication.

Executive Summary

NoSQL stands for “not only SQL” or “non-SQL,” which typically refer to any non-relational database that stores data in a format other than relational tables. It shifts away from relational database management systems (RDBMS) for dealing with enormous and constantly growing data and infrastructure needs and increasingly uses the Web 3.0 framework for big data and real-time web applications, including handling unstructured data like documents, emails, and social media. NoSQL databases are particularly useful for managing unstructured or very large data objects stored across distributed and cooperating devices. Use cases for retrieving such data range from critical scenarios (e.g., storing financial data and healthcare records) to more casual applications (e.g., chat log data, video, and images, readings from smart devices). Major Web 2.0 companies have developed or adopted different flavors of NoSQL databases to meet their growing data and infrastructure needs.

NoSQL database systems and data stores often outperform traditional RDBMS in various aspects, such as data analysis efficiency, system performance, ease of deployment, flexibility/scalability of data management, and users’ availability. However, with an increasing number of people storing sensitive data in NoSQL databases, security issues have become critical concerns. NoSQL databases suffer from vulnerabilities, particularly due to the lack of effective support for data protection, including weak authorization mechanisms. As access control is a fundamental data protection requirement of any database management system (DBMS), this document discusses access control on NoSQL database systems by illustrating the NoSQL database types along with their support for access control models and describing considerations from the perspective of access control.

1. Introduction

NoSQL stands for “not only SQL” or “non-SQL,” which typically refers to any non-relational database that stores data in a format other than relational tables. It emerged in the late 2000s, triggered by the growing popularity of distributed systems, such as cloud computing and big data, which required a shift away from relational databases (RDBMS) toward NoSQL databases. Organizations now deal with enormous and constantly growing data and infrastructure needs or increasingly use the Web 3.0 framework for big data and real-time web applications, including handling unstructured data like documents, emails, and social media [AA, NO].

NoSQL databases are particularly useful for managing unstructured or very large data objects stored across distributed and cooperating devices. Use cases for retrieving such data range from critical scenarios (e.g., storing financial data and healthcare records) to more casual applications (e.g., chat log data, video, images, and readings from smart devices). Major Web 2.0 companies like Amazon (Dynamo), Google (BigTable), LinkedIn (Voldemort), and Facebook (Cassandra) have developed or adopted different flavors of NoSQL databases to meet their growing data and infrastructure needs. Their success has inspired many of today’s NoSQL applications [AA, CJ].

NoSQL database systems and data stores often outperform traditional RDBMS in various aspects, such as data analysis efficiency, system performance, ease of deployment, flexibility/scalability of data management, and users’ availability. However, with an increasing number of people storing sensitive data in NoSQL databases, security issues have become critical concerns. NoSQL databases suffer from vulnerabilities, particularly due to the lack of effective support for data protection, including weak authorization mechanisms. As access control is a fundamental data protection requirement of any database management system (DBMS) [FF], this document discusses access control on NoSQL database systems by illustrating the NoSQL database types along with their support for access control models and describing considerations from the perspective of access control. Note that an access control system may store and manage access control data (e.g., subjects, objects, actions, and attributes) in external systems rather than the NoSQL database itself, which has a wide range of different implementations not discussed in this document.

This document is organized as follows:

- Section 1 is the introduction,
- Section 2 provides an overview of NoSQL database systems,
- Section 3 introduces access control models for NoSQL database systems,
- Section 4 describes considerations for NoSQL systems from the perspective of access control.
- Section 5 is the conclusion,
- References list articles referred by this document,
- Appendix A provides an example application of Graph model.

2. Overview of NoSQL Database Systems

Applications in web services, e-commerce, mobile computing, and social media require storing and processing vast amounts of structured and unstructured data driven by the significant increase in global datasets known as "big data," which includes data with high volume, velocity, and variety. However, handling such immense data becomes increasingly complicated in terms of processing and meeting users' requirements, such as scalability, performance control, high availability, low latency, workload distribution, and managing big data applications [AA].

Traditional RDBMSs often fall short when compared to NoSQL database systems, which utilize distributed and collaborative devices to store and retrieve data. NoSQL databases offer the flexibility to rapidly adapt to changes in the software stack and allow data to be distributed across multiple servers and regions in the cloud, scale out instead of scaling up, and intelligently geo-place data to optimize performance [MO].

2.1. Types of NoSQL Databases Systems

In general, there are four major types of NoSQL models: key-value, document, wide-column, and graph databases.

2.1.1. Key-Value Model

The key-value model, as shown in Fig. 1, stores data in a schemaless form, making them simple, efficient, and powerful. In this model, each database item contains keys and corresponding values, similar to an RDBMS with only two columns (i.e., key and the value). Data can be efficiently retrieved by using a unique key, which serves as an index like a hash table. The values in a key-value NoSQL database can be simple data types, like strings and numbers, or complex objects [HI]. The key-value model is primarily used for caching, session management, and leaderboard applications [WT].

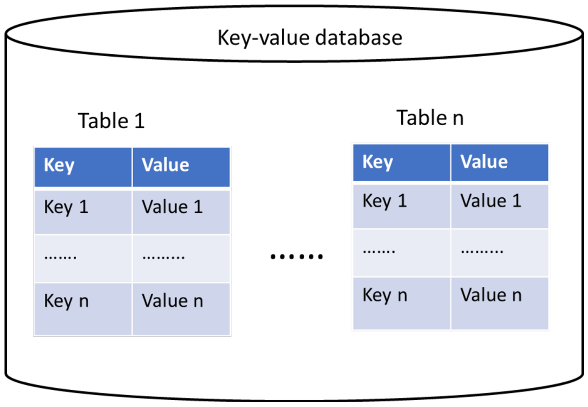

Fig. 1. Key-value database model

2.1.2. Wide-Column Model

The wide-column model, as shown in Fig. 2, stores data in tables, rows, and dynamic columns, similar to the key-value model. However, the key in this model is an integration of the row, column, and/or timestamp and refers to one or many columns as a "column family.” Each column family is equivalent to a table in an RDBMS, enabling analytics on a small number of columns for data mining and web applications. This design allows for more efficient data reading and retrieval with higher speed compared to traditional RDBMS [AA, HI]. It is well-suited for complex datasets and storing large amounts of data in distributed systems as it enables the easy addition of new columns by creating new files, eliminating the need to rebuild the entire table, as required in RDBMS [AA]. The model is usually optimized for specific use cases (e.g., Facebook’s Inbox search feature), allowing it to efficiently handle over 100 million users continuously using the system [OG]. Other popular use cases of the wide-column model include the Internet of Things (IoT), inventory management, and big data processing.

