An Entity Based RDF Indexing Schema Using Hadoop And HBase

Fateme Abiri
Dept. of Computer Engineering
Ferdowsi University
Mashhad, Iran
Abiri.fateme@stu.um.ac.ir

Mohsen Kahani
Dept. of Computer Engineering
Ferdowsi University
Mashhad, Iran
Kahani@ferdowsi.um.ac.ir

Fatane Zarinkalam
Dept. of Computer Engineering
Ferdowsi University
Mashhad, Iran
Fattane.zarrinkalam@stu.um.ac.ir

Abstract— **Recent development of semantic web has opened new research to design search engines which organize and manage semantic data. The core of a search engine is the indexing system which consists of two main parts: data storage and data retrieval. With the increasing amount of semantic data, the most important goal expected from an indexing system is the ability to store large amount of data and retrieve them as fast as possible. In other words, having a scalable indexing system is one of the major challenges in semantic search engines. In this paper, a scalable method is presented to index the *RDF* data which utilizes *HBase* database, a *NOSQL* database management system, as its underlying data storage. *HBase* provides random access to massive data on the distributed framework of *Hadoop*, therefore, it can be a proper option for the management of the massive data. Further, due to the importance and popularity of the entity-based queries, a new schema based on a clustering algorithm is designed to effectively respond to this type of queries. The experimental evaluation shows that the proposed indexing system is effective in terms of improving scalability and retrieval of RDF data.**

Keywords-RDF Indexing; Agglomerative Clustering Algorithm; Entity Based Queries; NOSQL Database; HBase; Hadoop;

I. INTRODUCTION

The Semantic Web is an extension of the traditional web. In traditional web, the requests of users are expressed simply by keywords and search engines retrieve the indexed documents in which the keywords occurred [1]. On the other hand, the Semantic Web has been introduced to enable search engines to respond to complex requests of users based on their meaning. So, the relevant information sources have to be structured semantically.

To deal with this issue, w3c[1] has introduced a framework which describes the information resources in a semantic structure. This Resource Description Framework is briefly called *RDF*. *RDF* documents are in subject-predicate-object expression format and can be interpreted as a graph in which the subjects and objects are nodes and predicates are the edge of the graph. This simple model of representing knowledge can also be readable by machines and automated software agents to exchange knowledge distributed through internet.

To be able to response to complex queries, data should be effectively organized. Therefore, as a challenge, developers are interested in investigating different methods of organizing the semantic data.

[1] World Wide Web Consortium: www.w3.org

The process of organizing data is called data indexing. Each indexing system composes of two main components: data storage and data retrieval. The most important aim expected from such a system is the ability to store large amount of data and retrieve them as fast as possible. To achieve this goal, it's necessary to design a schema with the ability to scalable index data and respond to complex requests of users in desired time. The user requests should be expressed using SPARQL language. In fact, in an SPARQL query; users tend to extract sub graphs of entities from RDF graphs.

As argued in [2], there are five types of SPARQL queries which can be applied to RDF graphs. These types of queries are single triple queried, star shaped queries, entity based queries, path based queries, and graph based queries.

According to the recent statistics [3], the majority of the users in their queries, are looking for an entity with its specific attributes. So, the aim of the proposed method is to effectively response to star shaped queries with one path or actually the entity based queries and the single triple queries.

The structure of this paper is organized as follows. Related works are reviewed in section II. Then the paper proceeds to concentrate on details of the proposed system in section III. Section IV deals with the experimental evaluation of current system and finally the paper ends with the conclusion and future work of this paper.

II. RELATED WORKS

There are two key factors which should be considered in designing desired RDF management systems. The first factor is the indexing schema and the second one is the systems which should be considered to index and manage RDF data. In the following, these two factors will be explained.

A. RDF Indexing Schema

A big challenge of semantic web is storing data and then query processing to retrieve data in desired time. To handle this challenge, a lot of indexing methods have been introduce which can be divided into two groups; the schemas which interpret *RDF* documents as a graph and the schemas which interpret *RDF* documents as a set of triples.

The first group of schemas discussed in [2,4,5,6,7,8], try to analyze the structure of *RDF* graph and extract the relationships between nodes. Then, metadata obtained from the analyzed graphs and the triples are indexed together. In this method of indexing, because of using metadata of *RDF* graphs, the query processing is complex. The purpose of these methods is preventing the expensive operation in query processing by reducing proliferation of self joins and processing extra data. But these methods suffer from complex query processing [2] and in some cases don't support all types of queries [5].

