

Łukasz SZEREMETA, Dominik TOMASZUK  
University of Białystok, Institute of Informatics

## DOCUMENT-ORIENTED RDF GRAPH STORE

**Summary.** Document-oriented NoSQL databases are not commonly used in Semantic Web and Linked Data environments. The article describes the idea of an document-oriented RDF graph store. We present alternative RDF serialisation, allowing for efficient processing of graph data in an NOSQL graph store. This means that a database such as RethinkDB can be an RDF graph store. Moreover, our proposal supports various techniques for caching, which is a novelty for an RDF/JSON serializa-tion.

**Keywords:** RDF, NoSQL, Semantic Web, semi-structured data

## DOKUMENTOWY MAGAZYN GRAFÓW RDF

**Streszczenie.** Dokumentowe bazy danych NoSQL nie są powszechnie używane w środowiskach Semantycznego Internetu i Danych Połączonych. Artykuł omawia propozycję dokumentowego magazynu grafów RDF. Przedstawiamy alternatywną se-rializację RDF, pozwalającą na wydajne przetwarzanie danych grafowych w bazie da-nych NOSQL. Oznacza to, że baza danych taka, jak RethinkDB może być magazynem grafów RDF. Co więcej, nasza propozycja obsługuje różne techniki buforowania, co jest nowością dla serializacji RDF/JSON.

**Słowa kluczowe:** RDF, NOSQL, Semantyczny Internet, dane semistrukturalne

### 1. Introduction and Motivation

NoSQL databases are not commonly used in Semantic Web and Linked Data [2] envi-ronments. As opposed to SQL, they are not based on a relational model. They usually use the semi-structured model, thanks to which they do not impose strictly defined schemes, and thus allow for a flexible serialisation of the Resource Description Framework (RDF) language, commonly used in the Semantic Web environment.

One of the possible formats of data in accordance with the semi-structured model is JavaScript Object Notation (JSON), considered an alternative to Extensible Markup Language (XML). The main advantage of the JSON format is the possibility of its quick processing by machines. Semi-structured data is often used in document-based databases.

The article proposes a modified RDF/JSON serialisation and a normalizing algorithm. It presents an idea of a document-based RDF graph store based on the JSON format, and it presents the result of research connected with its efficiency. Our proposal is cache-friendly. We conducted tests on the data generated by Berlin SPARQL Benchmark (BSBM). In addition, we present one of the first implementations of RDF/JSON serialisation and a complete RDF graph store.

The article is composed of the following sections: Section 2 presents related papers. Section 3 introduces the bases of RDF language and its serialisations. Section 4 demonstrates the architecture of the proposed system. Section 5 discusses research conducted in the test environment. The article ends with a conclusion.

## 2. Related Work

Current papers are mainly focused on finding the best method of storing and processing data in the RDF language. As the authors of cover article [15] admit that RDF may be well used for saving semi-structured data in different applications, however, the effective processing of this information is only possible in the cloud computing. The next article [6] compares different NoSQL databases processing RDF data. The authors point at the fact that NoSQL databases have a high potential at processing RDF data, yet they also notice the problems connected with optimization of queries. The author in [10] proposed ArangoDB – multi-model database. The database allows to choose between document and graph data model. The database uses AQL – SQL-like declarative query language.

The authors of paper [21] present an interesting combination of non-relational, distributive NoSQL database, based on Google BigTable – Apache HBase<sup>1</sup> with the MapReduce technique [9]. Rya [22] – a scaled RDF database using Accumulo<sup>2</sup> and built based on Google BigTable is yet another idea. The implementation of the said database as a plugin to OpenRDF Sesame [4] allows for handling SPARQL queries and different RDF serializations.

