Scientia Iranica B (2015) 22(6), 2178–2187

SCIENTIA IRANICA

Sharif University of Technology
Scientia Iranica
Transactions B: Mechanical Engineering
www.scientiairanica.com

Model for independent use of web ontology storage of SPARQL-to-SQL translation algorithms

J. Son[a,*], J.-D. Kim[a] and D.-K. Baik[b]

a. *Department of Computer and Radio Communications Engineering, Korea University, Seoul, Republic of Korea.*
b. *Graduate School of Convergence IT, Korea University, Seoul, Republic of Korea.*

Received 22 May 2014; received in revised form 26 January 2015; accepted 15 September 2015

KEYWORDS
Semantic web;
Web ontology;
Query translation;
Relational database;
SPARQL;
Relational view table.

Abstract. This paper proposes a storage-independent model for SPARQL-to-SQL translation algorithms based on a relational view. In the development of Web ontology research, the translation from SPARQL to SQL continues to be an issue. Previous research has focused on an efficient and complete translation from SPARQL queries to equivalent SQL queries. However, these translation algorithms depend on specific storage structures. When we modify the storage structure, the translation algorithm should also be modified to suit the changed storage structure. This has motivated study of the issue of a model for the independent use of storage structures by algorithms. These can then guarantee independence between translation algorithms and storages by generating relational views, and improve the application and usability of the translation algorithm. In addition, this paper presents experiment results showing the accuracy and no data loss rate of query results for different storages.

1. Introduction

Continuous and rapid increase in the amount of current Web information has made it difficult for users to extract desired information. The Semantic Web has been proposed as a solution to this problem [1,2]. In the Semantic Web environment, query languages for information retrieval of ontologies have been proposed [3-5]. Among these, SPARQL, recommended by W3C, is the most representative description language [5]. Further, many Web ontology storage systems have been developed on the basis of a relational database (RDB) for efficient data management [6-15]. However, there are several issues related to the use of RDB and SPARQL. The appropriate query language for an RDB is SQL, and, hence, it is necessary to develop a SPARQL-to-SQL translation algorithm for efficient data retrieval from an RDB using SPARQL. Common representative SPARQL-to-SQL translation algorithms include the Chebotko algorithm [16], sparql2sql over Jena [17], the Harris algorithm [18], and so on [19-24].

However, the algorithms proposed are dependent on specific storage models. If the storage structure is changed, the corresponding translation algorithm should be modified accordingly. This approach also causes some problems:

1. Making the original characteristics (advantages) of the storage weak or lost;
2. Interoperability difficulty with various relevant modules (e.g., inference engine, parser, etc.).

As a result, the cost associated with the modification and verification of the translation algorithm can be extremely high.

To resolve this issue, we have proposed a model for the independent use of storage structures of a

*. *Corresponding author. Tel.: +822 925 3706; Fax: +82 2 921 9137*
E-mail addresses: redfunky07@korea.ac.kr (J. Son); kjd4u@korea.ac.k (J.-D. Kim); baikdk@korea.ac.kr (D.-K. Baik)

SPARQL-to-SQL translation algorithm [25]. However, the model only supports triple structure storage. Therefore, in this paper, we extend the previous model used for various structure storages for Web ontology. Likewise, the proposed model uses a relational view on RDB. To evaluate the proposed model, this paper implements a prototype and describes the experiment. We describe the experimental results that clearly show the predominance of the proposed model.

The paper is organized as follows. Section 2 describes previous SPARQL-to-SQL translation algorithms and Web ontology storage based on RDB. Section 3 provides a comparison between the previous approach and our approach, including the proposed model. Section 4 describes our experiments to verify the proposed model, for which we measure accuracy and data loss rates by query results as evaluation items. We describe the comparison performance evaluation conducted between a dependent model and the proposed model in Section 5, and, finally, conclusions and future study plans are summarized in Section 6.

2. Related works

2.1. Previous SPARQL-to-SQL translation algorithms

Several SPARQL-to-SQL translation algorithms have been published [16-24], and the Chebotko algorithm [16], sparql2sql over Jena [17], and the Harris algorithm [18], which are the most representative, are described as follows.

To the best of our knowledge, the facts described below regarding the algorithms are correct at present. the Chebotko algorithm offers the maximum number of functions for translation of the OPTIONAL clause of a SPARQL query and uses the simplest triple storage structure compared to all others available [14]. In addition, this algorithm proposed a novel relational operator. It shows better performance than a general left outer join operator. Therefore, the Chebotko algorithm [16] supports a translation of semantic preserving, and provides a more efficient translation of SQL queries.

sparql2sql [17] is an algorithm in a query engine for SPARQL over Jena triple stores. It rewrites a SPARQL query into an SQL query. This approach offloads most of the query execution work on the database. The Harris algorithm [18] supports translation using a simple optional graph pattern using relational algebra. However, both sparql2sql and the Harris algorithm still have problems. Both algorithms are limited to translating a nested OPTIONAL clause to its corresponding SQL query. The translated SQL query suffers from inefficiency.

