A Massive RDF Storage Approach Based on Graph Database

Liu Jie
No.30 Institute of China Electronics Technology Group Corporation
Cyberspace security lab of CETC
Chuangye Rd., Chengdu, China
(+86)13808226348
liujie.bupt@gmail.com

Chen Jianfeng
No.30 Institute of China Electronics Technology Group Corporation
Cyberspace security lab of CETC
Chuangye Rd., Chengdu, China
(+86)13980995721
atrix@163.com

Rao Zhihong
No.30 Institute of China Electronics Technology Group Corporation
Cyberspace security lab of CETC
Chuangye Rd., Chengdu, China
(+86)13608184980
sleetext2@163.com

Sun Zhi
No.30 Institute of China Electronics Technology Group Corporation
Cyberspace security lab of CETC
Chuangye Rd., Chengdu, China
(+86)13881986933
sunzhi1019@163.com

Yang Hui
No.30 Institute of China Electronics Technology Group Corporation
Cyberspace security lab of CETC
Chuangye Rd., Chengdu, China
(+86)13648071340
yanghui032178@163.com

Xu Rui
No.30 Institute of China Electronics Technology Group Corporation
Cyberspace security lab of CETC
Chuangye Rd., Chengdu, China
(+86)13666167624
jasmine_x@163.com

ABSTRACT

The efficient management of massive Resource Description Framework (RDF) data is very important in the future of Semantic Web. The RDF data has a natural structure of graph. Thus, the traditional management models such as relational data model and object data model are difficult to meet the requirements of both low data redundancy and high query performance. If it uses a graph model to manage the RDF data, then there is no need to converse the logical data model to the physical data model. This paper proposes a massive RDF storage approach based on graph database. In the perspective of graph model, it partitions the RDF dataset based on a heuristic greedy strategy. It also considers the dynamic property of the data flow. It uses a dynamic mechanism of replicating, storing and deleting the RDF data to achieve load balancing. The experiment is conducted by data storage and query on three datasets. The experimental results show that the proposed approach outperforms traditional methods based on relational database. It also proves that the proposed approach could effectively support the distributed storage of the RDF data.

CCS Concepts

• **Information systems ~ Information storage systems**

Keywords

Graph database; Massive RDF data management; RDF dataset partitioning; Distributed storage.

ICGDA '18, April 20–22, 2018, Prague, Czech Republic

ACM ISBN 978-1-4503-6445-4/18/04…$15.00
DOI: https://doi.org/10.1145/3220228.3220253

1. INTRODUCTION

As an important feature of the Web3.0 era [1], Semantic Web [2] has become the focus of the public attention. As a format standard of semantic web, RDF defines a mechanism for describing resources. In recent years, many fields began to adopt RDF as the basic expression of metadata. The scale of the RDF dataset is constantly expanding. To achieve the transformation from the World Wide Web to the Semantic Web, how to efficiently store large-scale RDF data has become one of the most important tasks.

Most current RDF data management systems handle RDF data by using a relational data model or object data model. The RDF datasets are stored in a relational database or an object-oriented database. The typical works include Sesame, Jena and RDF-3X [3]. However, the relational database has a fixed table structure. Thus, it is often necessary to design multiple tables to meet different RDF queries when designing a storage scheme. This imposes many limitations on the storage scheme based on relational database. Thus, storage methods based on relational database are less adaptable to large-scale RDF data management.

The RDF data has a natural structure of graph. The graph model conforms to the semantic level of the RDF data. It could maintain the semantic information of the RDF data to the maximum extent. It is also beneficial to the query of the semantic information. Therefore, the graph database should be considered for the storage of the RDF data. However, there are still some problems to be solved [4][5]. The scale of the RDF data is expanding at a high speed. The centralized management has been difficult to meet the requirements of RDF data storage. Therefore, it needs distributed storage method [6].

When the RDF datasets are conducted distributed storage, they should be partitioned. A simple method of partitioning is to conduct hash mapping to the RDF data. It determines the storage partition according to the hash value. However, this method does not consider the graph characteristics of the RDF data. Two closely related vertices may be assigned to different storage partitions. When traversing the RDF graph, there will be a large number of cross-storage partition accesses. In addition, it is not conducive to the RDF query through subgraph matching. Furthermore, the RDF data in the system is gradually increasing.