Scalability is also emphasized by authors [16] and [21]. The former present an HBase based Jena-HBase triplestore, and the latter – PigSPARQL [27], i.e. a system enabling the

---

<sup>1</sup> <http://hbase.apache.org/>

<sup>2</sup> <https://accumulo.apache.org/>

processing of complex SPARQL queries in the MapReduce store. Article [24] presents an approach to solving the problems of graph pattern matching by the MapReduce technique with implementation of advanced algebra.

The authors in [12] propose an expanded system of query evaluation/optimization with large amounts of RDF – Partout data and algorithms of braking down data into fragments. In their paper, Hose and Schenkel [13] present a method of load balancing based on partitioning of data and replication of triples. Another idea worth mentioning is Trinity.RDF [33] – a distributed, scaled RDF database. This solution is based on the operation of linking. The authors in [19] present a different approach. They introduce a special algorithm that allows for the optimization of linking in the in the H2RDF database using MapReduce. The authors in [7] present techniques of effective breadth-first search. They also present how such solutions accelerate the interpretation of RDF graphs compared with competitive methods.

Paper [25] presents the SHARD graph database constructed based on Apache Hadoop<sup>3</sup>, where special iterative algorithms constitute a basis for the scalability of the whole system. Another interesting solution, also based on Apache Hadoop, is the S2RDF presented in [27]. It is characterized by a compilation of SPARQL to SQL with the use of semi-join optimization. What is important, the proposed techniques allow for the achievement of very good processing speeds, also for longer queries.

The range of methods allowing for the achievement of SPARQL scalability with large RDF graphs was presented in paper [14]. One of the ideas is the division of SPARQL queries into high-performance parts, which utilize the way in which they are divided within a cluster. Another way is the placing of data between nodes in a ways allowing for the acceleration of processing queries through local optimizations.

The authors in [1] present the Amada platform, based on the Amazon Web Services (AWS) and the Software as a Service (SaaS) approach. The proposed solution allows for the storage of data in the network, especially XML documents and RDF graphs. The article also presents how to construct and perform operations on their RDF graph store. Article [5] presents the study of the problem of indexing RDF data stored in the Amazon cloud. The SimpleDB database provided by AWS for small data was used for this purpose [23]. The authors of [29] also focused on the same software. They proposed the Stratustore, which works as a back-end for the Jena Semantic Web framework [18] and stores data in SimpleDB. Another key-value database is the CumulusRDF based on Apache Cassandra<sup>4</sup> presented in [17]. The authors worked out two index schemes for RDF in order to support Linked Data search and the basic search of triples according to the template.

---

<sup>3</sup> <http://hadoop.apache.org/>

<sup>4</sup> <http://cassandra.apache.org/>

### 3. Basic Ideas

The basic components of the RDF model relate to the resources. The elements are composed of three disjoint subsets: Internationalized Resource Identifier (IRI), literals, and blank nodes.

IRI references are global identifiers that may be used in order to define any resource.

**Example 3.1.** *The example illustrates the IRI reference that is used to define house in DBpedia<sup>5</sup>.*

```
<http://dbpedia.org/resource/House>
```

Literals are lexical values. Each literal has a data type that is identified by the IRI reference. Optionally, the literal may also include a language mark.

**Example 3.2.** *The example illustrates a literal with a type.*

```
"3.14"^^http://www.w3.org/2001/XMLSchema#float
```

The final subset are blank nodes that are variables used for marking a certain resource, for which the literal or IRI reference is not given.

**Example 3.3.** *The example presents a blank node.*

```
_ :x
```

The RDF data model is based on the idea of creating sentences about Internet resources in the *subject-predicate-object* form. In RDF terminology, these sentences are called RDF triplets. The subject defines the described resource. The predicate describes the characteristics and aspects of the resource and expresses the relation between the subject and the object. The object stores the value of this relation. An RDF triple definition is presented in [31].

### 4. Semi-structured graph store

This section will describe an idea of a semi-structured RDF graph store. Our proposition is based on RDF/JSON [32] serialisation. We extend this approach to be more effective, because we adjust it to run with cache environment.

