Efficient Access Control of Large Scale RDF Data Using Prefix-Based Labeling

JINHYUN AHN¹ AND DONG-HYUK IM²

¹Department of Management Information Systems, Jeju National University, Jeju City 63243, South Korea
²Department of Computer and Information Engineering, Hoseo University, Asan 31066, South Korea

Corresponding author: Dong-Hyuk Im (dhim@hoseo.edu)

This work was supported in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program, supervised by the IITP (Institute for Information and Communications Technology Planning and Evaluation), under Grant IITP-2020-2018-0-01417, and in part by the Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education under Grant NRF-2018R1D1A1B07048380.

ABSTRACT A massive amount of resource description framework (RDF) data are available on the web. An RDF data publisher may want to prevent a few users from accessing a certain part of the RDF data. Various approaches have been proposed to reject a given SPARQL query that is intended to access instances or classes that are required to be protected. The problem of such access control management can be cast to processing ancestor/descendant relationship query over class hierarchy. The prefix-based labeling scheme has been applied to the fast processing of ancestor/descendant relationship queries. However, we observed that the existing approaches are ineffective in dealing with massive amounts of RDF data because the adopted labeling schemes produce labels of large sizes. Hence, we adopted the state-of-the-art MapReduce-based algorithm for prefix-based labeling to reduce the label size based on the structural information of RDF data. Experiments with real-world RDF datasets showed that the proposed approach is more efficient than the conventional methods.

INDEX TERMS Access control, RDF, prefix-based labeling.

I. INTRODUCTION A large volume of resource description framework (RDF) data from various domains is publicly available on the web, with some examples including YAGO [1], DBPedia [2], UniProtKB [3], GeoNames [4], and MusicBrainz [5]. RDF data are a set of triples composed of a subject, predicate, and object. The linked data community is a popular effort to maintain massive RDF data in a sharable manner.

The world wide web consortium (W3C) proposed an RDF-based access control system to control access to W3C files on its servers. RDF data have been used to manage access to files on W3C servers successfully. However, the access control is handled by underlying databases. Hence, several vendors of database management systems have begun to support managing RDF data, and this is known as RDF data management system.

RDF data management systems were proposed to support RDF data storage and to process SPARQL queries. RDF triples are retrieved from the stored RDF data that are found by matching with a given SPARQL query. Access control issues exist in terms of querying and inferencing against the RDF data. In the case of querying, an RDF data publisher may want to impose restrictions on a few classes to prevent certain users from accessing these instances. For example, probationary employees are not allowed to access salary information. Specifically, RDF data management systems can be designed to reject SPARQL queries of certain users who intend to access instances containing confidential information. In the case of inferencing data, we consider entailment rules that generate new RDF statements. We can evaluate if the newly generated RDF statements belong to a security classification based on RDF triple patterns or not. According to the security levels using the entailment rules, we can determine the unauthorized inferences.

We can extract the class hierarchy and instances from the RDF data. The class hierarchy can be represented in a directed acyclic graph (DAC) that is a directed graph with no cycle. To model a class hierarchy in a DAG, classes become nodes and super/subclass relationships become edges.
Thing can be designated as the root node. Directed edges are established from a class to its subclasses. Similarly, directed edges are established from a class to instances that belong to the class.

In our context, the access control of RDF data is to give a user the permissions to specific classes such that they are allowed or denied access to subclasses or instances of these classes. It is assumed that the permissions for classes are derived from their super classes. When a user submits a SPARQL query, the query is compared with a list of user permissions maintained in an authorization database to determine whether the query should be accepted or rejected. A SPARQL query is rejected if the superclasses of classes or instances mentioned in the query are matched with disallowed permissions. In summary, the problem we consider is defined as follows: given a SPARQL query for a RDF data and an authorization database, we have to answer if the query should be rejected or accepted by matching a class or instance in the RDF data and checking its permissions with respect to the authorization database. In this regard, we cast the problem of access control of RDF data to the problem that determines if a particular instance $I_i$ in the query belongs to a class $C_d$ in the disallowed permissions. Because a class hierarchy is viewed as a DAG, we can treat the query as an ancestor/descendant relationship query over a DAG.