Most of all, all three algorithms depend on specific storage models. sparql2sql, the Harris algorithm, and the Chebotko algorithm are based on Jena storage, 3store, and TRIPLES, respectively. Therefore, we must modify either the translation algorithm or the given storage structure, which causes the low usability and applicability of the algorithm.

2.2. Web ontology storage based on RDB

Currently, Web ontology storages based on RDB are being developed for the efficient storage and management of enormous amounts of Web ontology data. Several Web ontology storages based on RDB have been proposed, such as Jena [6], Sesame [7], OWLJessKB [8], DLDB [9], and so on [10-15]. Jena is a Java framework for building Semantic Web applications [5]. In essence, Jena stores data in a triple structure. The *jena_gntn_stmt* table stores all data less than 256 bytes. However, in cases when that data length is more than 256 bytes, the data is managed in separate tables: Literal, URI and Prefix are stored in *jena_long_lit*, *jena_long_uri* and *jena_long_prefix*, respectively. Sesame provides a storage model based on RDB for the Resource Description Framework (RDF) [26] and Resource Description Framework Schema (RDF-S) [27]. The class and property tables in Sesame store the respective information of class and property in an ontology document. Sesame generates the *subClassOf* table to represents hierarchy among classes. The *subPropertyOf* table contains hierarchy information among properties. The relationship information on class, class instance, and property is stored in the table, *TRIPLES*. OWLJessKB is a memory-based reasoning tool that can be used with ontology specified in OWL. OWLJessKB uses an RDB to store service descriptions, with JESS as the inference engine. DLDB is a knowledge base system that extends an RDB management system with additional capabilities for DAML+OIL inference.

3. Proposed model

In this section, we describe a comparison made between the previous approach and our approach, which includes the proposed model. The overall process of the two approaches is as follows. First, a user inputs a SPARQL query. The input SPARQL query is translated by a SPARQL-to-SQL translation algorithm. The translated SQL is transmitted on RDB, and the query result is transmitted to the user. When SPARQL is translated to SQL; the SQL query essentially requires a table name for information retrieval. This table name is represented in a FROM clause by the SQL syntax. To do this, an algorithm developer should recognize a storage schema structure. Therefore, previous proposed translation algorithms depend upon a storage schema structure.

Previous approaches require the same number of translation algorithms as the number of storage,

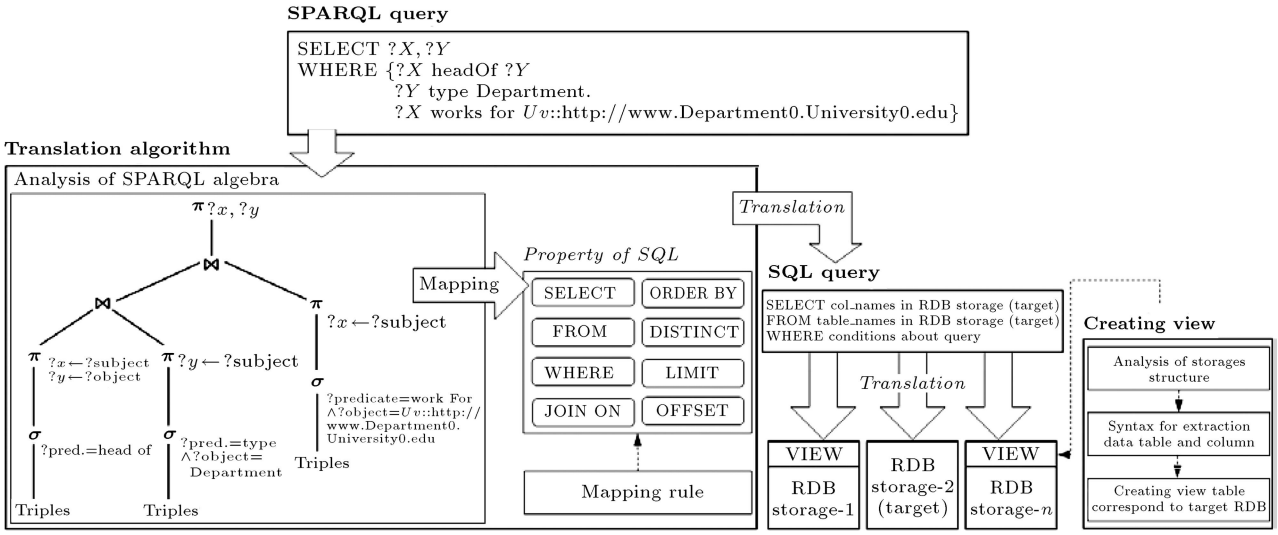

Figure 1. Proposed storage-independent model based on relational view table.