The second group of these methods store directly *RDF* as a set of triples as argued in [9-18]. These methods were usually implemented using relational databases or native databases as storage layer. With the growth of *RDF* data, the overhead of self join operation is highlighted. Although these methods support all types of *SPARQL* queries, most of them suffer from the overhead of expensive self joins operation and processing extra data. An inefficient horizontal scalability of *RDBMS* prevents the indexing system from the scalable data storage and effective query processing which is discussed in [19] .

In Both groups of methods, the schemas suffer from the overheads of self joins and data redundancy in storages schemas.

B. *RDF Storage System*

An indexing schema, alone, cannot prove the capabilities of a *RDF* management system. Another factor which should be considered in designing a desired *RDF* management system is the *RDF* storage system in which data are indexed to the designed schema. The capabilities of this system should be considered in designing schema of *RDF* indexing.

There are different types of systems which are used in *RDF* management system. Among those system, *NOSQL* databases, relational databases, and native *RDF* management system are common ones. The relational databases are one of the mainstream storage systems which have been used [6,7,12]. Relational databases scale well only when the scaling happens on a single server node (vertically scalability). These systems are expensive with the cost of hardware in vertical scalability and are limited to extend system horizontally [19,20].

Considering *NOSQL* databases, the *NOSQL* systems are designed to solve the scalability of relational database [21]. These databases are successfully tested in famous companies such as Facebook [19] and Google [22]. *NOSQL* databases trade full ACID to give performance. The important feature of the mentioned systems is horizontal scalability capabilities and fast distributed processing of data [19, 23].

Moreover, there are other databases called native databases [2,5,11]. These databases are specially designed for triples. Thus, the query optimization in designing databases is simply done only for triple structure. It can improve the performance of indexing system. But designing a database from the base is a complex and difficult procedure. Therefore, implementation of this kind of system needs many experts in designing databases and particularly in distributing databases.

III. THE PROPOSED SYSTEM

In this section, after describing the required background, the proposed indexing system is introduced in detail.

A. *Background*

The aim of the proposed paper is investigating a method to manage *RDF* data with effective manner. Our proposed method is the scalable entity based *RDF* indexing by using *HBase* as a *NOSQL* database.

There are four types of NOSQL databases: Graph based, document based, key-value based, and the column family based databases which all these databases are candidates to be used in *RDF* management systems [21]. We have considered two points to select the desired storage system; firstly, the selected option should be free schema in order to simply map our proposed entity based schema to that; secondly, we need a computing distributed framework to perform the pre-processing of storing.

Considering the mentioned points, *HBase* (*Hadoop* database) with the free schema facility is the best option to be used in present method. *HBase* is open source and is supported completely by its developers. *HBase* has been tested in a big scale of thousand nods of the messaging system of Facebook. Moreover, the data availability is provided by the replication mechanism of *Hadoop* and meanwhile, *Hadoop* provide us the feasibility of distributed computing [19].

By using *HBase*, the challenges of most of *RDF* indexing such as scalability and the presence of *NULL*s are solved. Also, by using the ability of *HBase* in versioning the cells, we can store the multi value predicate in a one cell. The similar method which we called that single-table schema has been introduced in [24] with three other systems. They used *HBase* to solve these challenges. In the next section the details of the proposed *RDF* indexing system will be explained.

B. *Proposed System Architecture*

As shown in **Fig.** 1, the proposed architecture is composed of three main units: schema creation unit, *RDF* indexing unit, and query processing unit. Each of these units has several phases and is explained in the following subsections.

1) *Schema Creation Unit*

For creating schema (**Fig.** 1:1), this unit first looks for the attributes patterns and groups them by using a clustering algorithm, then in the next phase, the HBase table's schema is created to store RDF data.

As mentioned before the proposed method of indexing use the entity based schema. The most important reason is the high abilities of these types of schemas to response to entity based queries. Further, Because of the popularity of these types of quires (97 percent of user requests are star shaped queries [3]), some search engines are implemented for answering only these types of queries in desired time and scale [5]. Hence, the proposed indexing system in this paper is also on entity based schema.