An ancestor/descendant relationship query can be represented in a pair of nodes such as $(a, b)$. The simplest method for answering an ancestor/descendant relationship query is to traverse the DAG to verify if we can encompass $a$ to $b$. This approach requires visiting all of the intermediate nodes in a path from $a$ to $b$. It would be a time-consuming task if the two given nodes are located far from each other. To avoid visiting nodes individually, approaches have been proposed that utilize auxiliary data for each node. The auxiliary data are called an index or label. For two given nodes $a$ and $b$ in an ancestor/descendant relationship query, we can answer the ancestor/descendant relationships by applying operations to the labels of $a$ and $b$. The brute-force method is to store a list of all ancestor/descendant nodes in the label. Obviously, this approach is inefficient because it requires a huge space to store the labels of each node. To handle the label size issue, tree labeling schemes have been proposed [8]. Of these labeling schemes, we focus on the prefix-based labeling scheme that has been utilized in many commercial systems [9].

The contributions of the paper are as follows:

- We utilize the MapReduce-based algorithms for the prefix-based labeling scheme on XML data to solve the problem of the access control of RDF data.
- The proposed algorithm allows generating labels of smaller size compared to the previous approaches.
- Experiments were conducted to confirm that the proposed approach enables faster processing of access control queries.

This paper is organized as follows. Section II briefly overviews the related work on the access control of RDF data as well as the prefix-based labeling scheme. Our approach is demonstrated in section III. The experimental results are discussed in section IV. The paper concludes with section V.

II. RELATED WORKS

The access control of RDF data is to enable the RDF data management system to support RDF data security. [12] proposed the RAP framework that supports query-based access control over RDF storage. The RAP policy ontology is described to define rules that control actions. [13] proposed the concept-level access control of RDF data. The propagation policies and authorization conflict resolutions are described in terms of RDFS semantics. However, these approaches didn’t consider SPARQL queries. [7] proposed a security model that incorporates RDF and RDFS entailments. A two-level conflict resolution strategy was demonstrated to address inconsistencies that occurred because of entailment. Jena was used in the approach which is not scalable. [14] proposed the SBAS ontology consisting of subject-ontology (SO), object-ontology (OO), and action-ontology (AO), all of which are used to include the semantic relationships to perform inferences for access control. The semantic authorization flow is demonstrated in the concept, individual, and property levels. Based on the model, the authorization problem is reduced to the problem of subsumption. However, no implementation over a RDF store was proposed, which makes it difficult to see the scalability.

To the best of our knowledge, few works on index-based approach for the access control of RDF data exist. By the index-based approach, we indicate the one where additional data are exploited to reduce the time required to determine if the query given by the users is accepted or rejected. [15] is the only approach that is related to the issue. [15] proposed the subsumption hierarchy-based RDF authorization and adapted the prefix-based labeling scheme to support conflict resolution. Their approach is different from our approach in two folds. First, they focused on the conflict resolution of authorized RDF data. A set of authorization conflicts is detected using the inference types. Next, they adapted the straightforward prefix-based labeling scheme that requires a huge space. Our system adapts a novel algorithm for the prefix-based labeling algorithm that saves space.

Because our approach is based on the labeling schemes, we will briefly overview the existing works pertaining to the labeling schemes. The prefix-based labeling scheme has been utilized in many commercial systems [9] including the Microsoft SQL server [16] and SIREN [17]. It is noteworthy that SIREN is an information retrieval engine for RDF data that is a graph data model. In SIREN, RDF triples are converted into tree data structures by grouping the subjects. The prefix-based labeling scheme is adapted to index the tree. With the popularity of the prefix-based labeling scheme, [10],[11] proposed a MapReduce-based algorithm for prefix-based labeling of XML data to render it more applicable for massive real-world XML datasets (or linked open data if processed similar to the SIREN).
Implementing an in-memory algorithm for tree labeling would be straightforward. However, it would be difficult to implement an algorithm for tree labeling that can handle a massive XML file if the memory is not sufficient. The MapReduce framework has been adapted to tackle these issues. [10] proposed two MapReduce-based tree labeling algorithms that support interval-based labeling and the prefix-based labeling scheme. The map phase assigns start and end values to each element in the XML inputSplit. Because the XML file is distributed to a machine in a cluster, a case exists where no corresponding end/start tag is assigned. In that case, the end (or start) value remains empty. These incomplete labels are considered during the Reduce phase. The information collected during the Map phase is stored in the HDFS (Hadoop Distributed File System) file. We can transform incomplete labels into complete ones by referring to the HDFS file. [11] proposed a more efficient MapReduce-based tree labeling algorithm for the prefix-based labeling scheme. They proposed the dynamic compressed element labeling (DCL) technique to reduce the space requirements by adjusting the label assignment order based on the number of children.