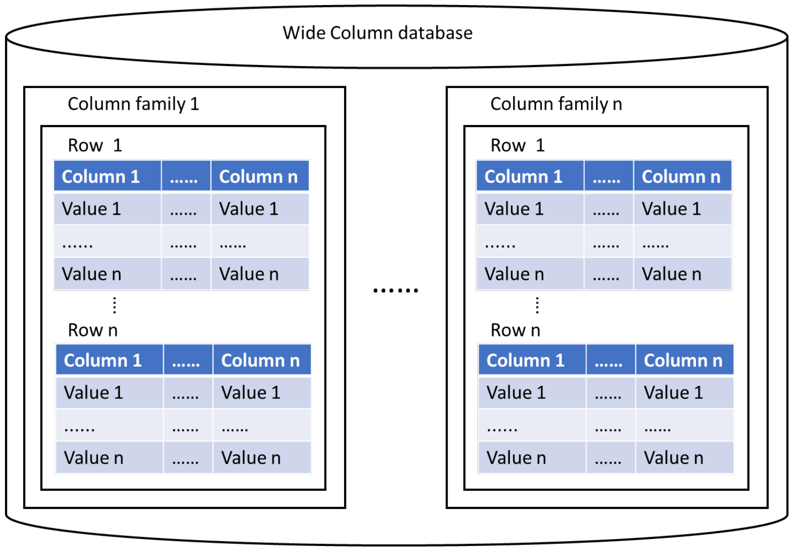

Fig. 2. Wide-column database model

2.1.3. Document Model

The document model, as shown in Fig. 3, stores data in documents, similar to JSON, BSON, and XML documents. In this model, data is organized into collections with a unique key that serves as an index for fast querying. A collection contains documents without a schema. Rather, each document contains pairs of fields and values of various types, such as strings, numbers, Booleans, arrays, and others. The data within documents can include structured data, semi-structured data (e.g., XML files), or unstructured data (e.g., text), which allows for a dynamic structure and the easy modification, addition, or deletion of fields. This flexibility allows documents to be stored and retrieved in a form that closely resembles the data objects used in applications, reducing the need for data translation during application use [HI] and contributing to high performance and horizontal scalability. This model has applications in blog software, content management systems [AA], and product catalogs, big data, and analytics [WT].

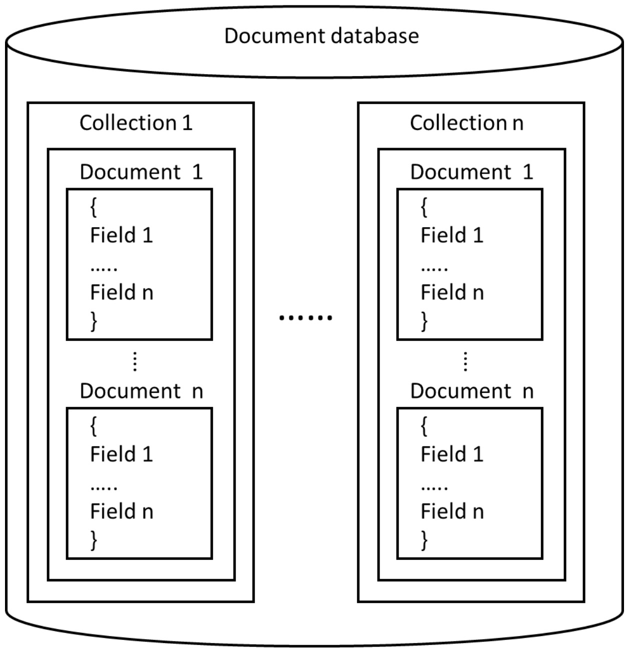

Fig. 3. Document database model

2.1.4. Graph Model

The graph model, as shown in Fig. 4, represents and stores data based on relationships. It is a schemaless model, and data is structured using nodes and edges. Nodes typically store data entities, while edges represent the relationships or links between the data nodes. The model is scalable but complex as it often utilizes shortest path algorithms to optimize data queries for real-time results. In this model, the efficiency of a query depends on the number of relationships among the nodes [HI]. Graph-based databases have various applications, such as recommendation systems, social networking, identity and access management (IAM), and content management [AA WT].

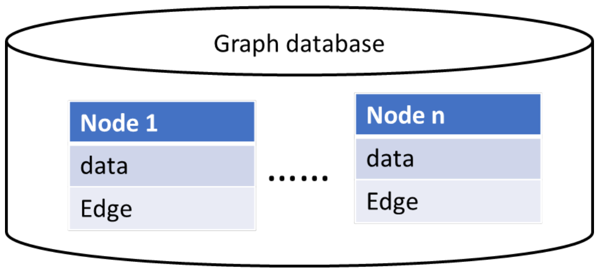

Fig. 4. Graph database model

2.2. Features of NoSQL Database Systems

NoSQL databases offer flexible and simple data models, horizontal scalability, and fast (i.e., primitive) queries, as well as security deficiencies [MO]. Table 1 compares the features of RDBMS with those of NoSQL databases [AH, SL].

Table 1. Comparison of RDBMS and NoSQL features

Database Model	RDBMS	NoSQL
Database	Relational	Non-relational
Scheme	Fixed and structured	Dynamic and unstructured
Queries	Complex queries using JOIN	Unsupported
Scalable	Vertical	Horizontal
Properties	Atomicity, consistency, isolation, and durability (ACID)	Consistency (eventually), availability, partial tolerance (CAP)

2.2.1. Flexible Data Model

NoSQL databases provide flexible schemas that enable easy database changes and seamless integration with database applications. For example, in document-based systems, documents do not need to follow the same schema, which allows for faster creation and maintenance of documents with minimal overhead.

2.2.2. Horizontal Scalability

NoSQL databases support horizontal scaling, which means that they handle increased capacity by distributing data across multiple servers, thus avoiding the need to migrate to a larger server when capacity exceeds the requirements of the current server. NoSQL key-value systems have a simple structure with only two columns (i.e., key and the value), enabling horizontal scaling without the need for complex field joins. Similarly, wide-column systems can handle more complex data structures by adding new columns through the creation of a new file.

2.2.3. Fast Queries

NoSQL databases store data in a way that optimizes queries by utilizing simple and efficient (i.e., primitive) query language [OG] without the need for join operations that can degrade query performance in typically normalized DBMS using SQL. For example, in document systems with open formats like XML and JSON, building a document does not require foreign keys. This allows for dynamic relationships between documents, making them independent of each other. Wide-column systems are also designed for efficient data reading and retrieval, resulting in better performance [HI].

2.2.4. Security Deficiencies

Despite their advantages, NoSQL databases lack a mechanism for handling and managing data consistency and maintaining integrity constraints (e.g., using foreign keys). Additionally, they often have limited support for security at the database level [OG].

3. Access Control Model on NoSQL Database Systems

This section illustrates how to apply access control properties on NoSQL models assuming that the access control system stores and manage access control data (e.g., subjects, objects, and attributes) in the NoSQL database.

3.1. Relationship Structures of NoSQL Models

Schemaless relationships between NoSQL data can be constructed using hierarchical structures. The key-value, wide-column, and document models can be represented in a tree data structure, as shown in Fig. 5, with a maximum of five tree levels if the NoSQL database itself is represented at the first (i.e., root) level.