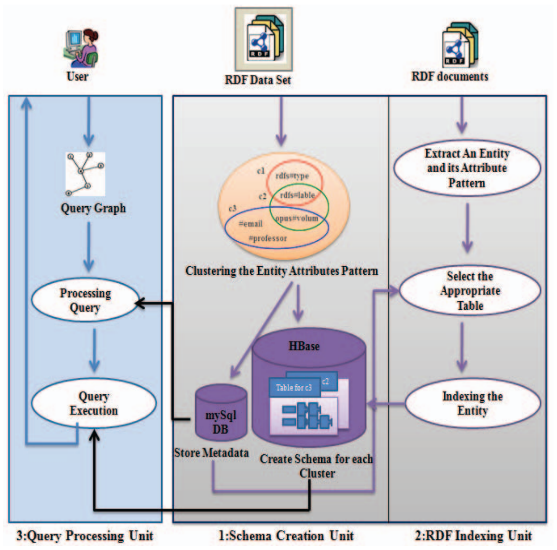


Figure 1. Proposed architecture.

In every data set, an entity is described by a set of attributes. In this paper, we call this set of attributes as a pattern. As shown in **Fig.** 2, pattern A={creator, title, abstract} with three attributes is used to describe an entity of *paper/5*.

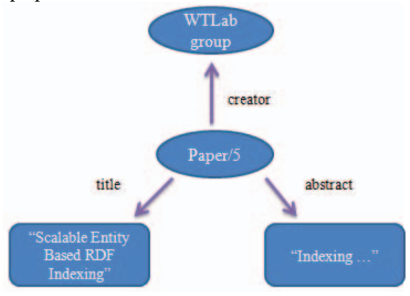
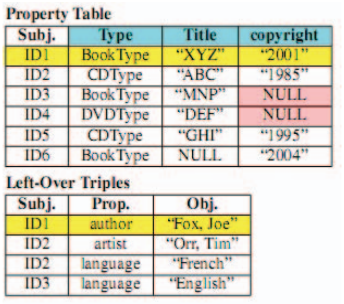


Figure 2. An entity with its attributes.

The purpose is grouping together the entities of data set with common attribute and storing them near each other. It should be mentioned that there are methods which operate in similar way [4,6]. These methods group the similar entities using a clustering algorithm. Since these methods usually use the relational databases, two types of tables are created. The main table is the property tables. For each cluster these tables are created. As shown in **Fig**. 3, the columns are the common attribute between the entities of a special cluster and the rows are the entities or actually subjects described by those attributes. The second types of tables are left-over triple table. This table are related to the attributes where are not present in any clusters. In fact these attributes have occurred less with the set of other attributes. So, entities based on these attributes may be stored in one table or in several tables.



Property Table

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID2	language	"French"
ID3	language	"English"

Figure 3. Entity based schema example [12].

The fundamental reason of creating two types of tables is reducing the overhead of saving *NULL* values in a property table. However, because of the semi-structured of *RDF* data, we can't completely eliminate *NULL*s due to the overhead of tables join. But when we reduce the number of tables in order to reduce the joins, the *NULL* values are increased naturally. The new methods are trying to introduce a clustering algorithm which compromise between the number of tables and *NULL* values [4]. Our proposed method is the scalable entity based *RDF* indexing by using *HBase* as a *NOSQL* database. In this method, however we will have *NULL* values, but by using the *HBase*, besides the scalability facility, the columns are created only for the attributes pattern of each row/entity. As a result, the overhead of *NULL* values is removed and we let the clusters overlap with each other. Therefore, there will be no need of a triple table and also no need of doing join tables to return the entities. Meanwhile, *HBase* sequential-read access pattern is faster than Random-read access pattern. By designing this schema the following goals should be achieved:

1: Storing the similar entities at near each other in a table to retrieve the requested entity quickly.

3: Reducing the search space by assigning a new table for each cluster.

2) *The RDF Indexing Unit*

In this unit, each document of the data set is processed separately to extract entities with their attributes pattern and their values. Then, according to the metadata stored in the creation schema unit, the most similar head cluster to the extracted entities is selected. In order to store the entity, the mapped *HBase* property table name is retrieved and the entity is stored in that table with its row-key structure.