III. THE PROPOSED APPROACH

In this section, we demonstrate the proposed approach that addresses the access control of RDF data. As mentioned in section I, we cast the problem to answering the ancestor/descendant relationship queries. Before explaining the proposed approach, we provide the definitions of the RDF data, permissions for access control, and XML data. RDF data are a set of triples as in definition 1.

Definition 1 (RDF Data): RDF data $T$ are a set of triples as follows:

$$T = \{(s, p, o) | (s, p, o) \in S \times P \times O\},$$

where $U$ represents a set of URI (Uniform Resource Identifier), $S$ is the union of $U$ and blank nodes, $P$ is the set of predicates, where a predicate is an URI that belongs to $U$, and $O$ is the union of $U$, blank nodes, and literals. $L$. s is typically called the subject, $p$ is called the predicate, and $o$ is called the object.

The access control of RDF data is to impose restrictions on the classes and instances. The class hierarchy and instances are to be extracted from the RDF data. We denote it as the hierarchy data as defined in definition 2.

Definition 2 (Hierarchy Data): Hierarchy data $S$ for RDF data $T$ represent a DAG $(V, E)$ where a set $V \subset U$ of vertices (nodes) and a set $E = E^C \cup E^I$ of edges, where

$$E^C = \{(cls_i, cls_j) | (cls_j, rdfs:subClassOf, cls_i) \in T\}$$

$$E^I = \{(cls, ins) | (ins, rdfs:type, cls) \in T\}$$

such that $cls_i, cls_j, cls, ins \in U$.

It is noteworthy that $E^C$ is a set of edges from a class to its direct subclasses and $E^I$ is a set of edges from a class to instances that belong to the class.

To focus on the performance of access control management based on the ancestor/descendant relationship query processing over hierarchy data, the simplified version of subsumption-based access control policy [13], [14] is used in our system, as defined in definition 3.

Definition 3 (Authorization): An authorization $A$ is imposed on a class in hierarchy data $S$, which is a triple $(user, access, cls)$, where user represents the user who wants to access the RDF data, access is either ALLOW (or GRANT) or DENY, and cls indicates a class.

We used a propagation rule stating that an authorization for a class $c$ is derived from the authorizations from superclasses of $c$. DENY overrides ALLOW when a conflict occurs. For example, if two authorizations exist, i.e., $(u1, ALLOW, c_p)$ and $(u1, DENY, c_a)$ where $c_a$ and $c_p$ are superclasses of $c$, we derive the authorization such that $(u1, DENY, c)$.

The prefix-based labeling scheme has been developed actively for trees [18]–[21]. As we handle hierarchy data, we focus on [22] who adapted the prefix-based labeling scheme for DAGs. Unlike a tree, a DAG is allowed to contain multiple parents. Thus, the prefix-based label for a node in a DAG is defined as a set of labels as defined in definitions 4 and 5.

Definition 4 (Prefix-Based Labeling Scheme for DAG): The prefix-based label $L(e)$ of a node $e$ in a DAG is a set of prefix-based labels derived from parents as follows:

$$L(e) = \{L(pa(e)).order(pa(e)) | pa(e) is a parent of e\},$$

where period(.) is the delimiter, and order(\(e\)) is the child order of $e$ under $pa(e)$. Throughout the paper, we denote $L(e)$ as a single label $L(pa(e)).order(e)$ if $e$ has a single parent and the context is clear. To indicate the suffix label order(\(e\)), we denote suffix($L(e)$).