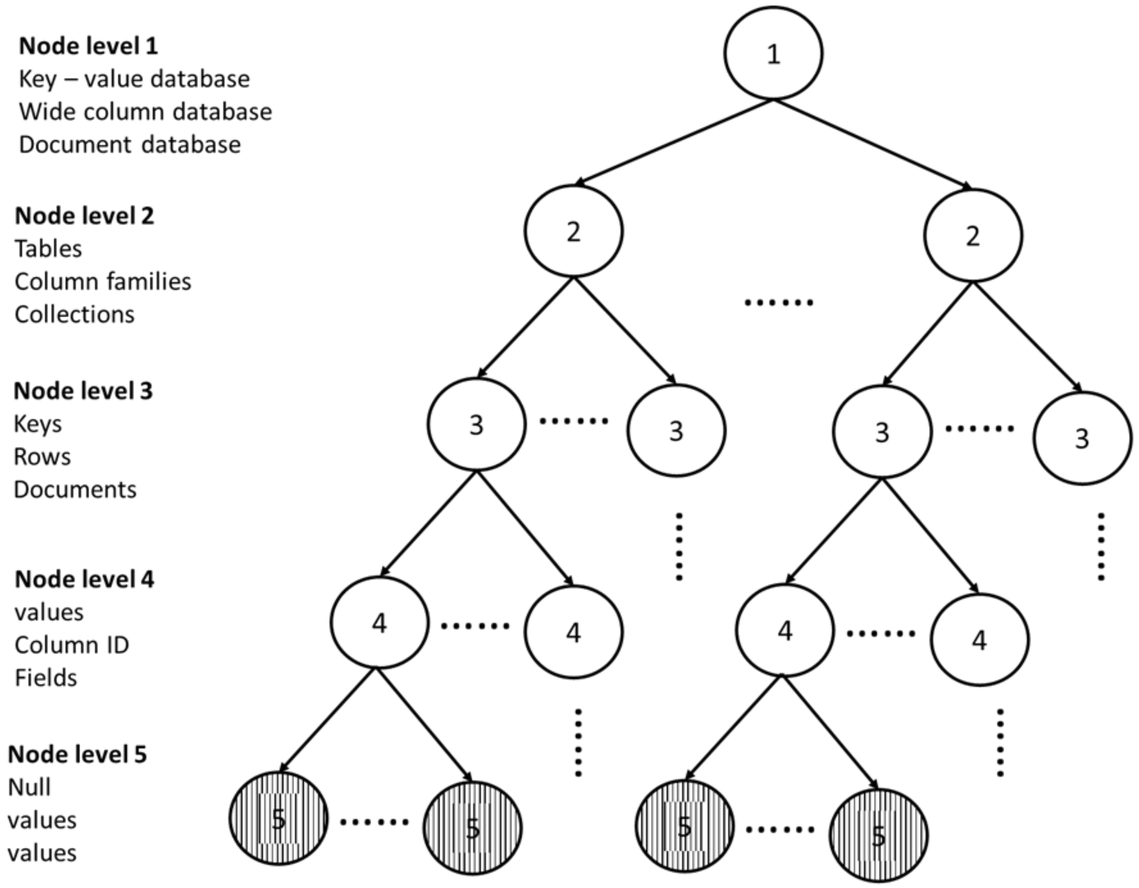

Fig. 5. Hierarchical structure relationships of the key-value, wide-column, and document models

Nodes in the second tree level represent the Tables, Column families, and Collections for key-value, wide-column, and document models, respectively. Nodes in the third tree level represent the data that can be indexed from the nodes in the second tree level, which are Key, Rows, and Documents for the key-value, wide-column, and document models, respectively. Nodes in the fourth tree level represent the data that can be indexed from nodes in the third tree level, which are values, Column IDs, and Fields for the key-value, wide-column, and document models, respectively. Nodes in the fifth tree level represent the data that can be indexed from nodes in the fourth tree level. There is no offspring on this level for the key-value model. It is essential to consider these hierarchical structures when designing access control on NoSQL databases to effectively leverage their schemaless nature and optimize data access.

Unlike hierarchical tree structures, the relationship structure of the graph model is represented by a directed graph, as shown in Fig. 6.

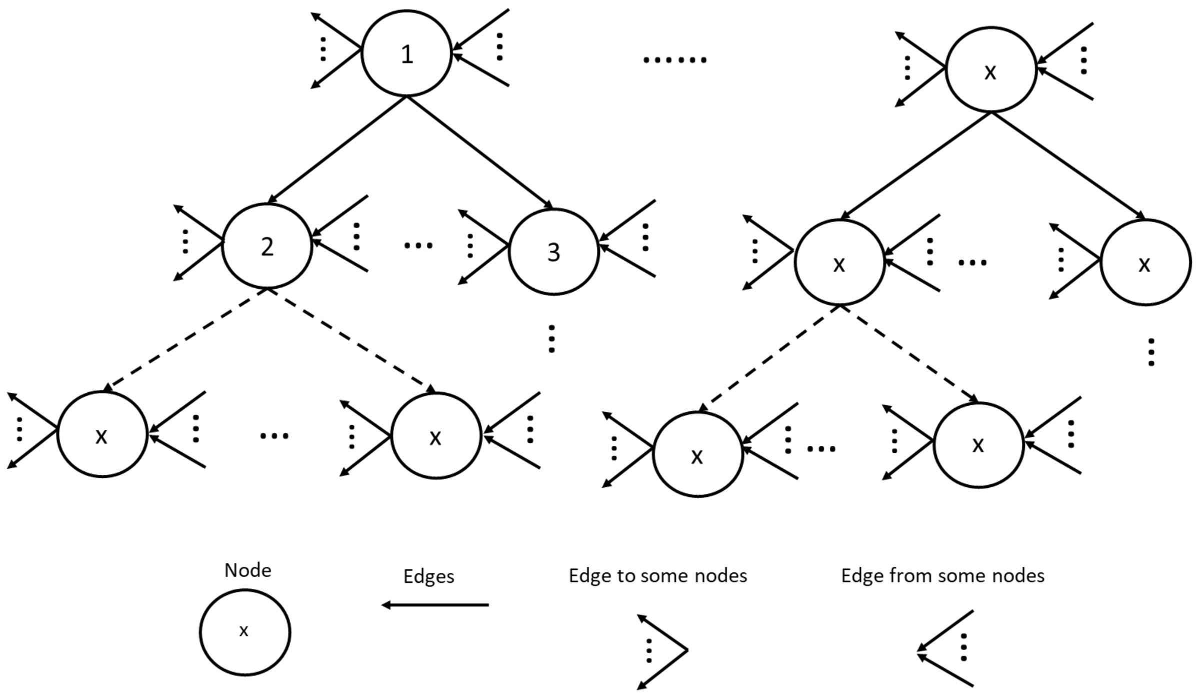

Fig. 6. Directed graph relationship structure of the graph model

In this model, each node can have more than one edge, which represent the connections or relationships between nodes in the graph. The directed graph allows for complex and interconnected data relationships, making it ideal for applications in which data connections and dependencies are critical to the analysis and processing of information.

3.2. Access Control Rules from NoSQL Relationship Structures

Enforcing access control policies — especially for fine-grained access control on schemaless NoSQL databases with data relationships in heterogeneous structures — presents challenges due to the absence of a reference data model and related manipulation language. This exacerbates the enforcement of access control policies on the protected data [CF]. Additionally, managing access control on NoSQL databases is not as straightforward as it is for RDBMSs, making it challenging to query “who can access what” in a straightforward manner. However, any static access control policy model based on subjects and object attributes can still be applied to NoSQL databases. The following sections describe how access control rules can be specified by attributes in the relationship structures for each of the NoSQL models presented in Sec. 3.1.

3.2.1. Key-Value Model

In the key-value model, a subject's attributes can be specified through the link relationships from level 2 notes to level 3 nodes, as illustrated in Fig. 7.