---

<sup>5</sup> <http://dbpedia.org/>

The syntax of RDF/JSON is meant for easy integration with implemented systems that use JSON language. Publishing RDF data in this format makes it accessible to web developers without installing additional libraries and other software for modifying documents. An example of RDF/JSON serialisation is included in example 4.1.

According to [32] the `type` and `value` keys are obligatory. The former describes the value type and the latter the given value. The handled values of data types are `uri`, `bnode`, `literal` and `typed-literal`. Additional keys are `lang`, defining the language and `datatype`, defining the type. The former may occur only when the `type` key has been defined as a literal, and the latter – only in the case of a `typed-literal`. Serialisation allows for supplementing and RDF triplet (definition 3.1) with context by using the `context` box.

Table 1 presents the rules of using particular types of data in each fields.

Table 1

Comparison of data types in RDF/JSON serialization

type	subject	predicate	object	context
IRI (uri) reference	yes	yes	yes	yes
(bnode) blank node	yes	no	yes	no
(literal) literal	no	no	yes	no

The proposed semi-structured RDF graph store is composed of collections of data that contain semis-structured documents.

**Definition 4.1. (Semi-structured document).** *Presuming that the key/value pair is  $P = \langle k, v \rangle$ , where  $k \in L$  is the key and  $v$  is the value of the key that may contain a logical, numerical value, a string of signs or another key/value pair. The semi-structured document  $D$  may be defined as  $D = \{P_1, P_2, \dots, P_n\}$  for  $n \geq 1$ .*

**Example 4.1.** *The example presents RDF/JSON<sup>6</sup> serialisation identical to example 3.4.*

```
{[{
  "subject":
  {
    "type" : "uri",
    "value" : "http://example.net/me#jk"
  },
  "predicate":
  {
    "type" : "uri",
    "value" : "http://xmlns.com/foaf/0.1/name"
  },
  "object":
  {
    "type" : "literal",
    "value" : "Jan Kowalski"
  }
}]}
```

---

<sup>6</sup> `uri` is treated as IRI

It is also presumed that semi-structured document represents exactly one RDF graph (G) and may contain named graphs (NG). According to [30], we adopt a data collection definition below.

**Definition 4.2 (Data collection).** *Data collection is a  $C = \langle id, [D_1, D_2, \dots, D_i] \rangle$  tuple, where  $id \in L$  is a collection name, and  $[D_1, D_2, \dots, D_i]$  is a list of semi-structured  $i \geq 1$  documents.*

**Definition 4.3. (Semi-structured RDF graph store).** *A semi-structured RDF graph store is  $GS = \{C_1, C_2, \dots, C_i\}$ , where every  $C_m$  means collections of data,  $m \geq 0$ .*

## 5. Implementation and Experiments

This subsection will present load tests that were performed with the Berlin SPARQL Benchmark (BSBM) [3]. This generator, in its original form, does not include the option of handling RDF/JSON serialisation. The authors have presented their implementation of this serialisation<sup>7</sup>.

The tested graph is after skolemization [8], so it does not contain blank nodes.