It is often a dynamic data stream. Thus, the storage approach should consider the dynamic property of the RDF data [7].

The contribution of this paper includes: It proposes an approach to partition the RDF dataset based on its structure of graph. It considers the dynamic property of the RDF data and uses a dynamic mechanism of replicating, storing and deleting to achieve load balancing. It validates the effectiveness of the proposed approach by data storage and query tasks on three datasets.

2. RELATED WORKS

2.1 Graph Partitioning Method

Graph partitioning is a NP-complete problem [8], where most methods adjust their strategy based on heuristic rules. Two main types exist: the heuristic one of vertices exchanging, such as KL algorithm; and the graph partition framework with multi-layers.

(1) Heuristic method. The typical algorithms include KL algorithm and FM algorithm [9]. KL algorithm uses the algorithm framework of recursive dichotomy: in each round of dichotomy, firstly divide the graph into two equal parts randomly, then calculate the beneficial value by exchanging two random vertices, at last select two vertices with the highest beneficial value and exchange them. The time complexity of the algorithm is $O(n^2)$.

(2) Multi-layered graph partition method. KL algorithm can only process graph within 10^4 vertices. In order to deal with graph in a larger scale, [10] proposed a multi-layered partitioning framework METIS. METIS could process large graph with less than one million nodes. [11] proposed a parallel version named METIS-ParMetis, which can handle larger graphs with ten million vertices.

The above two methods are mainly targeting at static graph data, however graph data model in practical applications are generally dynamic graph data flow. Many RDF data sets were not provided with all data in the very beginning, instead new vertices or nodes will be added to the system at regular intervals.

2.2 Heuristic Partitioning Strategy

The heuristic partitioning function assign storage area for vertex v. Typical types include: 1) balanced strategy. 2) Grouping strategy. 3) hash strategy. 4) greedy strategy.

Each strategy has its emphasis respectively. Balanced strategy and grouping strategy are the easiest way for a balanced partitioning, but ignore the structural characteristics of graph data. Hash strategy is the partitioning method employed by many current systems. Its main advantage is that every vertex can get result very fast without the necessity of referring to the information table of the whole partition. In case the value of these results are continuous, namely $H(V)$, then its effectiveness is comparative to balanced strategy and grouping strategy. It can be also understood as, if a pseudo-random hash function is used then the partition outcome is equal to random partitioning. Greedy strategy put its emphasis on the partition distribution of the vertices that related to v in the graph, it will choose partition with more vertices related with priority. A weight function w is defined as the same time to balance the influence introduced by different vertices size.

3. MASSIVE RDF STORAGE BASED ON GRAPH DATABASE

It needs to consider the following key factors when undertaking large-scale RDF data storage on a graph database: (1) Communication costs. The number of edges across the storage servers determines the communication overhead of system. To reduce the cost of the communication, it should minimize the number of edges across the storage servers. (2) Load balancing. Each partition of the dataset should have a similar size. (3) Storage redundancy. In order to reduce the cost of the communication, it should replicate specific nodes and edges to other storage servers. It should also maintain the consistency of multiple copies.

3.1 Dataset Partition Based on Heuristic Greedy Strategy

From the perspective of graph model, the RDF dataset is partitioned based on heuristic greedy strategy.

The heuristic greedy strategy is depicted as follows. Suppose there are k partitions, and the maximum vertex capacity of each partition is C. $P(i)$ represents the current state of vertices and edges of partition i. $|P(i)|$ represents the number of vertices in partition i. $P=\{P(1),P(2),\ldots,P(k)\}$is states set of all the current partitions. For graph $G=(V,E)$, $V=\{v_1,v_2,\ldots,v_n\}$ represents the vertex set, $E=\{e_1,e_2,\ldots,e_m\}$ represents the edge set. $e(v_i)$ denotes the adjacent edge of vertex v_i. $f(v_i)$ denotes the adjacent vertex set of v_i. $|e(v_i)|$ represents the number of adjacent edges. $|f(v_i)|$ represents the number of adjacent vertices.