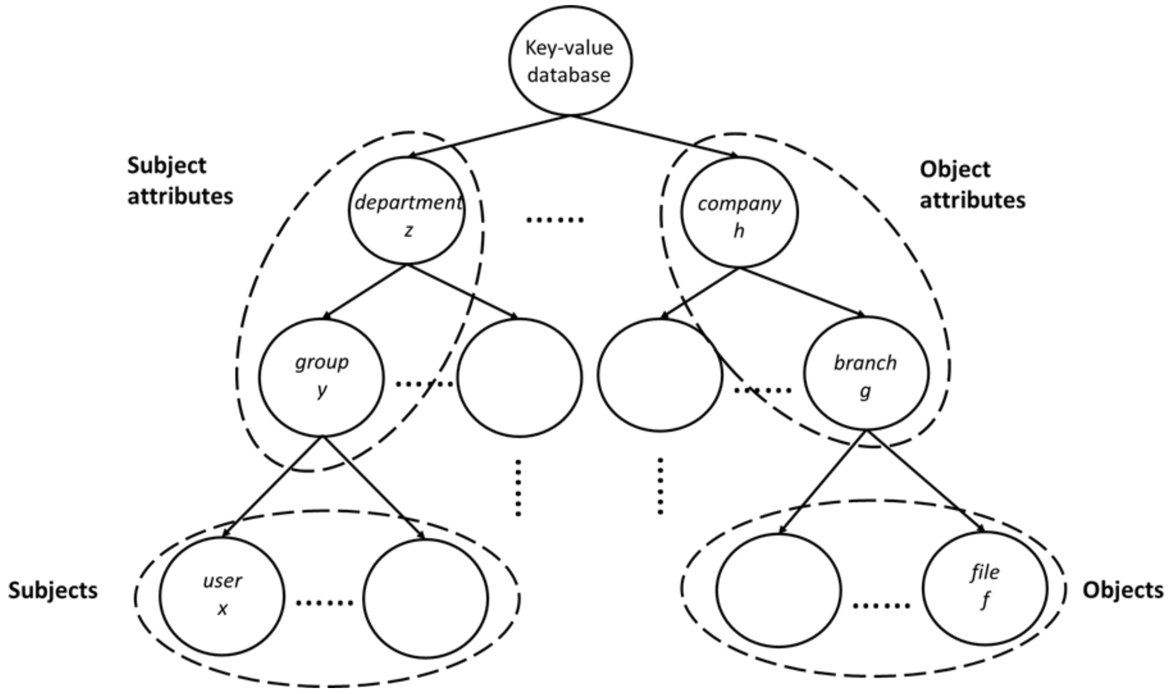

Fig. 7. Example of defining subject and object attributes for the key-value model

For example, a subject and its attributes can be specified as *x* (Value *user*), *y* (Key *group*), and *z* (Table *department*). Similarly, an object and its attributes can be specified as *f* (Value *file*), *g* (Key *branch*), and *h* (Table *company*). Based on these attributes, an access control policy rule can be created to state that "user *x* in group *y* of department *z* can read file *f* managed by branch *g* of company *h*" using the relationships of the nodes. It is not necessary to include all levels of nodes except for the leaf to form a policy rule, but at least one node is required for a rule, such as "user *x* in group *y* can access objects managed by branch *h*."

3.2.2. Wide-Column and Document Models

In wide-column and document models, the attributes of subjects can be specified through any link from level 2 through level 4 nodes or just one node, as shown in Fig. 8.

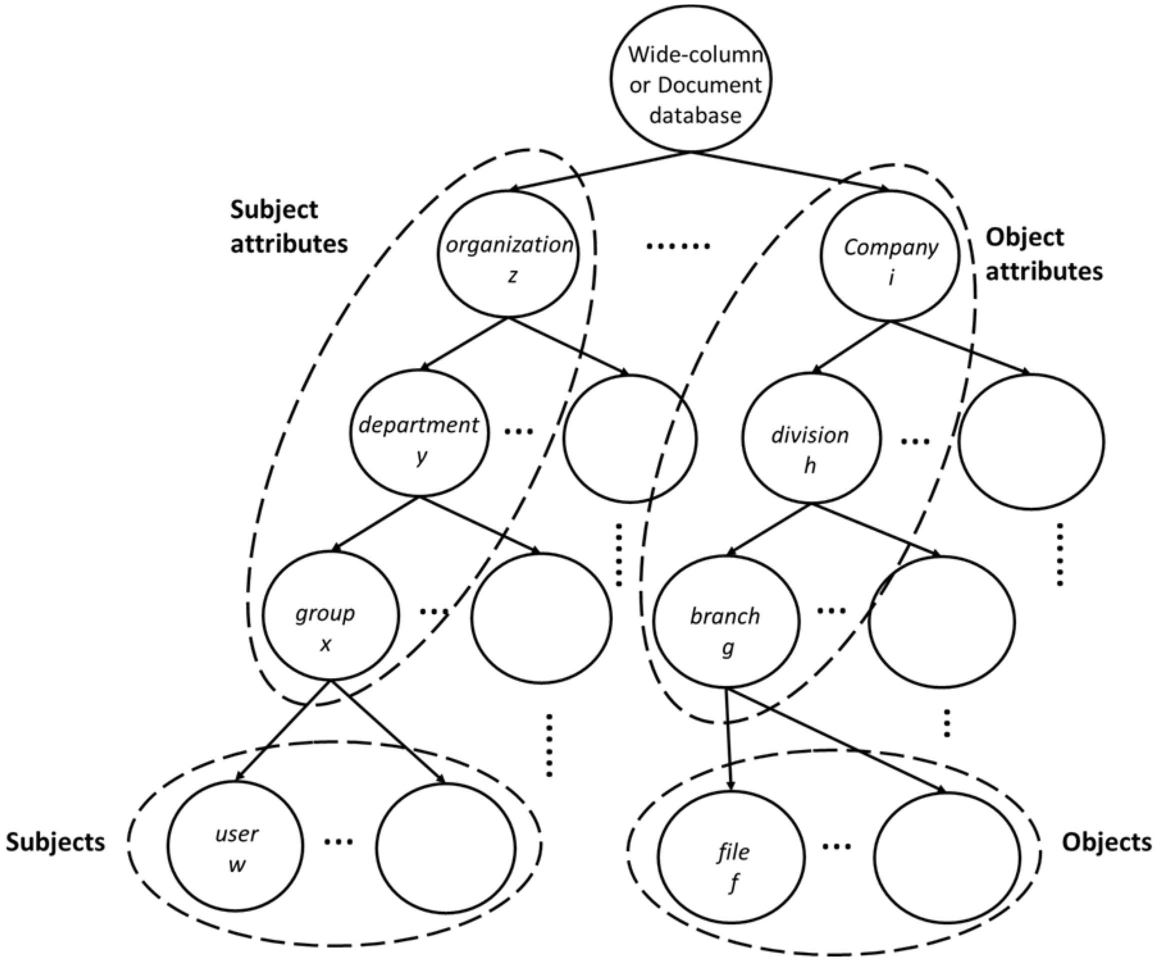

Fig. 8. Example of defining subject and object attributes for wide-column and document models

For example, a subject and its attributes can be specified as *w* (Value *user*), *x* (Column ID *group*), *y* (Row *department*), and *z* (Column family *organization*) for the wide-column model. Similarly, a subject and its attributes can be described as *w* (Value *user*), *x* (Field *group*), *y* (Document *department*), and *z* (Collection *company*) for the document model. An object and its attributes can be specified as *f* (Value *file*), g (Column-ID *branch*), *h* (Row *division*), and *j* (Column family *company*) for the wide-column model. For the Document model, an object and its attributes can be specified as *f* (Value *file*), *g* (Field *branch*), *h* (Document *division*), and *j* (Collection c*ompany*). Thus, access control based on attributes for the Wide-column model can be represented as follows: "The user *w* in group *x* of department *y* in company *z* can read file *f* managed by branch *g* that belongs to division *h* in company *j*." Similar access control rules can be specified for the document model. In both models, an access control rule can be specified using only one node (i.e., attribute) without linking to the next level of nodes. This implies that the lower-level attributes inherit access permissions from the attributes above them. For example, "Any users in group *x* can read objects that belong to division *h*."