because the translation algorithm depends on the storage structure. For example, if each of ten storages has a different schema structure, the SPARQL-to-SQL translation algorithm should develop ten algorithms. In addition, if the storage structure is changed, the translation algorithm should be modified to be suitable for a new storage structure. Therefore, the development, modification, and verification of additional translation algorithms require much time and cost.

To solve the aforementioned issue, we propose a storage-independent model for the translation algorithm based on a relational view table. Figure 1 shows the proposed model for our approach. We use a simple SPARQL query example, using a LUBM query, which is provided by Lehigh University [28]. This SPARQL query is transmitted to the translation algorithm. The proposed model requires only one translation algorithm, regardless of storage number and structure. However, let us assume that the translation algorithm will target any specific RDB structure (e.g., RDB storage-2). This algorithm analyzes SPARQL algebra, and then each SPARQL property is mapped with its equivalent SQL property by a mapping rule. The query translates the SQL query to the suitable target, RDB storage-2. SQL The query retrieves the query results directly, subsequent to transmission to the target RDB storage-2. In Figure 1, if we have several N-storages, the other storages with different structures (e.g., RDB storage-1, RDB storage-3, RDB storage-n), with the exception of the target RDB, retrieve the query results through view tables. To create the view table, first, the proposed model analyzes each storage schema structure. The next step selects the tables and columns that correspond to the target RDB storage-2. The selected tables maintain the relationship between the other tables. Finally, the view table is generated with the same structure as the target RDB storage-2. Using the view table, the proposed model can use storage independently. In addition, creation of the view table is simpler than developing the algorithm. The view table is generated by not demanding the development capability of high-level SQL syntax, but of a simple SQL syntax. Therefore, the proposed model can save the time and cost required for the development, modification, and verification of the algorithm.

4. Experiment for proposed model

We implemented a prototype for evaluation of the proposed model, and, in this paper, we conduct an experiment to verify our proposed model. To do this, we measure accuracy and data loss rate by using query results as the evaluation items. The experiment process is as follows. First, we define several types of query using LUBM queries [28], and select any specific algorithm for SPARQL-to-SQL translation. In this experiment, we use the Chebotko algorithm, because this algorithm supports more powerful functionality for SPARQL-to-SQL translation compared to other algorithms. The Chebotko algorithm depends on TRIPLE storage, and, therefore, our target storage is TRIPLES storage. The other storages for the experiment are Jena storage and Sesame storage. They generate a relational view that corresponds to the structure of the TRIPLES storage. We confirm the translated SQL statements using the Chebotko algorithm, which are equivalent to the defined SPARQL queries. Finally, we compare the correct answers and the results obtained from the three storages, which are TRIPLES storage, Jena storage, and Sesame storage. We generate an OWL dataset using UBA, which is provided by Lehigh University. As listed in Table 1, we use 14 LUBM queries for our experiment. A set of original LUBM queries requires

Table 1. Experiment test queries.

Query	Query description
Q1	All Graduate Students who take Graduate Course 0 in Department 0 of University 0
Q2	All Graduate Students who are now studying at the university from which they obtained their bachelor degrees
Q3	All publications of Assistant Professor 0 in Department 0 of University 0
Q4	All Assistant Professors in Department 0 of University 0 and their email addresses and telephone numbers
Q5	All Undergraduate Students in Department 0 at University 0
Q6	All Graduate Students
Q7	All Graduate Students who take the Graduate Course of Associate Professor 0 in Department 0 of University 0
Q8	All Undergraduate Students of University 0 and their email addresses
Q9	All Undergraduate Students who take those courses whose advisor is Full Professor
Q10	All Graduate Students who take Graduate Course 0 in Department 0 of University 0
Q11	All research groups in Department 0 of University 0
Q12	All department heads of University 0
Q13	All Assistant Professors who have master degrees from University 0
Q14	All Undergraduate Students of University 0

an inference support that can infer relationships between OWL data. However, the main purpose of this experiment is not reasoning, but correct SPARQL-to-SQL translation. Therefore, the SPARQL queries for Q4 to Q10, Q12, and Q13 are modified. Reasoning is not necessary to execute the modified queries.