3) *The Query Processing Unit*

In this unit, the user request is processed. In a request the user tends to find an entity with an attribute pattern (**Fig.** 4: a). So this attributed pattern is extracted from the query at first. Then, the head clusters which supported this pattern are extracted (**Fig.** 4: b). Thereafter, the system accesses to distributed cluster to retrieve data. We can speed up this phase using multi threading. So, we assign a thread to each *HBase* property table and every triple is responsible to retrieve requested data from its assigned table in desired time (**Fig.** 4: c).

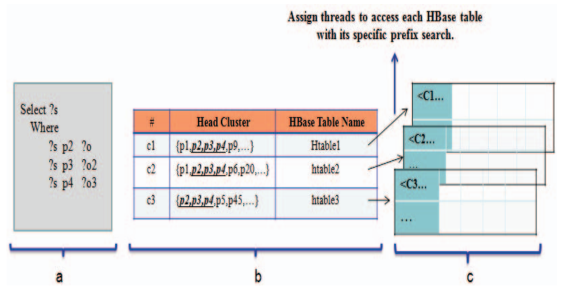


Figure 4. The stages of query processing. (a): Process the user query and extract its pattern. (b): Access to *RDBMS* to retrieve the head clusters which support the pattern. (c): Access to related tables by using threads which responsible to retrieve data.

As we see, to retrieve an entity the overhead of join operation is removed. However, by using the map/reduce programming model, the join operation can be performed on the *HBase* tables, but *HBase* by itself, doesn't support the join operation, Because it was developed to remove this overhead from the *RDBMS* tables.

IV. IMPLEMENTATION AND EXPERIMENTAL EVALUATION

The proposed system will be implemented and evaluated in this part. After describing the experimental setup, we will report the experimental results.

A. *Experimental Setup*

In this part, in subsection *one* we introduce the dataset. Subsection *two* deals with the evaluation metrics. Comparing schema and computational environment is discussed in subsection *three* and *four.*

1) *Data Set*

In this paper we use *LUBM* [2] as a standard dataset. *LUBM* has variety information about the university domain. In this evolution about 27 million triple (5GB) of *LUBM* is processed.

2) *Comparing scheme*

The proposed system has been compared with a single-table schema which is explained in [24] with three other indexing systems. The query processing of this system is implemented using Hive. But, here we have implemented this module using *HBase* functionalities. This method stores all entity in one *HBase* big table. The attributes are the columns and the subjects/entities are the row keys. So, in this method of indexing there isn't a creation schema unit. Entities after exacting from the data set are stored in one *HBase* big table.

3) *Evaluations metrics*

The three units of proposed system are evaluated separately by defining specific evaluation metrics for each of them. The evaluation metric of the creation schema unit is the time which it takes the system to create the schema. In other word, the time which it takes the system to extract the patterns, cluster them, store the metadata in *RDBMS* and create the *HBase* property tables. The evaluation metric of the indexing unit is the time of storing *RDF* data. The evaluation metric of query processing unit is the time which takes the system to respond to the queries. Obviously as the obtained time (particularly the time of responding to the queries) is less the *RDF* management system is more applicable.

4) *Computational Environment*

The proposed system is implemented in Java and apache *HBase* version of 0.94.11, *Hadoop* version of 1.2.1, and *MySQL* version of 6.0 are utilized. Regarding the *HBase* architecture, the distributed cluster has 6 nodes which one of them is both, master node of *HBase* and name node of *Hadoop*. All of six nodes using linux as an OS with 8-core *CPU* and 8 gigabytes *RAM*.

B. *Implementation and evaluation*

Experiment results are reported by each unit of the proposed system architecture separately as follows.

1) *Evaluation of Schema Creation Unit*

The schema creation phase point out the creating *HBase* tables with specific configurations and without any columns. While entities are inserting into tables, the columns are created for each row.

The single-table schema doesn't have this unit. In this unit we have grouped the patterns by a clustering algorithm. After extracting the patterns using map/reduce, we clustered the patterns. Meanwhile, the time which it took that each phase complete successfully, are shown in TABLE I.

TABLE I. THE EVALUATION OF SCHEMA CREATION UNIT (IN SECOND).