In summary, an access control policy can be formally specified as a tuple with a finite number of elements in a set: {database, non-tree nodes of attributes for subjects, leaf node for the subject (optional), non-tree nodes of attributes for objects, leaf node for the object (optional), actions, permission} for key-value, wide-column, and document models. Due to the limitations of models based on node hierarchies, accessing data at the finest granularity level of the original data resources becomes a challenge. For example, in wide-column databases, data has to be organized within a collection in Column IDs. In document databases, data is typically stored within Fields. However, there may be situations in which finer-grained access control is required at a level beyond what the existing node structures can support, leading to a need for more attributes in access control rules. In such cases, it becomes necessary to construct tree branches horizontally to accommodate additional attributes. This approach adds complexity to the implementation of the database system and can be challenging to manage effectively. Such a lack of inherent support for fine-grained access control may require developers to find creative solutions to achieve the desired level of access granularity, potentially leading to more complex and less straightforward implementations. In addition to attributes for attribute-based access control, role-based access control (RBAC) can also be implemented by assigning a layer of nodes as roles.

3.2.3. Graph Model

Attributes in the graph model can be constructed differently since Graph edges do not necessarily have hierarchical relationships between nodes. Instead, edges connected by nodes may form cyclic relationships rather than a tree structure. As shown in Fig. 9, attributes in a graph database can be described through a sequence of edges and/or nodes without hierarchical relationships.

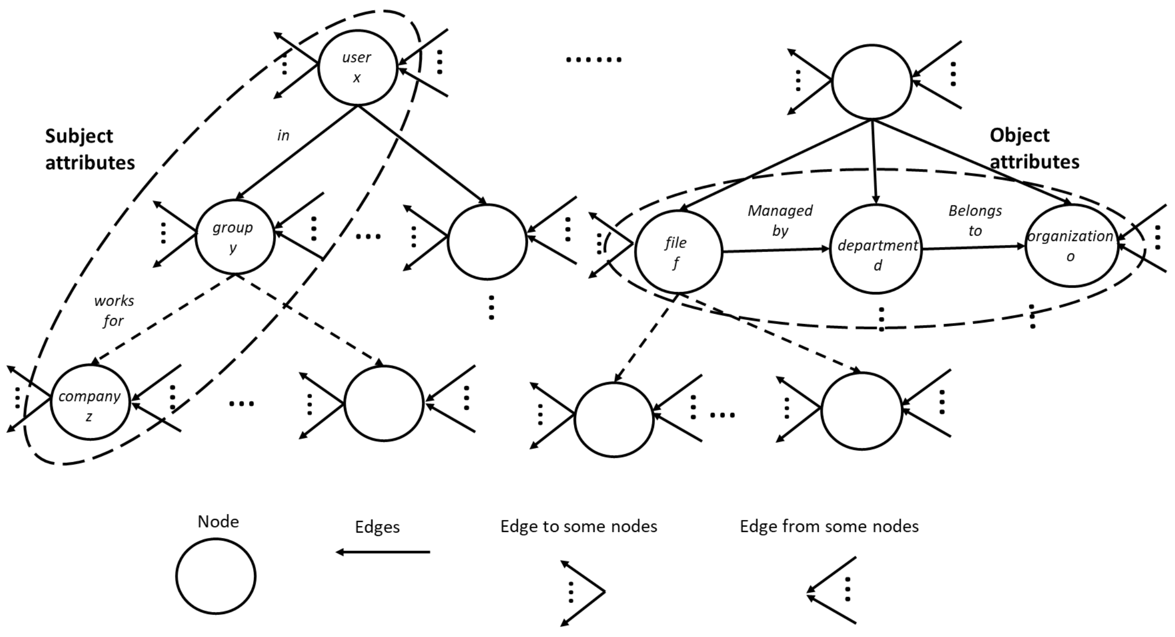

Fig. 9. Example of defining subject and object attributes for a graph model

For example, a subject attribute can be described as user *x* (node) in (edge) group *y* (node) works for (edge) company *z* (node), and an object attribute can be described as file *f* (node) managed by (edge) department *d* (node) belongs to (edge) organization *o* (node). An access control policy based on attributes can then be specified, such as "the user *x* in group *y* who works for company *z* can read file *f* managed by department *d* that belongs to organization *o*," or "any users in company *z* can read objects that belong to organization *o*" if only applied to attributes without specific subjects or objects.

Edges in the graph model are flexible and can be used to implement either an attribute or a permitted action in an access control rule. If implemented as a permitted action, the access control rule can be directly embedded in the database, treating a sequence of links as a rule. In other words, there is no need to store the access control rule outside of the database. An example graph in Fig. 10 contains the following nodes and edges: user *x* (node) in (edge directed to group *y* node) can read (edge directed to file *y* node).

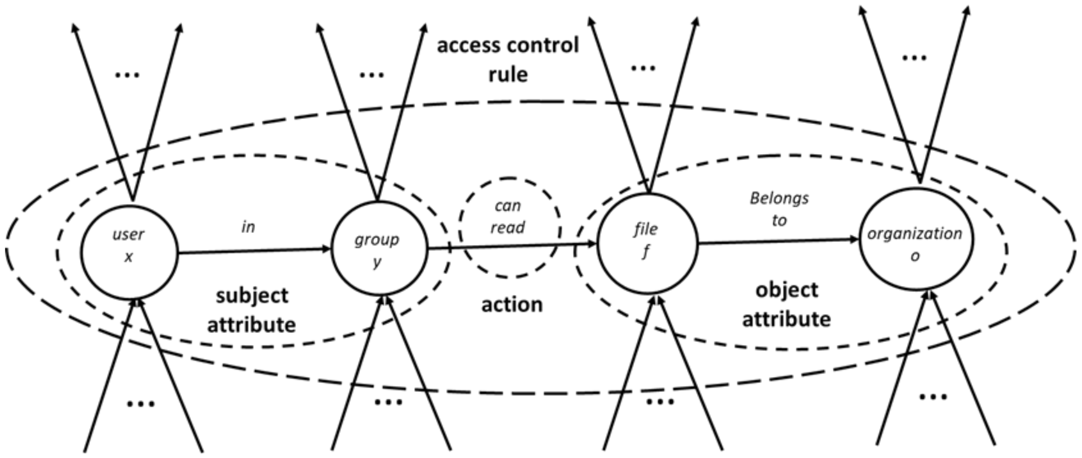

Fig. 10. Example of defining subject and object attributes for graph models using edges for permitted actions to embed an access control policy

Unlike the key-value, wide-column, and document models, the graph model does not have limitations on the number of edges and nodes connected in a sequence of relationships when constructing attributes for subjects and objects. The graph model allows access to data at the finest possible granularity that matches the original data resource thanks to the unlimited extension of the graph topology. Consequently, when composing access control rules, the graph model offers more flexibility and scalability but also introduces a more complex structure compared to the three other models. Additionally, RBAC can also be applied if any edges or nodes are assigned as roles instead of attributes. This flexibility in representing attributes and roles makes the graph model suitable for various access control scenarios. Appendix A describes an example application of this flexibility for implementations for a federated system environment.