Fig. 1a. Without optimizations

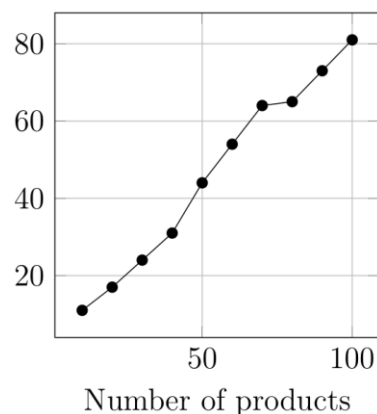


Fig. 1b. With optimizations

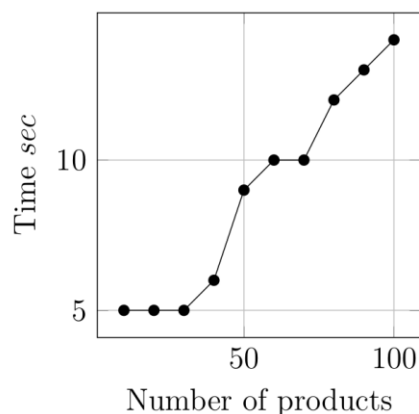


Fig. 1. Loading data without and with optimizations

Rys. 1. Testy ładowania bez i z optymalizacją

The load tests measure the time needed for loading all RDF triplets to the testing environment with use of the proposed client – RTriples<sup>8</sup> to the RethinkDB database. Fig. 1a presents the tests without the proposed optimizations. The times seem too long in real-life applications. Fig. 1b presents the results of loading data with the use of a cache. It is visible in the

<sup>7</sup> <https://github.com/lszeremeta/bsbmttools-json>

<sup>8</sup> <https://github.com/lszeremeta/RTriples>

chart that thanks to the reduction of the input/output operation, the loading times are much improved and seem to be satisfactory even in the case of larger amounts of data.

Another set of tests allowing for assessment of the effectiveness of the proposed system concerned the operation of selecting data from the semi-structured graph store. The time for all handled output formats and the data limit at level 10 and 100 was measured. The tests with a limit value were conducted directly after the experiments with a smaller value of this parameter and without restarting the server database. The testing environment was tested according to three REST [26] queries presenting in table 2.

Table 2

Testing queries		
Name	Description	Query syntax
Q1	query selecting all results	http://localhost:9292/
Q2	query selecting predicates and objects matching the given subject	http://localhost:9292/?s=http%3A%2F%2Fwww4.wiwiss.fu-berlin.de%2Fbizer%2Fbsbm%2Fv01%2Finstances%2FdataFromVendor1%2FOffer1629
Q3	query selecting subjects and predicates for a given object of typed-literal type with a certain type	http://localhost:9292/?o=2008-04-01T00:00:00&type=typed-literal&datatype=http://www.w3.org/2001/XMLSchema%23dateTime

Fig. 2 presents tests without the proposed optimizations. The performance times seem to be acceptable, especially when operation on small data sets.

Fig. 2a. The limit is set to 10

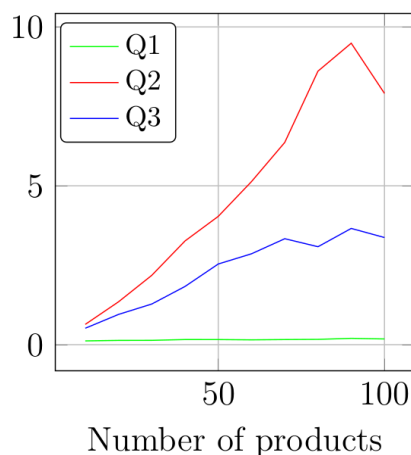


Fig. 2b. The limit is set to 100

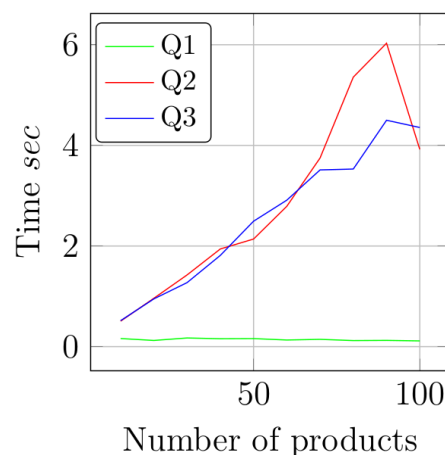


Fig. 2. Select data tests without proposed optimizations

Rys. 2. Testy wybierania bez optymalizacji

Table 3 presents a comparison of times for the Q2 query for all handled formats. It may be noticed that the values are very similar.