Count the number of adjacent vertices of v_i in each partition. N_j represents the number of adjacent vertices of v_i in partition j. Set the weight function $w=1-|P(j)|/C$. This weight function means it uses the remaining capacity of each partition as the weight. W_j represents the weight of each partition, then $W_j = N_j \times w$. Select the partition with the largest weight. If there are multiple partitions meeting the requirements, then randomly select one. The Partition selecting function is depicted as follows:

$$Partition(v_i) = random(\{ j \mid \max(|W_j|), j \in |k|\}) \qquad (1)$$

The RDF dataset partition algorithm contains two stages. The first stage includes the generation of subgraph information. The second stage includes the subgraph partition. The RDF dataset partition algorithm is described as follows.

Algorithm: RDF dataset partitioning

Input: RDF items, partitions number K, max partition capacity C.

Output: partitions table, global vertex table, global edge table.

//Stage1: Generation of subgraph information.

(1) Convert the resource vertices and text vertices in RDF items into the vertex ID and vertex attribute information.
(2) Convert edges into edge ID and edge attribute information.
(3) Add the vertex ID in the global vertex table.
(4) Add the edge ID in the global edge table.
(5) Generate the subgraph G_j by using the vertex ID and edge ID.
(6) Generate the adjacent vertex set V_j of the subgraph G_j.

//Stage2: Partition the subgraph.

(7) for ($v_i \in V_j$){
(8) if (v_i has been assigned a partition) to the next vertex v_{i+1}.
(9) else{
(10) Generate the adjacent vertex set $f(v_i)$ of v_i.
(11) Use function $Partition(v_i)$ to find out the partition that v_i should be assigned into.
(12) Update partitions table, global vertex table, global edge table.

(13) }//end else
(14) }//end for

3.2 Load Balancing

Some RDF data would be accessed frequently over a period of time, and they are treated as hot data. These hot data may just happen to accumulate in one or a few storage servers. Then these storage servers are heavily loaded, while other storage servers are lightly loaded. This would seriously affect system performance.

In this paper, it uses a dynamic mechanism of replicating, storing and deleting the RDF data to achieve load balancing. The mechanism is based on the system query log. This log records the information of each query operation, including the query time, the query result and the storage server operated.

First, it finds out the current hot data and several light-loaded storage servers from the system query log. Then the hot data is copied to these light-loaded storage servers. The storage locations of these hot data in metadata are updated. When there is a query, it firstly obtains the data location from the metadata. When there are multiple storage locations, one is selected at random. Thus, the access to the hot data is shared by several light-loaded servers.

For the RDF dataset, *TotalRdfNum* represents the total number of RDF items. A_i represents the number of occurrences of the item *i*. Then, the hot degree of the data could be computed as follows:

$$HotDegree(i) = A_i \ / \ TotalRdfNum \tag{2}$$

If $HotDegree(i)$ exceeds a threshold *h*, item *i* is hot data.

The cold degree of the RDF data could be computed as follows:

$$ColdDegree(i) = 1 - A_i \ / \ TotalRdfNum \tag{3}$$

If $ColdDegree(i)$ exceeds a threshold *l*, item *i* has been cooled.

The workload of storage server *i* could be computed as follows:

$$WorkLoadDegree(i) = vp(j) \cdot cp(j) \tag{4}$$

$$vp(j) = S_j \ / \ TotalVisitNum \tag{5}$$

$$cp(j) = P_j \ / \ C_j \tag{6}$$

In formula (4), formula (5) and formula (6), *TotalVisitNum* represents the total number of accesses to the storage servers. S_j represents the number of accesses to the storage server *j*. The storage capacity of server *j* is C_j, and the current amount of the stored data on server *j* is P_j. The procedure of dynamic replication and storage for hot data is as follows:

Step1: Find out all the query operations in the latest time interval *T* from the query log.
Step2: Extract the RDF results from the query operations in Step1 and calculate *TotalRdfNum* and A_i.
Step3: Use formula (2) to calculate the hot degree of each RDF item and find out the hot data.
Step4: Extract the access information of the storage servers and use formula (4) to calculate the workload of each server.
Step5: Sort the servers from light to heavy according to the workload and choose the first *N* servers as the storage servers of hot data.
Step6: Copy the hot data to the *N* storage servers, and update the location information in the metadata.
Step7: Record the location information in Replication Log file in order to maintain the consistency between the source data and the replica.

The hot data is temporary. After a period of time, it may no longer be hot, which needs to be removed from the replica storage server.