4.1. Experiment results and evaluation

Table 2 lists all the experiment results from 14 test queries. The *Correct Answer* is the true value, with respect to the result of the 14 queries. By comparing the query results between the three storages, we evaluate whether data loss occurs, and whether the value is true. Table 2 indicates that the number of all query results with the *Correct Answer*; *TRIPLES* storage as target, and *Jena storage* and *Sesame storage* as the view tables, are the same. Furthermore, the values of the retrieved results are the same. Consequently, although the three storages have different storage structures, we can guarantee the same results without loss of query results from the three storages that use a relational view table.

5. Performance evaluation

This section evaluates the previous storage-dependent and storage-independent models. We describe prelimi-

Table 2. Result of 14 queries.

Query	Result of query (number)				Accuracy (%) (data loss rate (%))
	Correct answer	TRIPLES storage (target)	Jena storage (view)	Sesame storage (view)	
Q1	4	4	4	4	100% (0%)
Q2	0	0	0	0	100% (0%)
Q3	6	6	6	6	100% (0%)
Q4	40	40	40	40	100% (0%)
Q5	532	532	532	532	100% (0%)
Q6	1874	1874	1874	1874	100% (0%)
Q7	16	16	16	16	100% (0%)
Q8	17748	17748	17748	17748	100% (0%)
Q9	90	90	90	90	100% (0%)
Q10	4	4	4	4	100% (0%)
Q11	10	10	10	10	100% (0%)
Q12	30	30	30	30	100% (0%)
Q13	1	1	1	1	100% (0%)
Q14	5916	5916	5916	5916	100% (0%)

nary constraints, factors, and notations for the models, and, in addition, the comparison evaluation between the two models.

5.1. Comparative items and assumptions

The comparative item of the two models uses two methods. The first method compares the performance of the SPARQL-to-SQL algorithm development time with the performance of the view generation time. The second method compares the performance of the query translation time with the performance of the query transmission time. However, in the case of the first method, the performance of the SPARQL-to-SQL algorithm development time generally has a higher performance compared with that of the view generation time. Therefore, this paper measures performance using the second method. The query translation time is the time required for that SPARQL to be translated to SQL through the translation algorithm. The query transmission time is the time required for that generated SQL to be executed in storages.

The main factors for our evaluation are summarized as follows:

- Comparative targets: Storage-independent model and storage-dependent model;
- Comparative items: Query translation time and query transmission time;
- Main factors: Number of storages, number of translation algorithms, query types, network environment, storage structure, and dataset size.

The key notations are defined as follows:

- S: Set of storages;
- $t(S \text{ to } S)$: SPARQL-to-SQL translation time by translation algorithms;
- $n(S)$: Number of storages;
- $t(TM)$: SQL query transmission time to a storage;
- $n(A)$: Number of translation algorithms;
- V_Q: SPARQL-to-SQL translation time affected by query type;
- V_T: Transmission time;
- V_{RAN}: random values, $0 < V_{\text{RAN}} < 1$.

For the performance evaluation, several assumptions and definitions are required, as follows:

- The query translation time, $t(S \text{ to } S)$, affected by the network environment, is $V_Q \times V_{\text{RAN}}$. V_Q is the actual translation time of the query type used in the Lehigh University benchmark. V_{RAN} is one of the values generated by a random number generator, $(0 < V_{\text{RAN}} < 1)$;
- The query transmission time, $t(TM)$, affected by the network environment, is $V_T \times V_{\text{RAN}}$. V_T is the SQL-to-storage transmission time;
- The SQL query type generated by the translation algorithm is affected by the SPARQL query type;
- This performance evaluation does not consider algorithm patterns, because we assume that the query translation time, $t(S \text{ to } S)$, generates uniform algorithm patterns, rather than different ones;
- This performance evaluation does not consider dataset size, because the evaluation uses the same dataset;
- We do not consider algorithm complexity because this evaluation focuses on the efficiency of the storage-independent model using a relational view, not on the performance of translation algorithms.

5.2. Performance evaluation

To evaluate the performance of the storage-dependent and storage-independent models, two key factors, $t(S \text{ to } S)$ and $t(TM)$, should be considered. The $t(S \text{ to } S)$ and $t(TM)$ factors are affected by $n(A)$ and $n(S)$, respectively. Estimation of model performance is defined as follows:

$$\text{Performance}_{(\text{Model})} = \sum_{i=1}^{n(A)} t(S \text{ to } S)_i + \sum_{j=1}^{n(S)} t(TM)_j. \quad (1)$$