Table 3

Test of selecting without optimization: Q2 query, all selected formats, limit equalled 10  
(times given in seconds)

Set	HTML	RDF/JSON	N-Triples	CSV	TXT
10	0.639317	0.499001	0.506421	0.502461	0.504450
20	1.349230	0.965588	0.969615	0.969669	0.966011
30	2.182343	1.423137	1.424770	1.425798	1.423723
40	3.264202	1.958241	1.958052	1.954727	1.962659
50	4.037859	2.136330	2.138006	2.147609	2.133514
60	5.132782	2.794015	2.796026	2.803888	2.799835
70	6.364546	3.738322	3.751541	3.749293	3.749151
80	8.608902	5.383463	5.390001	5.374771	5.378377
90	9.487836	6.024153	6.048865	6.050515	6.028922
100	7.907734	3.981267	3.978394	3.952579	3.975572

Fig. 3a. Limit equal to 10

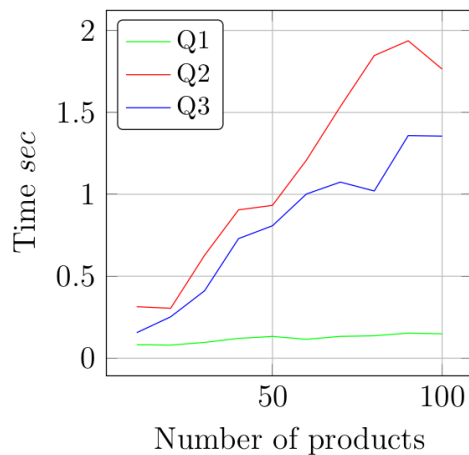


Fig. 3b. Limit equal to 100

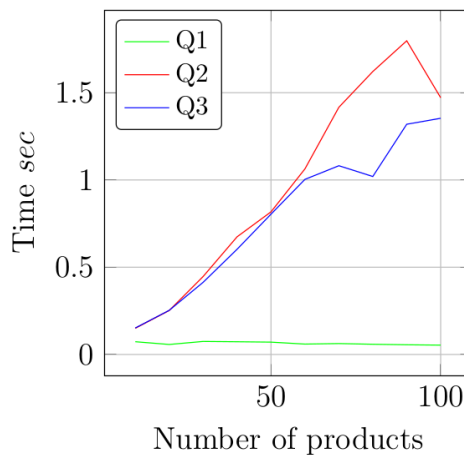


Fig. 3. Select data tests with caching

Rys. 3. Testy wybierania z buforowaniem

Table 4

Tests of selecting with caching: Q2 query, all handled formats, limit equalled 10  
(times given in seconds)

Set	HTML	RDF/JSON	N-Triples	CSV	TXT
10	0.314240	0.151175	0.149455	0.149980	0.149815
20	0.304652	0.251588	0.251627	0.259146	0.250956
30	0.628254	0.591456	0.594124	0.591899	0.594013
40	0.905058	0.665024	0.669216	0.670328	0.669601
50	0.932533	0.804828	0.813917	0.816908	0.813912
60	1.208271	1.055738	1.060337	1.063326	1.060050
70	1.534901	1.415004	1.415126	1.412973	1.418806
80	1.846534	1.626495	1.631112	1.630704	1.633280
90	1.936703	1.767093	1.790035	1.793109	1.793608
100	1.763964	1.453812	1.466341	1.468721	1.464906



Fig. 4a, 4b, and Table 4 present times of RDF triples' selection for the testing environment after the use of caching similar to the LRU [20] model. The results show that loading times are about 5 times shorter than without optimization (Fig. 3).

Charts 4a and 4b present the results of select data tests with caching on and the HTTP headers set to `Cache-Control` and `Expires` [11]. Using the proper HTTP headers is especially important in real-life applications, effectively limiting the amount and time of data transfer.