3.3. Access Control Model Implementations

While mandatory access control policies (MAC) (e.g., ABAC [SP800-162] and RBAC [SF]), discretionary policies (DAC), and various context-based models [NISTIR7316] can be implemented in NoSQL databases using add-on applications, most current NoSQL implementations enforce authorization at a higher level. Specifically, authorization mechanisms are often implemented at a database level rather than at a more granular level [CF], such as the document collection level or column family level. Some NoSQL systems apply different enforcement mechanisms for different operation modes, such as read-only and read-write permissions being set for users in unshared modes but lacking support for authorization in shared modes [AA]. To support access control models, many research and commercial NoSQL databases propose or implement advanced enforcement mechanisms [CF] that vary from one system to another and may include:

- Modifying the format (schema) of the NoSQL structures according to the access control model
- Modifying the query methods according to the access control model [AA]
- Integrating an access control model in a NoSQL hierarchical level

Managing the heterogeneous, non-normalized, schemaless data of NoSQL databases characterized by complex hierarchical structures can make it difficult to handle the dynamic requirement of access control policies, such as conditional control (e.g., a rule regulating “no access to data that has a class greater than 10” or “no access for a name field that is empty”).

The graph model’s No-SQL structure of nodes and edges can leverage the pervasive capability of semantic content and the fluency of machine-understandable knowledge of Semantic Web technology. One important application is access control that manages federated resources in federated access control environments, as demonstrated in Appendix A.

4. Considerations of Access Control for NoSQL Systems

NoSQL systems can handle fast-paced agile development, the storage of large volumes of structured and semi-structured data, the requirements for scale-out architecture, and modern application paradigms (e.g., microservices and real-time streaming) [MO]. However, they also face many challenges with regard to access control, such as a lack of proper authentication, encryption, dynamic model support, and fine-grained authorization (as described in Sec. 3.3). Therefore, careful considerations need to be made when implementing access control on NoSQL databases to address these issues effectively.

4.1. Fine-Grained Access Control

Fine-grained access control (FGAC), which determines the scope of data authorized to users, plays a crucial role in access control mechanisms. However, due to the schemaless nature of most NoSQL databases, access control mechanisms are often implemented at a coarse-grained level. Unlike traditional RDBMS, where FGAC allows for the enforcement of access control at the cellular (e.g., row or column) level [FF], granular control is not available in current NoSQL databases. For example, some document NoSQL systems grant access control to the whole database or none [CJ]. This limitation hinders the ability to provide customized data protection levels, which could enhance the usability and expansion of these systems. While additional enforcement mechanisms can be implemented at various levels (e.g., a column family, column, or row), these solutions may not be easily adapted to other NoSQL databases.

To address these challenges, various proposed solutions include identifying suitable engineering approaches for encoding policies, defining an enforcement monitor, modifying the format of NoSQL structures, and adapting query methods according to access control models within a NoSQL hierarchical level. Despite ongoing research efforts, the integration of FGAC into NoSQL databases is still a work in progress [AA, FF].

4.2. Security

Due to NoSQL's distributed nature, security becomes a more significant concern compared to central management-oriented RDBMS. From the perspective of data at rest and data in transit (i.e., communications between databases), NoSQL databases generally provide weaker security features when compared to RDBMS.

For data at rest, some NoSQL databases may allow users backdoor access to other users' data due to poor logging and log analysis methods [ZA]. In some systems, authorization is applied at a per-database level using a role-based approach, limiting certain roles to specific privileges at different database levels. In contrast, some systems may provide no authorization by default, allowing any action by any user [CJ]. In such cases, an external security enforcement mechanism is essential. For example, some systems utilize metadata and provide management functions based on the database structure implementing access control operations with authorization principles. This allows different applications to implement their own access control.

The security of data in transit and communications in horizontally scaled NoSQL databases is a critical issue. Some systems use symmetric key algorithms to encrypt data based on the NoSQL structure (i.e., types). For example, in a key-value system, the value of the key is the encrypted data, including all of the other key values being concatenated and encrypted. When a query is created, the system retrieves data entities and then uses the symmetric key to decrypt the data [ZM]. As a result, the data is protected by encryption even if the key is intercepted while in transit between different NoSQL hosts.

Just like RDMBS, NoSQL databases are susceptible to injection attacks, which allow attackers to add malicious data to the NoSQL and cause unavailability and corrupt data. NoSQL injection attacks typically occur when the attack string is parsed, evaluated, or concatenated into a NoSQL API call. Attackers who are familiar with the syntax, data model, and underlying programming language of the target database can design specific exploits, especially in cases where server-side middleware (e.g., JavaScript, PHP) is heavily used to enhance database performance. For example, an internal operator "$where," designed to be used like the "where" clause in SQL, can accept sophisticated JavaScript functions to filter data. An attacker can exploit this by passing arbitrary code or commands into the $where operator as part of the query. The Open Worldwide Application Security Project (OWASP) Test Guide (v4) [OW] plans to include new procedures for testing NoSQL injections to assess NoSQL systems built upon JavaScript and/or PHP engines that may possess similar vulnerabilities [CJ].

NoSQL databases are designed to meet the requirements of the analytical world of big data with less emphasis on security during the design stage. Since they do not inherently embed security features in the system itself, developers must impose security mechanisms in the middleware using third-party tools without compromising scalability.

4.3. Query Language

There is currently no standard NoSQL query language in general or for a specific datastore category. Instead, each database adopts its own unique query language. This lack of standardization reduces interoperability among existing systems. For example, it is currently not possible to write even a basic query that can be executed within several different NoSQL systems. Similarly, data portability can be problematic since importing a dataset from one NoSQL database to another often requires preliminary data manipulation activities, even when dealing with the same data type (e.g., JSON objects). The heterogeneity of NoSQL databases and the diversity of their query languages make defining a general FGAC enforcement solution a complex task. Additionally, there is no systematic support for views, as in standard views for RDBMS, which further adds to the challenge of achieving uniformity and standardization across the NoSQL landscape [FF].

4.4. Data Consistency

Data consistency is essential for some access control models, especially dynamic models such as RBAC sections, separation of duty, workflow control, and n-person control [HA]. These models rely on a current and accurate access state maintained by consistent data to make access permission decisions. However, most NoSQL databases are designed using a shared base architecture across distributed commodity servers, which introduces the possibility of inherent data inconsistency among clustering nodes [CJ]. As a result, NoSQL databases do not always guarantee consistent results as not all participating commodity NoSQL servers may be entirely synchronized with other servers that hold the latest information. Additionally, if a single commodity server fails, it can lead to load imbalance among other commodity servers and affect data availability [ZA]. This lack of strong data consistency may explain why NoSQL databases have not been widely adopted to process critical financial transactions. Instead, it is often the responsibility of developers to design applications that can work with the eventual consistency model of NoSQL databases and to carefully weigh the trade-offs between data consistency and performance impacts [CJ].