In the storage-dependent model, a given SPARQL query should be translated to its corresponding SQL query by the translation algorithms that are valid for all storages. The $t(S \text{ to } S)$ factor is measured as $n(A)$. Then, the translated SQL is executed in different storage structures. The $t(TM)$ factor is measured as $n(S)$. In the storage-independent model, a given SPARQL should be translated to its corresponding SQL query by one specific translation algorithm. Therefore, the $n(A)$ value that affects $t(S \text{ to } S)$ is uniform. However, one SQL query is executed as many $n(S)$ values. The $t(TM)$ factor is measured as $n(S)$. The $t(S \text{ to } S)_i$ factor is defined as:

$$t(S \text{ to } S)_i = \begin{cases} n(A_i) \times V_{\text{RAN}} \times V_{Qi}, & 0 < V_{\text{RAN}} < 1 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

The $t(TM)_j$ factor is defined as:

$$t(TM)_j = \begin{cases} n(S_j) \times (1 - V_{\text{RAN}}) \times V_{Tj}, & 0 < V_{\text{RAN}} < 1 \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

5.3. Evaluation results

In this section, we compare and evaluate, quantitatively, the performance efficiency of the storage-dependent and storage-independent models using

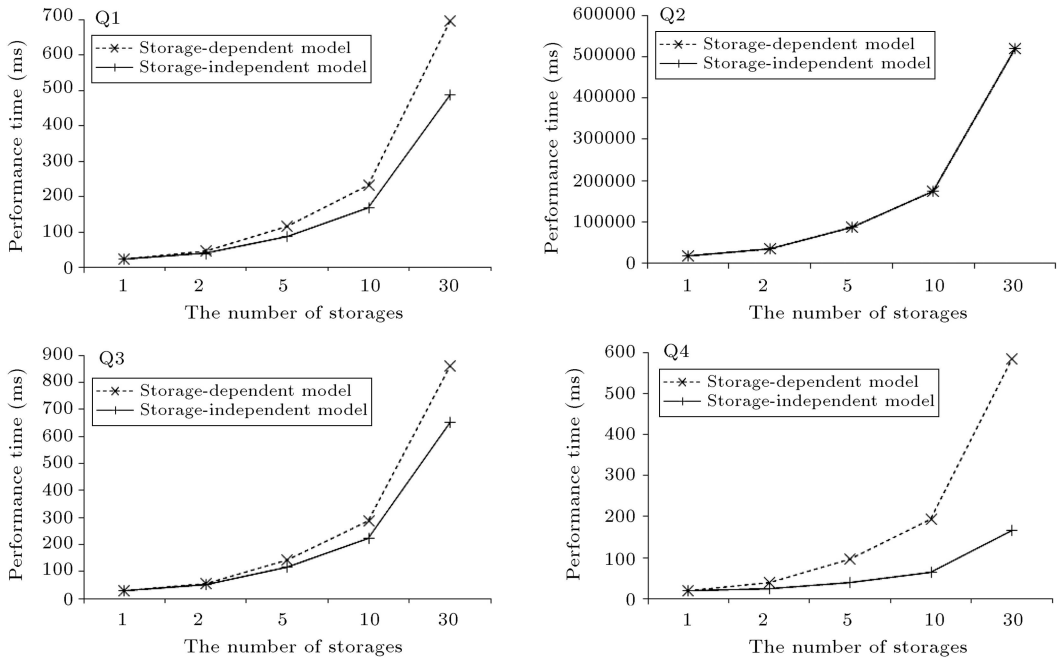

Figure 2. Performance results 1.

Eq. (1) defined in Section 5.2. This paper represents two evaluation results. First, we show the evaluation results for the case where the number of storages increased. Second, we show the evaluation results reflected by query type.

For the first evaluation, we measure the translation time of SPARQL to SQL and transmission time of SQL to storages. Each time was measured three times for each query, and we obtained the average values. Figures 2 and 3 show the performance evaluation of two models through 14 LUBM test queries. As shown in Figures 2 and 3, all the results from the storage-independent model represent a higher performance in comparison with all the results from the storage-dependent model. When the number of storages is one, the performance result is the same between the storage-independent and the storage-dependent models, because the translation algorithm equivalent to the storage is one. However, as the number of storages increases, the difference between the evaluation results of the two models ranges from a low of 1.1 times (Q2, Q7, and Q8) to a high of 23 times (Q12). On average, the performance results of the 14 test queries show a performance improvement of the storage-independent model, which is approximately three times faster than the storage-dependent model.

The second evaluation is performance by query patterns. The query patterns are determined by the number of join operations when the SPARQL query is translated to a SQL query. We define five types of query generated when increasing the number of join operations, as listed in Table 3, and then measure the performance time for both storage-dependent and storage-independent models as the number of storages increases: $n(S) = 1$, 5, and 10.