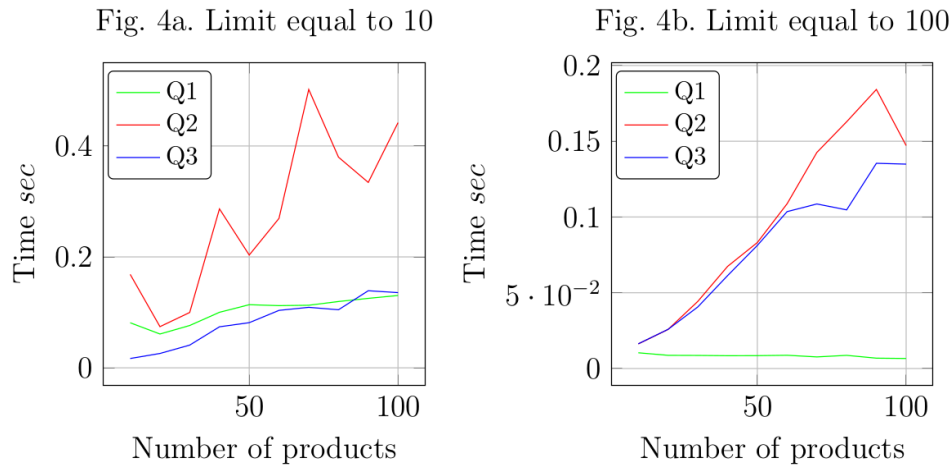


Fig. 4. Select data tests with caching and HTTP headers

Rys. 4. Testy wybierania z buforowaniem i ustawionymi nagłówkami HTTP

## 6. Conclusion

The problem how to store RDF data in relational and graph databases has seen many solutions. Unfortunately there are few of them that utilize document-based databases. In the article, we present a complete document-based RDF graph store. Its main advantage is the ability to store data in the form of RDF documents and to bridge the already existing solutions and the classic RDF graph stores.

In future projects, we will focus on the improvement of data processing efficiency in our document-based RDF graph store, concentrating on the indexation of RDF triplets and caching data.

**BIBLIOGRAPHY**

1. Aranda-Andújar A., Bugiotti F., Camacho-Rodríguez J., Colazzo D., Goasdoué F., Kaoudi Z., Manolescu I.: AMADA: web data repositories in the Amazon Cloud. Proceedings of the 21<sup>st</sup> ACM international conference on Information and knowledge management, ACM, 2012, p. 2749÷2751.
2. Bizer C., Heath T., Berners-Lee T.: Linked data-the story so far. 2009.
3. Bizer C., Schultz A.: The Berlin SPARQL benchmark. 2009.
4. Broekstra J., Kampman A., van Harmelen F.: Sesame: A generic architecture for storing and querying RDF and RDF schema. In The Semantic Web – ISWC 2002, Springer, 2002, p. 54÷68.
5. Bugiotti F., Goasdoué F., Kaoudi Z., Manolescu I.: RDF data management in the Amazon Cloud. Proceedings of the 2012 Joint EDBT/ICDT Workshops, ACM, 2012, p. 61÷72.
6. Cudré-Mauroux F., Enchev I., Fundatureanu S., Groth P., Haque A., Harth A., Keppmann F.L., Miranker D., Sequeda J.F., Wylot M.: NoSQL databases for RDF: An empirical evaluation. In The Semantic Web – ISWC 2013, Springer, 2013, p. 310÷325.
7. Cuzzocrea A., Cosulschi M., de Virgilio R.: An Effective and Efficient MapReduce Algorithm for Computing BFS-Based Traversals of Large-Scale RDF Graphs. Algorithms, Vol. 9(1), 2016, p. 7.
8. Cyganiak R., Wood D., Lanthaler M.: RDF 1.1 Concepts and Abstract Syntax. W3C recommendation, World Wide Web Consortium, February 2014, <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.
9. Dean J., Ghemawat S.: MapReduce: simplified data processing on large clusters. Communications of the ACM, Vol. 51(1), 2008, p. 107÷113.
10. Dohmen L., Edlich I.S., Hackstein M.: A Declarative Web Framework for the Server-side Extension of the Multi Model Database ArangoDB. 2014.
11. Fielding R., Nottingham M., Reschke J.: Hypertext Transfer Protocol (HTTP/1.1): Caching. Technical Report 7234, RFC Editor, June 2014, <http://www.rfc-editor.org/rfc/rfc7234.txt>.
12. Galárraga L., Hose K., Schenkel R.: Partout: A distributed engine for efficient RDF processing. Proceedings of the companion publication of the 23<sup>rd</sup> international conference on World Wide Web companion, International World Wide Web Conferences Steering Committee, 2014, p. 267÷268.