4.5. Performance

Performance is key in choosing a NoSQL database, particularly when dealing with high volumes of data. However, security — including access control — is often a trade-off that impacts performance. Many NoSQL databases come with default security settings that are either set to none or minimum. Therefore, the most effective way to maintain performance while reducing risk is to deploy these databases in an environment where proper security measures and performance can be implemented and monitored.

4.6. Audit

Most currently distributed monitoring and reporting tools focus on database performance (e.g., information about the system's running state and connecting clients) with limited support for access auditing [CJ]. Approaches to implementing auditing functions in NoSQL databases may depend on the type of NoSQL database being used, which may have their own specific methods and tools for implementing access auditing and security logging. For example, in wide-column systems, auditing can be enabled on a per-node basis, allowing for maximum audit information by turning on auditing on every node of the system. It is important to consider the specific requirements and capabilities of the NoSQL database being used to ensure effective security logging and monitoring.

4.7. Environment Conditions Control

Properly managing and ensuring situational awareness is crucial for maintaining a secure and efficient NoSQL database environment. The environmental conditions can either be derived from the NoSQL database itself or provided by an independent outside source. If the environment conditions are derived from the NoSQL database itself, considering consistency (Sec. 4.4) and performance (Sec. 4.5) is critical. However, if the environment conditions are provided by an external source, addressing security issues (Sec. 4.2) is essential.

4.8. Support for AI

Formal documents (e.g., laws, statutes, regulations, memorandum, articles, etc.) are typically written in plain natural language (e.g., English) that may contain access control information. An artificial intelligence (AI) natural language processing (NLP) system can be used to render an access control policy by taking natural language documents as input and generating formal access control rules. To achieve this, the natural language contents must first be converted into a formal format. For example, "employees from the product division have to attend a security training course to work on project X" is translated into a formal format for an access control rule: "(user attribute = *employee* from *product division*) $\wedge$ (operation = *work*) $\wedge$ (resource attribute = *project X*) $\wedge$ (condition = *attended security training*) $\rightarrow$ (permission = *grant*)" connected by Boolean operators.

To make this transformation possible, the AI NLP system must:

a. Recognize if a sentence in the document can form access control rule(s)
b. Build a dictionary used by the NPL system
c. Define grammar for formal access control rules[1]

NoSQLs databases, especially graph systems, can parse and construct hierarchical order relationships between data, which is necessary for the AI analyst work in steps *a* and *b* above. They provide a data structure to support AI NLP systems in rendering access control policy from natural language documents and allow for more efficient and accurate translations of access control information from formal documents to formal access control rules.

4.9. Combine RDBMS

Considering the specific requirements of an application, it may be beneficial to deploy both RDBMS and NoSQL to process different data flows and achieve the optimal combined features of both types of databases. This hybrid approach can help leverage the strengths of each database model while addressing their respective limitations [CJ].

[1] In term of NLP processes, the requirements might go through the following: 1) sentence segmentation: generate sentences (a list of strings); 2) tokenization: generates tokenized sentences (a list of lists of strings); 3) part-of-speech tagging: generate post-tagged sentences (list of lists of tuples); 4) entity recognition: generated chunked sentences (list of trees); and 5) relationship recognition: generate relations (list of tuples).

5. Conclusion

NoSQL database systems offer promising features, such as flexible data models, horizontal scaling, and fast queries that allow for data analysis efficiency, improved system performance, ease of deployment, flexibility/scalability of data management, and users' availability when compared with RDBMS. However, despite these advantages, NoSQL databases lack a mechanism for handling and managing data consistency and maintaining integrity constraints. Additionally, they often have limited support for security at the database level. With an increasing number of people storing sensitive data in NoSQL databases, security issues are becoming critical concerns. Since access control is a fundamental data protection requirement for any database management system, this document discusses access control on NoSQL database systems by illustrating the NoSQL database types along with their support for access control models and describing considerations from the perspective of access control.

References

[AA] Alotaibi AA, Alotaibi RM, Hamza N (2019) Access Control Models in NoSQL Databases: An Overview, *JKAU: Comp. IT. Sci., Vol. 8 No. 1*, pp 1– 9. Available at https://marz.kau.edu.sa/Files/320/Researches/72524_45671.pdf

[AH] Ahmad N (2023) When to Use NoSQL Databases: NoSQL vs. SQL, *ServerWatc*h. Available at https://www.serverwatch.com/guides/when-to-use-nosql

[CF] Colombo P, Ferrari E (2021) Evaluating the effects of access control policies within NoSQL, *Future Generation Computer Systems*. https://doi.org/10.1016/j.future.2020.08.026

[CJ] Crawford J (2014) Current Data Security Issues of NoSQL Databases, *Fidelis Cybersecurity Volume 17*. Available at https://silo.tips/download/current-data-security-issues-of-nosql-databases

[FA] Fullstack Academy (2017) RDF Tutorial - An Introduction to the Resource Description Framework. Available at https://www.youtube.com/watch?v=zeYfT1cNKQg

[FF] Colombo P, Ferrari E. (2016) Fine-Grained Access Control Within NoSQL Document-Oriented Datastores, *Data Sci. Eng. 1,* pp 127–138. https://doi.org/10.1007/s41019-016-0015-z

[HI] hiteshreddy2181(2023) Types of NoSQL Databases, *Geeksforgeeks*. Available at https://www.geeksforgeeks.org/types-of-nosql-databases/

[HS] Hu VC, Quirolgico S, Scarfone K (2008) Access Control Policy Composition for Resource Federation Networks Using Semantic Web and Resource Description Framework (RDF), *Proceeding of 2008 International Computer Symposium*, pp 497-502 (v1). Available at https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=890067

[MO] MongoDB (2023) What is NoSQL? Available at https://www.mongodb.com/nosql-explained

[NISTIR7316] Hu VC, Ferraiolo DF, Kuhn DR (2006) Assessment of Access Control Systems. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency or Internal Report (IR) 7316. https://doi.org/10.6028/NIST.IR.7316

[NO] NoSQL (2020) Your Ultimate Guide to the Non-Relational Universe! Available at https://nextweb.gnoss.com/en/resource/nosql-databases-your-ultimate-guide-to-the-non--/479e0de7-8060-48d1-9670-bbdcd1df0610

[OG] Okman L, Gal-Oz N, Gonen Y, Gudes E, Abramov J (2011) Security Issues_in_NoSQL_Databases, *International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11*. https://doi.org/10.1109/TrustCom.2011.70

[OW] OWASP (2020) Web Security Testing Guide. Available at https://owasp.org/www-project-web-security-testing-guide

[SF] Sandhu R, Ferraiolo DF, Kuhn D.R (2000) The NIST Model for Role Based Access Control: Toward a Unified Standard, *5th ACM Workshop Role-Based Access Control.* pp 47–63. https://doi.org/10.1145/344287.344301

[SL] Simplilearn (2003) NoSQL Tutorial For Beginners | A Complete Guide To NoSQL Databases. Available at https://www.youtube.com/watch?v=aUPVpIYiLCc

[SP800-162] Hu VC, Ferraiolo DF, Kuhn DR, Schnitzer A, Sandlin K, Miller R, Scarfone KA (2014) Guide to Attribute Based Access Control (ABAC) Definition and Considerations. (National

Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-162, Includes updates as of August 02, 2019. https://doi.org/10.6028/NIST.SP.800-162

[WT] WeAreTechies (2022) NoSQL database | key value | document | wide column | graph stores. Available at https://www.youtube.com/watch?v=zG6CHYCx6ag

[ZA] Zaki AK (2014) NoSQL Databases: New Millennium Database for Big Data, Big Users, Cloud Computing and its Security Challenges, *International Journal of Research in Engineering and Technology* 3(3):403-409 (May). https://doi.org/10.15623/ijret.2014.0315080

[ZM] Zaki AK, Indiramma M (2015) A Novel Redis Security Extension for NoSQL Database Using Authentication and Encryption, *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. https://doi.org/10.1109/ICECCT.2015.7226101.