Figure 4 shows the performance evaluation results by query type using the five queries from Table 3. The performance results of the two models indicate that the more the number of join operations increases, the more SPARQL-to-SQL transformation time is required. In addition, the performance result is affected by the number of storages. When the number of storages is one, the performance time is the same between the storage-independent and storage-dependent models, as shown in Figure 4(a). Figure 4(b) and (c) show that when the number of storages increases, the performance difference between the two models is significant. In other words, when the query type is complicated, the storage-independent model requires less performance time than the storage-dependent model. Consequently, as shown by the two evaluation results, we can determine that the storage-independent model is more efficient than the storage-dependent model.

6. Conclusions and future work

In this paper, we propose a model that can independently apply a given translation algorithm to different storages. The proposed model is based on a relational view table, and we implemented a prototype. As the experiment results show, the proposed model ensures accuracy with regard to the query result. In addition, no data loss occurred from the query results based on the view table. Consequently, we show that using a view table, the proposed model can be used

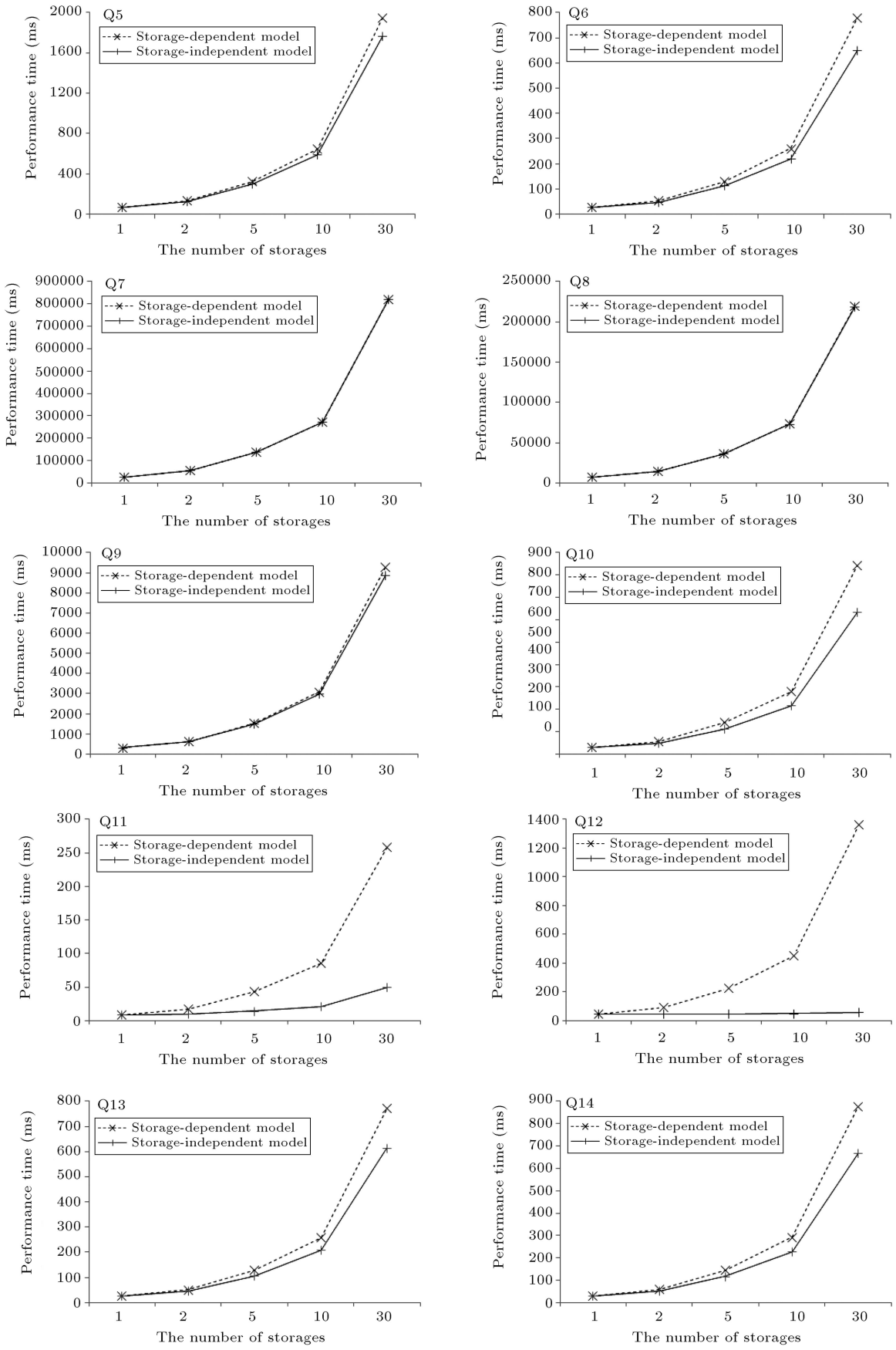

Figure 3. Performance results 2.