Definition 5 (Label Size): The label size of a prefix-based label $L(e)$ is as follows:

$$size(L(e)) = \sum_{w \in L(e)} len(w),$$

where $len(w)$ is the number of digits in $w$ except for the delimiter(.)

Example 1: See Fig. 1. The root element $A$ has 1 as its prefix-based label. $L(E)$ is 1.2 because $L(A) = 1$ and $E$ is the second child corresponding to order($E$) = 2. $L(D)$ is [(1.1.2], [1.2.1}) because $D$ has two parents, $B$ and $E$. The label size of $L(D)$ is 6 = 3 + 3, while the label size of $L(E)$ is 2.

To achieve access control management over the RDF data, we transform the hierarchy data into XML data. See Fig. 1, a DAG shown in the left side can be written in a XML file in the right side.

Because $D$ has two parents $B$ and $E$ in hierarchy data, we write $D$ twice in the XML file. This simple transformation allows us to apply an algorithm for the prefix-based labeling of XML data to hierarchy data. During the labeling of the XML data, even if they have the same name $D$, they can be labeled separately because they are distinguished by offset
prefix-based labeling is performed. During access controlling instances are extracted from the RDF data over which the in Fig. 2. During index building (offline), class hierarchy and query processing time remain a challenge. Another line of approaches assign labels to each node in the order presented in a constant time. Unfortunately, previous approaches are inefficient at producing smaller label sizes. This renders it difficult for the prefix-based labeling scheme to handle massive RDF data. Previous approaches assign labels to each node in the order presented in the RDF file. [11] reported that if we change the label assignment order appropriately, the resultant prefix-based label size is reduced while conforming to the prefix-based labeling scheme. Reducing the label size is important because the query processing performance depends on the label size. The smaller the labels, the faster the ancestor/descendant relationships can be determined. Therefore, we decided to adapt [11] where the efficient and scalable algorithms of the prefix-based labeling scheme were proposed. The main point of our idea is to allow the system to deal with large amount of RDF data.

A. SYSTEM ARCHITECTURE
The architecture of the access control of RDF data is depicted in Fig. 2. During index building (offline), class hierarchy and instances are extracted from the RDF data over which the prefix-based labeling is performed. During access controlling (online), a query for access control is created by combining the permissions with classes or instances mentioned in a given SPARQL query. The ancestor/descendant relationship is verified against the prefix-based labels to determine the grant or deny for the query. The details are as follows.

B. INDEX BUILDING (Offline)
Given RDF data in the N-triple format, we create an XML file that contains the class hierarchy and instances as elements extracted from the RDF data. The procedure is described in Algorithm 1. The N-triple format serializes the RDF data in a triple in a line. It allows a triple to be processed in a triple manner without loading the whole RDF data in the primary memory. Triples having the predicate ` subclassof ` are considered to extract the class hierarchy. Likewise, triples having the predicate ` instanceof ` are considered to extract the instances of specific classes. From the extracted information, we write an XML file that has classes and instances as elements. The XML file is fed into a cluster that operates MapReduce processes. We adapt [11] to compute the prefix labels of each class and instance. This index building process is performed once for the RDF data. Once we obtain the prefix-labels of each class and instance, the access controlling task will be performed upon the prefix labels.

Algorithm 1 The Procedure of Index Building

```
1: procedure IndexBuilding(R) ↦ R is a RDF file in N-triple format
2: \[ L \leftarrow \emptyset \] \quad \triangleright \text{initialize an empty list}
3: for each triple \( T \) in \( R \) do
4: if \( T \)’s predicate is ` subclassof ` or ` instanceof ` then
5: \[ \text{add} \ T \text{ to } L \]
6: end if
7: end for
8: \( X \leftarrow \text{write an XML using } L \)
9: \( P \leftarrow \text{prefix-labeling}(X) \) \quad \triangleright \text{the algorithm is from [11]}
10: \( \text{return } P \)
11: end procedure
```