The procedure of dynamic deletion for cool data is as follows:

Step1: Find out all the query operations in the latest time interval *T* from the query log.
Step2: Extract the RDF results from the query operations in Step1 and calculate *TotalRdfNum*.
Step3: Extract the hot items from the Replication Log file and calculate A_i for item *i*.
Step4: Use formula (3) to calculate the cold degree of item *i*.
Step5: Find out the replication storage locations of the cooled data from the Replicate Log file.
Step6: Delete the cooled data from the corresponding storage servers and delete the records of these cooled RDF data from the Replicate Log file.

4. EXPERIMENT

4.1 Experimental Settings

In the experiment, it uses five servers to form a cluster. The configuration is: Intel Xeon 2.2GHz CPU, 64GB DDR4 memory. One server plays the master role, while the others play the slave role. The operating system for each server is Ubantu16.04. The graph database is neo4j-enterprise-3.2.0. The metadata is stored in Redis 4.0. The relational database is Mysql 5.7. Each module of the system is implemented in JAVA. The parameter settings are as follows. The time interval in load balancing is *T*=600s. The hot threshold *h*=0.4. The cold threshold *l*= 0.9.

The experimental datasets include: SNOMED-CT RDF dataset for drugs [12], the dataset generated by LUBM [13] data generator, and the dataset of 1 billion RDF data provided by BSBM [14]. The datasets list is shown in Table 1.

Table 1. The experimental datasets

Dataset	Number of RDF items	Number of concepts
SNOMED-CT	800,000	103,843
LUBM	8,000,000	358,212
BSBM	1,000,000,000	8,847,439

4.2 Performance Comparison

In order to compare with the approach proposed in this paper, two RDF data storage scheme based on relational database are designed. The first one uses a simple three-column scheme. It only creates one table with three columns. The columns are corresponding to the subject, predicate, object of the RDF data, respectively. The second one uses a full index scheme. It creates six tables, each with the same structure as a simple three-column table. It adds different types of composite indexes to each table. For example, one order of the composite index could be: subject, predicate, object. Another order of the composite index could be: subject, object, predicate. Thus, it has a total of 6 combinations.

It conducts the comparison in three aspects: storage space, data query, and storage distribution. The storage space consumed by the dataset in each database is as shown in Table 2. GDB represents the graph database. RDB-1 represents the relational database using the first storage scheme. RDB-2 represents the relational database using the second storage scheme. It can be

seen that the storage space consumed by RDB-1 is the lowest. The storage space consumed by RDB-2 is about 2.1 times higher than that of the GDB. It is due to the massive storage of the indexes produced by the storage scheme. When the amount of RDF data is larger, the difference is more obvious.

Table 2. Storage consumed by datasets (GB)

Dataset	GDB	RDB-1	RDB-2
SNOMED-CT	0.1	0.05	0.2
LUBM	1.8	0.6	3.8
BSBM	80	14	180

For each dataset, 2 queries are designed for testing. All the query cases are represented by Q1-Q6.For the SNOMED-CT dataset, it conducts Q1 and Q2. Query Case Q1 has three triple matching modes. It just checks the name and sequence number of a drug. Query Case Q1 is depicted as follows:

Query Case Q1

```
PREFIX snomed:
www4.wiwiss.fu-berlin.de/drugbank/resource/targets/.
SELECT ?geneName, ?geneSequence
WHERE

   }?drugl snomed: genecardId ‘EPHA2’.
    ?drugl snomed: geneName ?geneName.
    ?drugl snomed: geneSequence ?geneSequence.}
```

Query Case Q2 has 4 triple matching modes. It checks the ID and name of the same type of a drug. Q2 involves the inference of RDF Data. Query Case Q2 is depicted as follows:

Query Case Q2

```
PREFIX snomed:
www4.wiwiss.fu-berlin.de/drugbank/resource/targets/.
SELECT ?genecardId,?geneName
WHERE
   }?drugl snomed: genecardId ‘EPHA2’.
    ?drugl snomed: sameType ?drug2
    ?drug2 snomed: genecardId ?genecardId.
    ?drug2 snomed: geneName ?geneName. }
```

For the LUBM dataset, it conducts Q3 and Q4. Query Case Q3 has a lot of input data and a high degree of selectivity. It involves a type and an attribute. Query Case Q3 is depicted as follows:

Query Case Q3

```
PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
PREFIX ub:
 http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
SELECT ?X
WHERE
{?X rdf:type ub:GraduateStudent.
   ?X ub:takesCourse
   http://www.Department0.University0.edu/GraduateCourse0}
```

Query Case Q4 involves 3 types and 3 attributes. It conducts a complex inferencing. Query Case Q4 is depicted as follows:

Query Case Q4

```
PREFIX rdf: http://www.w3.org/1999102/22-rdf-syntax-ns#
PREFIX ub:
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
   SELECT ?X,?Y,?Z
   WHERE
   {?X rdf:type ub:GraduateStudent.
    ?Y rdf:type ub:University.
    ?Z rdf:type ub:Department.
    ?X ub:memberOf ?Z.
    ?Z ub:subOrganizationOf ?Y.
    ?X ub:undergraduateDegreeFrom ?Y}
```

For the BSBM dataset, it conducts Q5 and Q6. Query Cases Q5 and Q6 are depicted as follows:

Query Case Q5

```
PREFIX bsbm-inst:
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm:
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?product ?label
WHERE {
  ?product rdfs:label ?label .
  ?product bsbm:product Feature bsbm-inst:Product Feature2595 .
  ?product bsbm:product Property Numeric1 "644" .
}

LIMIT 10
```

Query Case Q6

```
PREFIX bsbm-inst:
 http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/
PREFIX bsbm:
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/
PREFIX rdfs: http://www.w3.org/2000/O1/rdf-schema#
PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
SELECT DISTINCT ?product, ?typeLabel
WHERE{
   ?product bsbm:productFeature bsbm-inst:ProductFeature2595
   ?product bsbm:productPropertyNumericl "644".
   ?product rdf:type ?productType
   ?productType rdfs:label ?typeLabel
}
LTMTT 10
```

The query time of each query case is shown in Table 3.

Table 3. The query time of each query case (ms)

Query	GDB	RDB-1	RDB-2
Q1	82	4436	68
Q2	102	7987	96
Q3	256	63688	352
Q4	298	113245	410
Q5	1264	N/A	2276
Q6	1583	N/A	3124

As can be seen from the experimental results in Table 3, the query performance of the GDB and RDB-2 is obviously better than the RDB-1. It would involve multiple table join operations when conducting the inferencing query. The number of table joins gets bigger when the complexity of inferencing grows higher. Thus, the query performance of RDB-1 gets lower. When the dataset is small, the GDB approach and the RDB-2 approach have the similar query performance. However, when the data size becomes larger, the GDB approach is better than the RDB-2 approach. As

can be seen from the experimental results in Table 2, the storage space consumed by GDB is about 2.1 times than that of RDB-2 solution. Therefore, comprehensively speaking, the proposed storage approach based on the graph database is superior to the storage approach based on the relational database.

4.3 Effectiveness of Distributed Storage

This experiment uses five storage servers. After the distributed storage, the number of vertices and edges of each dataset on each storage server is shown in Table 4.

Table 4. The number of vertices and edges on each server

Datasets	SNOMED-CT	LUBM	BSBM
Server1 (vertex)	257,717,835	1384,592,345	84,378,382,472
Server2 (vertex)	214,756,382	1243,184,372	81,573,204,573
Server 3(vertex)	281,567,029	1178,326,904	88,362,584,273
Server 4(vertex)	273,756,187	1379,823,198	79,583,263,659
Server 5(vertex)	223,039,845	1227,402,046	85,271,606,725
Cross server edges	17,463	213,721	4,453,829

As can be seen from the experimental results in Table 4, the number of vertices and edges on each storage server is relatively uniform. Furthermore, the ratio of the number of edges across storage servers is 0.18, 0.19, 0.21, 0.23 and 0.24, respectively. This ratio is slightly smaller than [15], which has the results close to 0.2-0.3. This is due to the large amount of literal constants in the RDF dataset. In this paper, it treats the literal constants as the vertices, but when partitioning, they are directly assigned to the partition containing the resource vertices.

5. CONCLUSIONS

In this paper, a massive distributive RDF data storage method based on graph database is proposed. From the perspective of graph data model, it partitions the RDF datasets by heuristic greedy strategy according to dynamic data flow diagram mechanism. In the experimental section, it proves that graph database outperform RDBS in processing RDF data. Meanwhile, it verifies the effectiveness of the RDF dataset partitioning algorithm based on graph model. This work assumes that all the data corresponding to the query are stored in the same storage server, as it incorporates the RDF data set partitioning algorithm based on graph model. Thus, it could preserve good locality. Nevertheless, the assumption fails in most cases so it needs further improvement with distributed query [16].