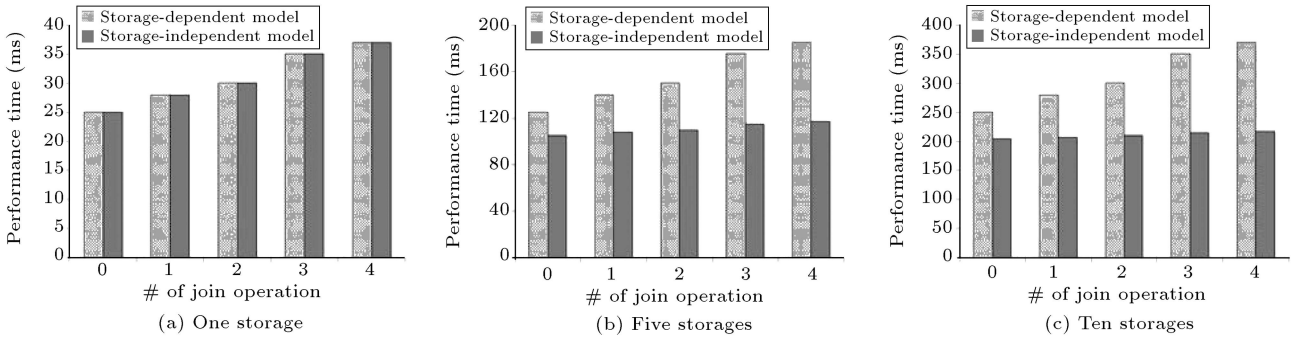

Figure 4. Performance results by query type.

Table 3. Query type of five patterns.

Query	SPARQL	n (join operation)
Q1	?X {?X type UndergraduateStudent}	0
Q2	?X {?X type Graduate Student. ?X takesCourse Uv::http://www.Department0.University0.edu/GraduateCourse0}	1
Q3	?X, ?Y {?X headOf ?Y. ?Y type Department. ?X worksforUv::http://www.Department0.University0.edu.}	2
Q4	?X, ?Y, ?Z {?X memberOf ?Z. ?Z subOrganizationOf ?Y. ?X undergraduateDegreeFrom ?Y}	3
Q5	?X, ?Y {?X type GraduateStudent. ?Y type GraduateCourse. ?X takesCourse ?Y. http://www.Department0.University0.edu/AssociateProfessor0 teacherOf ?Y}	4

independently of storage. In addition, creation of the view table is simpler compared to development of an algorithm. The view table is generated, not by demanding the development capability of high-level SQL syntax, but by simple SQL syntax. Therefore, the proposed model can save time and cost in the development, modification, and verification of the algorithm.

Future work includes a semi-automatic view generation method required for the high usability of the proposed model. In addition, the model will be extended to support inference queries. For verification and evaluation of the view generation method, we intend to apply it to various Web ontology storages.

Acknowledgments

This work was supported by a Korea University Grant, and the basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education, Science and Technology (2011-0025588).

References

1. Berners-Lee, T., Hendler, J. and Lassila, O. "The semantic web", *Scientific American*, **284**(5), pp. 34-43 (2001).
2. Shadbolt, N., Berners-Lee, T. and Hall, W. "The

semantic web revisited", *IEEE Intelligent Systems*, **21**(3), pp. 96-101 (2006).