13. Hose K., Schenkel R.: WARP: Workload-aware replication and partitioning for RDF. 2013 IEEE 29<sup>th</sup> International Conference on Data Engineering Workshops (ICDEW), IEEE, 2013, p. 1÷6.
14. Huang J., Abadi D.J., Ren K.: Scalable SPARQL querying of large RDF graphs. Proceedings of the VLDB Endowment, Vol. 4(11), 2011, p. 1123÷1134.
15. Kaoudi Z., Manolescu I.: RDF in the Clouds: A Survey. The VLDB Journal, 2014, p. 1÷25.
16. Khadilkar V., Kantarcioglu M., Thuraisingham B., Castagna P.: Jena-hbase: A distributed, scalable and efficient RDF triple store. Proceedings of the 11<sup>th</sup> International Semantic Web Conference Posters & Demonstrations Track, ISWC-PD, Vol. 12, Citeseer, 2012, p. 85÷88.
17. Ladwig G., Harth A.: CumulusRDF: Linked Data Management on Nested Key-Value Stores. Proceedings of the 7<sup>th</sup> International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011) at the 10<sup>th</sup> International Semantic Web Conference (ISWC2011), 2011.
18. McBride B.: Jena: A semantic web toolkit. IEEE Internet computing, Vol. 6(6), 2002, p. 55÷59.
19. Papailiou N., Konstantinou I., Tsoumakos D., Koziris N.: H2RDF: adaptive query processing on RDF data in the cloud. Proceedings of the 21<sup>st</sup> international conference companion on World Wide Web, ACM, 2012, p. 397÷400.
20. Podlipnig S., Böszörményi L.: A survey of web cache replacement strategies. ACM Computing Surveys (CSUR), Vol. 35(4), 2003, p. 374÷398.
21. Przyjaciół-Zablocki M., Schätzle A., Hornung T., Dorner C., Lausen G.: Cascading map-side joins over HBase for scalable join processing. CoRR, abs/1206.6293, 2012.
22. Punnoose R., Crainiceanu A., Rapp D.: Rya: A Scalable RDF Triple Store for the Clouds. Proceedings of the 1<sup>st</sup> International Workshop on Cloud Intelligence, Cloud-I '12, ACM, New York, NY, USA 2012.
23. Ramanathan S., Goel S., Alagumalai S.: Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable. 2011 International Conference on Recent Trends in Information Systems (ReTIS), IEEE, 2011, p. 165÷168.
24. Ravindra P., HyeonSik K., Anyanwu K.: An intermediate algebra for optimizing RDF graph pattern matching on MapReduce. In The Semantic Web: Research and Applications, Springer, 2011, p. 46÷61.
25. Rohloff K., Schantz R.E.: Clause-iteration with MapReduce to scalably query data-graphs in the SHARD graph-store. Proceedings of the fourth international workshop on Data-intensive distributed computing, ACM, 2011, p. 35÷44.