Appendix A.

A.1. Federation of Access Control

The availability of pervasive information will be greatly facilitated by the increase of globally distributed and interconnected information services. To support this global architecture, services that reside in groups of local networks (i.e., federations) interact with services that reside in other federations. All member federations form a federated network. To achieve networking in a global-computing framework, it is necessary to facilitate seamless access to federated services through inter-federation resource sharing and inter-trust between limited numbers of participating members of the global federation. However, the management of access control in a multi-organization global environment does not scale well. Since the shared resources of a federation are available both locally and conditionally globally, both the local and global access control policies are integrated under one static access control system so as to not violate the principles of the reference monitor. Therefore, it is challenging to 1) specify access control rules that manage the dynamic trust relations among federated parties, 2) separate local resource access control policy from global (federation) policy and risk the possible leaking of authorization, and 3) share the access control profile among federated members providing similar services.

The requirements for the interaction between global (or federation) and local access control policies are complex because most access control mechanisms and models are not flexible enough to arbitrarily combine and compose access control policies. Moreover, federated resources are distributed and shared by interoperating between three services:

1. Resource providers (RPs) store information for sharing with federated members. The information is managed locally by the resource contributors or administrators of the RP. The availability and integrity of the resource is the central operation goal.
2. Resource managers (RMs) are responsible for locating the resources in response to the access request from a resource consumer. The security and accessibility of communications between the RPs and their connected resource consumers are the primary concerns of an RM.
3. Resource consumers (RCs) are client applications that accept user requests for resources and forward those requests to an RM. Ideally, an RP should only have to communicate with one RM because the dissemination of shared resources is achieved by the RM. Only one connection between an RM and an RC is expected as well because the discovery of resource locations should be done by an RM, as illustrated in Fig. 11.

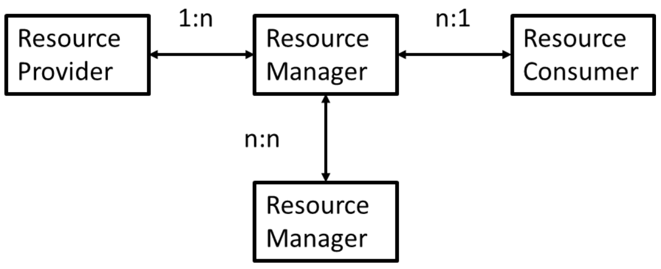

Fig. 11. Generic resource federation model

In reality, a federated community may be networked in a variety of architectures. The three basic services may be incorporated or simplified such that more than one service is managed or hosted in one physical system. However, these three services and their connections are assumed to be essential for any resource-sharing federation, and the resource-sharing protocols between them are composed by interlacing the following scenarios:

- Scenario 1: The information request from an RC is sent to an RM and then directly relayed to an RP without passing the request to other RMs or RPs.
- Scenario 2: A resource query cannot be satisfied by the connected RP, so the RM must collect and consolidate the partial results returned from more than one RP.
- Scenario 3: An RM does not have a direct or static connection to any RP that can provide the information as requested, so the resource discovery protocols need to be invoked to exchange information with other RMs that may have connections to other RPs with locations for the resources.

A.2. Graph NoSQL Implementation for AC Policies

To support accessibility and maintain the integrity of resource-sharing, access control policies between the three services are required such that a service has its own policy for the federation. Fig. 12 illustrates a generic scheme for a resource federation network and AC policies associated with each of the services.

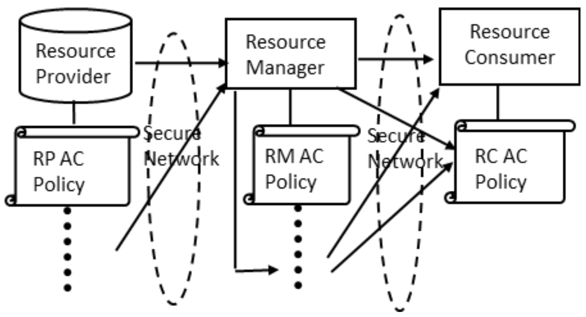

Fig. 12. Resource federation scheme

In addition to security between services in the lower-level communication mechanism (e.g., through a PKI infrastructure), support for the federation according to the access control policies posted by the services requires access control functions to be implemented. These functions manage and manipulate the types of enforcement rules listed below. Here, it is assumed that the policy for each service is maintained locally by the administration of the service.

- RP

  (p1) share (or conditional) rules

  (p2) non-share (or conditional) rules

- RM

  (m1) list of trusted RPs

  (m2) list of trusted RMs

  (m3) credibility rules for m1 and m2 (e.g., RP *A* has more credential than RP *B*)

  (m4) priority rules for m1 and m2 (e.g., RP *A* can be replaced by RP *B* — *A* “is a replacement” of *B*)

  (m5) reference rules (information from RP *A* is composed of information from RPs *B*, *C*, and *D* — *A* “should be supplemented by” *B*, *C*, and *D*)

  (m6) mediation rules (information from RP *A* cannot conflict with information from RP *B*)

- RC

  (c1) reference rules (similar to the reference information in RM except at the application level such as logic operations (AND, OR, XOR) between collected information)

  (c2) mediation rules (similar to the mediation information in RM except at the application level, such as data *a* from RM *X* cannot conflict with data *b* from RM *Y*)

  (c3) constraint rules (for RMs, such as no information older than 10 days can be trusted)

Rules p1, p2, m1, m2, and c3 contain resource availability information, while m3, m4, m5, m6, c1, and c2 contain information for trust management. Each rule is an access control policy assertion enforced upon two of the RPs, RMs, or RCs. Such a formal relationship can be annotated as members of a set that contains the binary relationships that the rule set is enforced upon: Rule *x* = {…… (S_X, S_Y) ….}, where S_X service is related to service S_Y by the enforcement of Rule *x*. For example, *Credential* = {…. (S_1, S_2) ….} says that the resource from RP S_1 has more credentials than RP S_2 and *Replace* = {…. (S_1, S_2) ….} says that the resource from RP S_1 should be requested if RP S_2 is not available. Thus, by conventional set operations, an access control trust management policy can be composed and combined through the Boolean or closure properties of the sets of trust management rules. A trust management rule can be expressed by a relationship pair (S_X, S_y) in a set that contains the type of rule such that the pair in the set Rule *x*, which are subject S_X and predicate Rule *x*, and object S_Y [HS] can be supported by the Graph NoSQL in the form of node S_X edge rule *x* node S_Y for an access control policy rule. For example, *Replace* = {…. (S_X, S_Y) ….} is translated into S_X and can replace S_Y of a triple in a graph NoSQL model.