3. ICS-FORTH, The RDF Query Language, http://139.91.183.30:9090/RDF/RQL/ (2008).
4. W3C, RDQL, http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/ (2004).
5. W3C, SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/ (2008).
6. Apache Jena, Jena Semantic Web Framework, http://jena.sourceforge.net/.
7. Broekstra, J., Kampman, A. and Harmelen, F.V. "Sesame: A generic architecture for storing and querying RDF and RDF schema", *Lecture Notes in Computer Science*, **2342**, pp. 54-68 (2002).
8. OWLJessKB, http://edge.cs.drexel.edu/assemblies/software/owljesskb.
9. Pan, Z. and Heflin, J. "DLDB: Extending relational databases to support Semantic Web queries", *2nd Int. Semantic Web Conf. (ISWC2003)*, Florida, USA, **2870**, pp. 109-113 (2003).
10. Harris, S. and Gibbins, N. "3store: efficient bulk RDF storage", *Int. Workshop on Practical and Scalable Semantic Systems (PSSS)*, Florida, USA, pp. 1-15 (2003).
11. Volz, R., Oberle, D., Motik, B. and Staab, S. "KAON SERVER - a semantic web management system", *Int. World Wide Web Conf. (WWW), Alternate Tracks - Practice and Experience*, Budapest, Hungary (2003).
12. Ma, L., Su, Z., Pan, Y., Zhang, L. and Liu, T. "RStar: an RDF storage and query system for enterprise resource management", *Int. Conf. on Information and Knowledge Management (CIKM)*, Washington, DC, USA, pp. 484-491 (2004).
13. OpenLink Software Virtuoso, http://virtuoso.openlinksw.com/
14. Theoharis, Y., Christophides, V. and Karvounarakis, G. "Benchmarking database representations of RDF/S stores", *Lecture Notes in Computer Science*, **3729**, pp. 685-701 (2005).
15. Chebotko, A., Fei, X., Lu, S. and Fotouhi, F. "Scientific workflow provenance metadata management using an RDBMS-based RDF store", Technical Report TR-DB-092007-CFLF, Wayne State University (2007).
16. Chebotko, A., Lu, S. and Fotouhi, F. "Semantics preserving SPARQL-to-SQL translation", *Data Knowl. Eng.*, **68**(10), pp. 973-1000 (2009).
17. Apache Jena, sparql2sql - a query engine for SPARQL over Jena triple stores, http://jena.sourceforge.net/
18. Harris, S. and Shadbolt, N. "SPARQL query processing with conventional relational database systems", *Lecture Notes in Computer Science*, **3807**, pp. 235-244 (2005).
19. Polleres, A. "From SPARQL to rules (and back)", *Int. World Wide Web Conf. (WWW)*, Banff, Alberta, Canada, pp. 787-796 (2007).
20. Schenk, S. "A SPARQL semantics based on datalog", *Lecture Notes in Computer Science*, **4667**, pp. 160-174 (2007).
21. Anyanwu, K., Maduko, A. and Sheth, A. "SPARQ2L: towards support for subgraph extraction queries in RDF databases", *Int. World Wide Web Conf. (WWW)*, Banff, Alberta, Canada, pp. 797-806 (2007).
22. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C. and Reynolds, D. "SPARQL basic graph pattern optimization using selectivity estimation", *Int. World Wide Web Conf. (WWW)*, Beijing, China, pp. 595-604 (2008).
23. Weiss, C., Karras, P. and Bernstein, A. "Hexastore: sextuple indexing for Semantic Web data management", *Proc. of the VLDB Endowment (PVLDB)*, **1**(1), pp. 1008-1019 (2008).
24. Chong, E.I., Das, S., Eadon, G. and Srinivasan, J. "An efficient SQL-based RDF querying scheme", *Int. Conf. on Very Large DataBases (VLDB)*, Trondheim, Norway, pp. 1216-1227 (2005).
25. Son, J., Kim, J.D., and Baik, D.K. "Performance evaluation of storage-independent model for SPARQL-to-SQL translation algorithms", NTMS, Paris, France, pp. 1-4 (2011).
26. W3C, Resource Description Framework, http://www.w3.org/RDF/ (2014).
27. W3C, RDF Vocabulary Description Language 1.1: RDF Schema, http://www.w3.org/TR/rdf-schema/ (2014).
28. Guo, Y., Pan, Z. and Heflin, J. "LUBM: A benchmark for OWL knowledge base systems", *J. of Web Semantics*, **3**(2), pp. 158-182 (2005).

Biographies

Jiseong Son received her BS degree in Computer Engineering from Seoul Women's University, Seoul, Korea, in 2007, and her MS degree, in 2009, in the same subject, from Korea University, where she is currently a PhD student in the Department of Computer and Radio Communications Engineering. Her research interests include semantic Web data management, relational databases, query translation, and ontology engineering, e-Learning, learning management systems, and access control.

Jeong-Dong Kim received his MS and PhD degrees in Computer Science and Computer Engineering, in 2008 and 2012, respectively, from Korea University, Seoul, Korea, where he is currently research Professor in the Department of Computer and Radio

Communications Engineering. His research interests include learning management systems, metadata-based integration, semantic Web, ontology, social computing, and access control.

Doo-Kwon Baik received his BS degree in Mathematics from Korea University, Seoul, Korea, in 1974, and his MS and PhD degrees in Computer Science from Wayne State University, USA, in 1983 and 1986, respectively. He is currently full Professor in the Department of Computer and Radio Communications Engineering, Korea University, Seoul, Korea. His research interests include modeling, simulation, data, and software engineering.