C. ACCESS CONTROLLING (Online)
The access controlling task verifies whether the given SPARQL query should be accepted or rejected based on the authorizations database and prefix-based labels obtained from the index building task. The procedure is described in Algorithm 2. First, classes or instances are extracted from a given SPARQL query. The access control query building module scans the authorizations database to identify the user permissions regarding the classes and instances extracted from the SPARQL query. Subsequently, we verify the ancestor/descendant relationships between classes and instances with respect to the RDF data, i.e., prefix-based labels that have been created during the index building task. Finally, we determine if the SPARQL query should be accepted or rejected.
Online (Access Controlling)

(Instance, Class) or (Class, Class)

Ancestor/Descendant Relationship Query Processing

Prefix-based Labels

Access Control Query Building

Instance, Class

(classes or instances)

Authorized

Grant/Deny

(User, SPARQL query)

Algorithm 2 The Procedure of Access Controlling

1: procedure AccessControlling(P, S, A) ⊲
2: \( P \) is the prefix-labeled XML file from Algorithm 1, \( S \) is a SPARQL query given by the user, \( A \) is the authorization database
3: \( C \leftarrow \) a list of classes and instances extracted from \( S \)
4: \( C' \leftarrow \) an access control query built based on \( C \) with respect to \( A \)
5: \( M \leftarrow \) matches for \( C' \) in \( P \)
6: if \( M \) is empty then
7: \( \text{return REJECT} \)
8: end if
9: for each match \( m \) in \( M \) do
10: if \( m.access \) is DENY then
11: \( \text{return REJECT} \)
12: end if
13: end for
14: \( \text{return ACCEPT} \)
15: end procedure

IV. PERFORMANCE STUDY

We performed experiments to demonstrate the effectiveness of the proposed approach (denoted herein as OUR). There is few work that adopt the prefix-based labeling method for access control of linked data. The access controlling issues are relatively new in the linked data community. We could find two competitors [15] and [23]. We implemented the state-of-the-art approach [15] that is based on the conventional prefix-labeling scheme described in [10] (denoted herein as Conv). We wanted to prove that the conventional prefix-based labeling schema is not applicable to deal with large scale RDF data. Another competitor was [23] that used Jena TDB to realize the access control of RDF data (denoted herein as TDB). In [23], ASK queries were used to verify whether the triple patterns in a given SPARQL query existed in the target RDF data. That is the reason why we choose Jena TDB even if it does not employee the prefix-based labeling schema. The main difference between access controlling over conventional database and RDF data is that RDF data has class hierarchy. The experiments were designed to compare performance by different levels of classes and instances. The experiments on varying parameters for MapReduce environment are not in the scope of the paper.

We tested against YAGO3, which is one of popular RDF data that includes Wikipedia and Wordnet [1]. We downloaded two datasets called yagoTaxonomy and yagoTypes in the TAXONOMY theme. yagoTaxonomy contains the class hierarchy and yagoTypes contains the instances that belong to the classes, which are fitted to our definition of hierarchy data in definition 2. We verified that no cyclic existed in the hierarchy. Fig. 3 and 4 show the number of classes and instances for each level in YAGO3, respectively.

The level indicates the number of edges from the root. The total number of unique classes and instances in YAGO3 is 5,696,833. Because classes and instances with multiple parents appear repeatedly in the hierarchy data, the total number
of nodes (i.e., classes and instances) in the hierarchy data from YAGO3 becomes 20,464,396.

Table 1 shows the space requirements for the prefix-based labeled YAGO3. It is trivial that TDB requires the largest space requirements as it has indices for processing complex SPARQL queries. The average label size is missing in TDB, because it is only meaningful in the prefix-based labeling scheme. As shown, the reduction by OUR is relatively larger than Conv, where the average label size is reduced by 0.14. Thus, the proposed approach (OUR) is more applicable to massive RDF data, because OUR can reduce the prefix-based label size. The improvement does not seem to be very significant. This is owing to the characteristics of the prefix-based labeling approaches. It is very hard to reduce the size of the prefix-based labels as they are generally represented as integers. Even though each node is successfully assigned smaller numbers than in the previous approaches, it sometimes consumes same amount of space. For example, the number 10 uses the same size as the number 99. The reduction in label size can also be examined by the number of parents for each class or instance in Fig. 5. More than half the nodes contain two or more parents.