Data Set	Pattern Recognition	Clustering Algorithm	Schema Creation
LUBM	943	0.1	8

2) *Evaluation of Indexing Unit*

The time which took these systems to store entities is shown in **Fig**. 5. Considering our method of indexing is based on the entities;"Entity storing rate of *LUBM*" is the number of stored entities divided by the total amount of time spending for storing them. If we see the rate of storing entities, the indexing rate of the proposed methods is more than the single-table schema. The indexing rate of our system is 3112.6 entity/s but the indexing rate of single-table schema is 2980 entity/s. It is important to note that some of the previous systems are not entity based and they have reported "triple storing rate of *LUBM*" [9]. Hence, we have also decided to report it to simply compare our system with them. "triple storing rate of *LUBM*" is the number of stored triples divided by the total amount of time spending for storing them.

[2] *SWAT Projects - the Lehigh University Benchmark (LUBM)*: www.swat.cse.lehigh.edu/projects/*LUBM*/

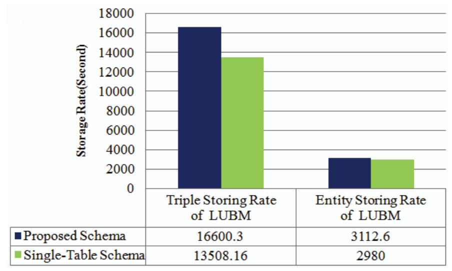


Figure 5. The evaluation of *RDF* indexing rate.

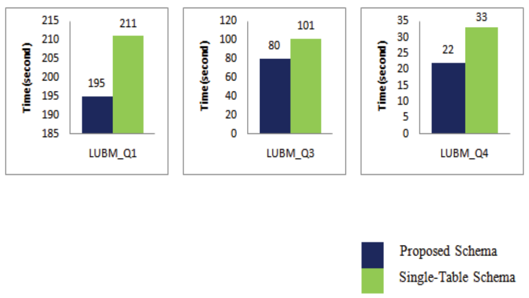


Figure 6. The results of executing *LUBM* queries.

3) *Evaluation of Query processing Unit*

The *LUBM* dataset has several standard queries. Since this dataset consider the reasoning and our proposed system doesn't consider this facility at now, we only can use three of the queries from 8 entity based queries of that.

Because of the overlapping between clusters, the requesting of an entity with a several attribute causes the system to access one or more *HBase* tables. So, the execution process of the queries is the same as each other without any join overhead. Owing to the attributes patterns of entities, the proposed schema has divided the *RDF* data in four *HBase* tables. But the single-table schema has only one table.

According to the statistics shown in TABLE II, during the processing of all three queries, our proposed method only access to one of four tables. However, It should be mentioned that the time of accessing to more than one *HBase* table is equal to the time of accessing to one table. This feasibility is provided by the *connection pool* function of *HBase* database.

TABLE II. THE GENERAL INFORMATION ABOUT THE QUERIES EXECUTED.

Queries	**Q1**	**Q2**	**Q4**
Number of data Returned	3	4	10
Number of Tables Accessed	1	1	1

Finally, the results of query execution are shown in **Fig.** 6. The new indexing method can execute all three queries with the more speed than single-table schema. The proposed system has the overhead of extracting structure which implemented optimally by using map/reduce programming model of *Hadoop* framework. But we suffer from the overhead of the extracting structure to achieve the below goals:

1). Storing the entities of a cluster at near each other and helping to retrieve them by their structure effectively. It's the result that HBase read data quickly by using sequential-read access pattern than Random-read access pattern [9].

2). Reducing the searching space by dividing entities according to their structure or in fact the attribute pattern. As a result, even if the selectivity of query is less, system will respond quickly by reducing the searching space.

V. CONCLUTION

To benefit from the massive growth of semantic web, it is necessary to organize data effectively. Hence, various methods have been introduced and the fundamental purpose of these methods is responding to the queries in desired time. In this paper we have divided *RDF* indexing methods in new categories with their strengths and weaknesses and accordingly, we introduced a new entity based method of indexing using *HBase* and *Hadoop* capabilities. To implement our method, we present a new architecture which has three units with different phases and procedure. To reduce the overhead of processing we use the map/reduce programming model of *Hadoop* except in clustering algorithm phase. The new method satisfies our expected results in query processing unit. By this method we remove the join operation and provide a facility to access *HBase* table precisely by accumulating metadata about the entities structure.