6. ACKNOWLEDGMENTS

This work is sponsored by National Key R&D Program of China with project No.2016YFB08013 / sub-project No.2016YFB0801301 and is supported by Cyberspace Security Technology Laboratory of CETC.

7. REFERENCES

[1] Hendler, J. 2009. *Web 3.0 Emerging*. Computer, 42(1). DOI= 10.1109/MC.2009.30.

[2] Berners-Lee, T., Hendler, J., & Lassila, O. 2001. *The semantic web*. Scientific american, 284(5), 28-37.

[3] Neumann, T., & Weikum, G. 2008. *RDF-3X: a RISC-style engine for RDF*. Proceedings of the VLDB Endowment, 1(1), 647-659. DOI=10.14778/1453856.1453927.

[4] Abdelaziz, I., Al-Harbi, M. R., Salihoglu, S., & Kalnis, P. 2017. *Combining Vertex-centric Graph Processing with SPARQL for Large-scale RDF Data Analytics*. IEEE Transactions on Parallel and Distributed Systems. DOI= 10.1109/TPDS.2017.2720174.

[5] Zou, L., & Özsu, M. T. 2017. *Graph-Based RDF Data Management*. Data Science and Engineering, 2(1), 56-70. DOI=https://doi.org/10.1007/s41019-016-0029-6.

[6] Cuzzocrea, A., Buyya, R., Passanisi, V., & Pilato, G. 2017. *MapReduce-based algorithms for managing big RDF graphs: State-of-the-art analysis, paradigms, and future directions*. In Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (pp. 898-905). IEEE Press. DOI=10.1109/CCGRID.2017.109.

[7] Fernández, J. D., Martínez-Prieto, M. A., de la Fuente Redondo, P., & Gutiérrez, C. 2017. *Characterising RDF data sets*. Journal of Information Science, 0165551516677945.

[8] Koivisto, M., Laakkonen, P., & Lauri, J. 2017. *NP-completeness Results for Partitioning a Graph into Total Dominating Sets*. In International Computing and Combinatorics Conference (pp. 333-345). Springer, Cham. DOI=https://doi.org/10.1007/978-3-319-62389-4_28.

[9] Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., & Schulz, C. 2016. *Recent advances in graph partitioning*. In Algorithm Engineering (pp. 117-158). Springer International Publishing. DOI=https://doi.org/10.1007/978-3-319-49487-6_4.

[10] Karypis, G., & Kumar, V. 1999. *Parallel multilevel series k-way partitioning scheme for irregular graphs*. Siam Review, 41(2), 278-300. DOI=https://doi.org/10.1137/S0036144598334138.

[11] Karypis, G., & Kumar, V. 1998. *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*. Journal of Parallel and Distributed Computing, 48(1), 71-95. DOI=https://doi.org/10.1006/jpdc.1997.1403.

[12] Elkin, P. L., Brown, S. H., Husser, C. S., Bauer, B. A., Wahner-Roedler, D., Rosenbloom, S. T., & Speroff, T. 2006. *Evaluation of the content coverage of SNOMED CT: ability of SNOMED clinical terms to represent clinical problem lists*. In Mayo Clinic Proceedings (Vol. 81, No. 6, pp. 741-748). Elsevier. DOI=https://doi.org/10.4065/81.6.741.

[13] Guo, Y., Pan, Z., & Heflin, J. 2011. *LUBM: A benchmark for OWL knowledge base systems*. Web Semantics: Science, Services and Agents on the World Wide Web, 3(2-3).

[14] Bizer, C., & Schultz, A. 2008. Berlin SPARQL benchmark (BSBM) specification-v2. 0.

[15] Stanton, I., & Kliot, G. 2012. *Streaming graph partitioning for large distributed graphs*. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1222-1230). ACM. Doi=10.1145/2339530.2339722.

[16] Stolpe, A., & Halvorsen, J. 2017. *Distributed query processing in the presence of blank nodes*. Semantic Web, 8(6), 1001-1021. DOI=10.3233/SW-160250.