To evaluate the access control of the query performance, we used a set A of authorizations and a set Q of queries. The parameters for configuring A and Q are as follows.

- $Q_i$: the level of instances that are included in a query set
- $Q_{\text{MAX}}$: the maximum number of queries in a query set
- $A_c$: the level of classes that are included in an authorization set
- $A_p$: the proportion of denied classes to all classes in a specific level $A_c$

For example, if we have an authorization set configured by $A_p = 0.1$ and $A_c = 4$, then randomly selected 10% of classes in level 4 are denied and 90% of classes in the same level are allowed. If we have a query set based on $Q_i = 6$ and $Q_{\text{MAX}} = 100$, then we have 100 queries, each of which consists of an instance from level 6. These 100 queries will be asked to an authorization set. This method allows us to reflect the characteristics of the prefix-based labeling scheme where labels are appended to the nodes’ label in the order from the root to the leaf. The label size will become larger as the node is located in the larger level in hierarchy. Indeed, the level is the same with the number of delimiters in a prefix-based label.

The combination of parameters are determined based on Fig. 3, 4, and 5 as follows: $Q_i = 4, 8, 16$, $Q_{\text{MAX}} = 1000$, $A_c = 4, 8, 16$, and $A_p = 0.5$. The access control management in our context operates as follows. When one of the queries in a query set is submitted, it is compared sequentially with every authorization in the authorization set. Whenever it finds an authorization indicating that the query should be rejected, the process stops. This means that different queries have different number of trials against an authorization set. The query processing time is measured by the time from submitting a query until receiving an indication such as ALLOW or DENY. We used the average time over 10 runs with the cold queries. It is noteworthy that if $Q_i < A_c$, then no queries will be rejected because every query belongs to the allowed classes.

Fig. 6 shows the experimental results for processing the access control queries for each method. The query processing time of TDB is omitted here because it was time consuming and could not be compared with OUR and Conv. Overall, OUR is more efficient than Conv. For the case when $A_c = 8$, the time scale is different from that of the other cases. This is because as the queries and authorizations are randomly selected, they have different number of trials to eventually determine a grant or deny. More time is required if it requires many trials irrespective of the parameters on the level of classes or instances. Therefore, to understand the results of the query processing time, we need to focus more on the relative comparison between OUR and Conv.

We observed that the smaller $Q_i$, the more that OUR can save the query processing time. This is closely related to the
average label size of classes and instances for each level, because it will consume more time when the label size is large. See Table 2, where the average label size of the classes and instances are shown. Because the nodes with multiple parents contain multiple labels, the average label sizes are complicated. Specifically, the query processing time is determined by the characteristics of both the authorizations (classes) and queries (instance). For instance, the difference between OUR and Conv for the class and instance is different. For the case of level 16, the difference is 0.1 for class and 0.3 for instance. The reason for the difference between OUR and Conv appears to be small is that each label itself is originally short consisting of around 4 to 16 integers. OUR demonstrated huge improvements because it saved some space for labels of each node comprising 4 to 16 integers.

V. CONCLUSION

We proposed a MapReduce-based approach for the access control of RDF data. Specifically, the prefix-based labeling scheme was utilized to handle massive RDF data. We adapted the state-of-the-art prefix-based labeling algorithm to handle massive RDF data. We demonstrated the method to utilize the prefix-based labeling scheme for the access control of RDF data by building an index for processing subsumption-based access control over the class hierarchy. The access control query performance was evaluated over massive RDF data YAGO3 that showed better performance than the previous methods (based on an existing triple store and conventional prefix-based labeling algorithm). The experimental results confirm that the proposed approach is more applicable to massive RDF data. In the future work, larger RDF datasets will be incorporated, and an approach that encodes the prefix-based labels into more efficient IDs, such as hexadecimal representations, will be considered. We will adopt another labeling scheme to address the problem of RDF data access control, such as prime number labeling [24]–[27], interval-based labeling [28], [29], and 2-hop labeling [30], [31]. Furthermore, RDF data changes over time. Handling updated RDF data will be considered in terms of change detection [32]–[34] and version management framework [35]. In this paper, we only consider subclassof and instanceof predicates. Although the proposed method can be extended easily to other predicates, it needs to be verified with extensive experiments.