VI. REFRENCES

[1] S. Melink, and et al, "Building a distributed full-text index for the web", ACM Transactions on internet Technology, pages 217-241, 2001.

[2] T. Tran, G. Ladwig and S. Rudolph, "RDF Data Data Partitioning and Query processing Using Structure Indexes", IEEE Transactions Knowledge and Data Engineering, 2012.

[3] J. Gallego Fernández, M. Martínez-Priet and P. Fuente,"An empirical study of real-world SPARQL queries", In 1st International Workshop on Usage Analysis and the Web of Data (USEWOD2011) at the 20th International World Wide Web Conference (WWW 2011), 2011.

[4] J. Levandoski and M. Mokbel, "RDF Data-Centric Storage", In Proceedings of the IEEE International Conference on Web Services, pages. 911-918, 2009.

[5] R. Delbru ,Searching Web Data: an Entity Retrieval Model. Ph. D. thesis. National University of Ireland, 2010.

[6] K. Wilkinson, "Jena property table implementation", International workshop on Scalable Semantic Web Knowledge Base Systems (SSWS) at the International Semantic Web Conference(ISWC), 2009.

[7] H. MahmoudiNasab and S. Sakr, AdaptRDF: adaptive storage management for RDF databases, International Journal of Web Information Systems, pages 234-250, 2012.

[8] O. Udrea, A. Pugliese and V. Subrahmanian,"Grin: a graph based RDF index", AAI Conference of Artifical Inteligent, 2007.

[9] S. Fundatureanu, Scalable RDF Store Based on HBase. MS. D.thesis. Vrije Universiteit. Amsterdam. Netherlands, 2012.

[10] A. Owens and et al. ClusteredTDB: A clustered triple store for Jena , Tec. Rep, 2008.

[11] T. Neumann, and G. Weikum,”The RDF-3x engine for scalable management of RDFdata”, The International Journal onVery Large Data bases, pages 91-113, 2010.

[12] D. J. Abadi and et al, “Scalable semantic webdata management using vertical partitioning”, In Proceeding of theVeryLarge Data base, pages 1-12, 2007.

[13] C. Weiss, P. Karras and A. Bernstein, “Hexastore: sextuple indexing for semantic webdata management”, The International Journal on Very Large Data bases, pages 1008 –1019, 2008.

[14] V. Khadilkar, M. Kantarcioglu and B. Thuraisingham, ”Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store”, Thech. Rep, 2012.

[15] A. Harth and et al,”YARS2: A Federated repository for Searching and Querying Graph Structured Data”, In Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference, pages 211-224, 2007.

[16] N. Papailiou and et al, H 2RDF: Adaptive Query Processing on RDF Data in the Cloud, In Proceedings of the 21st International World Wide Web Conference (WWW2012 Demo Track), pages 16-20, 2012.

[17] A. Hogan, and et al. “Searching and Browsing Linked Data with SWSE: the SemanticWeb Search Engine”, The International Journal on Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier Science, pages 401-365, 2012.

[18] J. Sun, “Scalable RDF store based on HBase and mapreduce”, 3rd International Conference In Advanced Computer Theory and Engineering (ICACTE), 2010.

[19] L. George, HBase: The Definitive Guide, 1^{th}ed. New York: O'Reilly Media, 2011.

[20] ReadWrite, Is the Relational Database Doomed?, Available at: www.readwrite.com/2009/02/12/is-the-relational-database-doomed#awesm=~osqZ3qpcwLCwCq, [Accessed 1 January 2014].

[21] NOSQL DataBase (Updated 2014). Listof NOSQL DataBase, Available at: www.NOSQL-database.org [Accessed 1 April 2014].

[22] F. Chang and et al, “Bigtable: A distributed storage system for structured data”, In Proceedings of the 7th conference on usenix symposium on operating systems design and implementation, pages 205-218, 2006.

[23] F. Bugiotti, F. Goasdoué and Z. Kaoudi, “RDF Data Management in the Amazon Cloud” , Workshop on Data analyze in Cloud, 2012.

[24] P. Cudré-Mauroux and et al, NOSQL Databases for RDF: An Empirical Evaluation. In Proceedings of the 12th International Semantic Web Conference (ISWC). pages 310-325, 2013.