26. Fielding R.T.: Architectural styles and the design of network-based software architectures. Diss. University of California, Irvine 2000.
27. Schätzle A., Przyjaciół-Zablocki M., Hornung T., Lausen G.: PigSPARQL: a SPARQL query processing baseline for big data. Proceedings of the 2013<sup>th</sup> International Conference on Posters & Demonstrations Track-Volume 1035, CEUR-WS. Org, 2013, p. 241÷244.
28. Schätzle A., Przyjaciół-Zablocki M., Skilevic S., Lausen G.: S2RDF: RDF Querying with SPARQL on Spark. arXiv preprint arXiv:1512.07021, 2015.
29. Stein R., Zacharias V.: RDF on cloud number nine. 4<sup>th</sup> Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic, 2010, p. 11÷23.
30. Tomaszuk D., Rybiński H.: Grouping Multiple RDF Graphs in the Collections. International Conference: Beyond Databases, Architectures and Structures, Communications in Computer and Information Systems, Vol. 424, Springer International Publishing, 2014.
31. Tomaszuk D., Skonieczny Ł., Wood D.: RDF graph partitions: A brief survey. International Conference: Beyond Databases, Architectures and Structures, Communications in Computer and Information Systems, Vol. 521, Springer International Publishing, 2015.
32. Tomaszuk D.: Named graphs in RDF/JSON serialization. Zeszyty Naukowe Politechniki Gdańskiej, 2011, p. 273÷278.
33. Zeng K., Yang J., Wang H., Shao B., Wang Z.: A distributed graph engine for web scale RDF data. Proceedings of the VLDB Endowment, Vol. 6(4), 2013, p. 265÷276.

## Omówienie

Bazy danych NoSQL (ang. Not Only SQL) nie są powszechnie używane w środowiskach Semantycznego Internetu (ang. Semantic Web) i Danych Połączonych (ang. Linked Data). W przeciwieństwie do SQL, nie bazują zwykle na modelu relacyjnym. Korzystają one najczęściej z modelu semistrukturalnego, dzięki czemu nie narzucają ściśle określonych schematów, a więc pozwalają w sposób elastyczny na serializację języka Resource Description Framework (RDF), używanego powszechnie w środowisku Semantycznego Internetu. Jednym z przykładowych formatów danych zgodnych z modelem semistrukturalnym jest JavaScript Object Notation (JSON), uważany za alternatywę dla Extensible Markup Language (XML). Główną zaletą formatu JSON jest możliwość szybkiego przetwarzania przez maszyny. Dane semistrukturalne często wykorzystywane są w dokumentowych bazach danych. Artykuł omawia propozycję dokumentowego magazynu grafów RDF. Przedstawiamy alternatywną serializację RDF, pozwalającą na wydajne przetwarzanie danych grafowych w bazie danych NOSQL. Oznacza to, że dokumentowa baza danych, taka jak RethinkDB może być magazy-

nem grafów RDF. W artykule zaproponowano zmodyfikowaną serializację RDF/JSON. Artykuł omawia propozycję dokumentowego magazynu grafów RDF opartego na formacie JSON, a także przybliża wyniki badań związanych z jej wydajnością, w szczególności z buforowaniem (rys. 1, rys. 3, tabela 4). Jego główną zaletą jest to, że przechowuje on dane w postaci dokumentów RDF i może być dobrym pomostem między rozwiązaniami już istniejącymi a klasycznymi magazynami grafów RDF. Autorzy przeprowadzili testy na wygenerowanych za pomocą narzędzia testów wzorcowych Berlin SPARQL Benchmark (BSBM). Prezentujemy ponadto jedno z pierwszych implementacji serializacji RDF/JSON i kompletny magazyn grafów RDF.

## Addresses

Łukasz SZEREMETA: University of Białystok, Institute of Informatics, ul. Konstantego Ciołkowskiego 1M, 15-245 Białystok, Poland, lszeremeta@ii.uwb.edu.pl.

Dominik TOMASZUK: University of Białystok, Institute of Informatics, ul. Konstantego Ciołkowskiego 1M, 15-245 Białystok, Poland, dtomaszuk@ii.uwb.edu.pl.