REFERENCES

[1] F. Mahdisoltani, J. Biega, and F. M. Suchanek, “YAGO3: A knowledge base from multilingual Wikipedias,” in Proc. Biennial Conf. Innov. Data Syst. Res. (CIDR), 2013, pp. 1–12.
[2] J. Lehmann, R. Isle, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, “DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia,” Semantic Web, vol. 6, no. 2, pp. 167–195, 2015.
[3] F. Jungo, L. Bougueleret, I. Xenarios, and S. Poux, “The UniProtKB/swiss-prot tox-prot program: A central hub of integrated venom protein data,” Toxicon, vol. 60, no. 4, pp. 551–557, Sep. 2012.
[4] D. Ahlers, “Assessment of the accuracy of GeoNames gazetteer data,” in Proc. 7th Workshop Geographic Inf. Retr. (GIR), 2013, pp. 74–81.
[5] C. Mesnage, “Using the Musicbrainz database in the classroom,” in Proc. Int. Conf. Inf., Commun. Technol. Educ. (ICICTE), 2017, pp. 338–344.
[6] W3C ACL System. Accessed: May 1, 2020. [Online]. Available: http://www.w3.org/2001/04/20-ACLs
[7] A. Jain and C. Farkas, “Secure resource description framework: An access control model,” in Proc. 11th ACM Symp. Access Control Models Technol. (SACMAT). New York, NY, USA: ACM, 2006, pp. 121–129.
[8] A. Klaib and L. Joan, “Investigation into indexing XML data techniques,” in Proc. Int. Conf. Internet Comput. Big Data, 2014, pp. 83–89.
[9] P. PaI, I. Cseri, O. Seeliger, M. Rys, G. Schaller, W. Yu, D. Tomic, A. Baras, B. Berg, D. Churin, and E. Kogan, “XQuery implementation in a relational database system,” in Proc. 31st Int. Conf. Very Large Data Bases, 2005, pp. 1175–1186.
H. Choi, K.-H. Lee, and Y.-J. Lee, “Parallel labeling of massive XML data with MapReduce,” J. Supercomput., vol. 67, no. 2, pp. 408–437, Feb. 2014.

J. Ahn, D.-H. Im, and H.-G. Kim, “A MapReduce-based approach for prefix-based labeling of large XML data,” in Proc. Joint Int. Semantic Technol. Conf., 2016, pp. 83–98.

P. Reddivari, T. Finin, and A. Joshi, “Policy-based access control for an rdf store,” in Proc. Workshop Semantic Web Collaborative Knowl. Acquisition (IJCAI), 2007, pp. 1–10.

L. Qin and V. Atluri, “Concept-level access control for the semantic Web,” in Proc. ACM Workshop XML Secur. (XMLSEC). New York, NY, USA: ACM, 2003, pp. 94–103.

S. Javanmardi, A. Amini, and R. Jalili, “An access control model for protecting semantic Web resources,” in Proc. 2nd Int. Semantic Web Policy Workshop. New York, NY, USA: Citeseer, 2006, pp. 1–15.

J. Kim and S. Park, “Efficient authorization conflict detection using graph labeling in controlling access to RDF,” Comput. Standards Interface, vol. 38, pp. 51–63, Feb. 2015.

P. O’Neill, E. O’Neill, S. PaI, I. Cseri, G. Schaller, and N. Westbury, “ORDPATHS: Insert-friendly XML node labels,” in Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD), 2004, pp. 903–908.

R. Debru, N. Toukipov, M. Catasta, and G. Tummarello, “A node indexing scheme for Web entity retrieval,” in Proc. Extended Semantic Web Conf., 2010, pp. 240–256.

L. Xu, T. W. Ling, H. Wu, and Z. Bao, “DDE: From Dewey to a fully dynamic XML labeling scheme,” in Proc. 35th SIGMOD Int. Conf. Manage. Data (SIGMOD), 2009, pp. 719–730.

I. Tatarenov, S. D. Viglas, K. Beyer, J. Shammugasundaram, E. S. Shikita, and C. Zhang, “Storing and querying ordered XML using a relational database system,” in Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD). New York, NY, USA: ACM, 2002, pp. 204–215.

R.-R. Lin, Y.-H. Chang, and K.-M. Chao, “A compact and efficient labeling scheme for XML documents,” in Database Systems for Advanced Applications. Cham, Switzerland: Springer, 2013, pp. 269–283.

J. Lu, X. Meng, and T. W. Ling, “Indexing and querying XML using extended Dewey labeling scheme,” Data Knowl. Eng., vol. 70, no. 1, pp. 35–59, Jan. 2011.

V. Christophides, G. Karvounarakis, D. Plexousakis, M. Scholl, and S. Tourtounis, “Optimizing taxonomic semantic Web queries using labeling schemes,” J. Web Semantics, vol. 1, no. 2, pp. 207–228, Feb. 2004.

S. Kirrane, A. Abdelrahman, A. Mileo, and S. Decker, “Secure manipulation of linked data,” in Proc. 12th Int. Semantic Web Conf.-1 (ISWC). New York, NY, USA: Springer-Verlag, 2013, pp. 248–263.

J. Ahn, T. Lee, and D.-H. Im, “Efficiently answering reachability queries for tree-structured data in repetitive prime number labeling schemes,” Appl. Sci., vol. 8, no. 5, p. 785, May 2018.

G. Wu, K. Zhang, C. Liu, and J. Li, “Adapting prime number labeling scheme for directed acyclic graphs,” in Database Systems for Advanced Applications. Cham, Switzerland: Springer, 2006, pp. 787–796.

S. Morozov, H. Saiedian, and H. Wang, “Reusable prime number labeling scheme for hierarchical data representation in relational databases,” J. Comput. Inf. Technol., vol. 22, no. 1, p. 31, 2014.

X. Wu, M. L. Lee, and W. Hsu, “A prime number labeling scheme for dynamic ordered XML trees,” in Proc. ICDE, 2004, pp. 66–78.

C. Li and T. W. Ling, “QED: A novel quaternary encoding to completely avoid re-labeling in XML updates,” in Proc. 14th ACM Int. Conf. Inf. Knowl. Manage. (CIKM), Oct. 2005, pp. 501–508.

L. Xu, Z. Bao, and T. W. Ling, “A dynamic labeling scheme using vectors,” in Database and Expert Systems Applications. Cham, Switzerland: Springer, 2007.

E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, “Reachability and distance queries via 2-hop labels,” SIAM J. Comput., vol. 32, no. 5, pp. 1338–1355, Jan. 2003.

H. Wei, J. X. Yu, C. Lu, and R. Jin, “Reachability querying: An independent permutation labeling approach,” VLDB J., vol. 27, no. 1, pp. 1–26, Feb. 2018.

D.-H. Im, S.-W. Lee, and H.-J. Kim, “Backward inference and pruning for RDF change detection using RDBMBS,” J. Inf. Sci., vol. 39, no. 2, pp. 238–255, Apr. 2013.

T. Lee, D.-H. Im, and J. Won, “Similarity-based change detection for RDF in MapReduce,” Procedia Comput. Sci., vol. 91, pp. 789–797, Jan. 2016.

J. Ahn, D.-H. Im, J.-H. Eom, N. Zong, and H.-G. Kim, “G-DIFF: A grouping algorithm for RDF change detection on MapReduce,” in Proc. Joint Int. Semantic Technol. Conf. Cham, Switzerland: Springer, 2014, pp. 230–235.

D.-H. Im, S.-W. Lee, and H.-J. Kim, “A version management framework for RDF triple stores,” Int. J. Softw. Eng. Knowl. Eng., vol. 22, no. 1, pp. 85–106, Feb. 2012.