KKU Engineering Journal
http://www.en.kku.ac.th/enjournal/th/

Design and evaluation of a NoSQL database for storing and querying RDF data

Kanda Runapongsa Saikaew*[1], Chanuwas Asawamenakul[1] and Marut Buranarach[2]

[1]Department of Computer Engineering, Faculty of Engineering, Khon Kaen University, Khon Kaen, Thailand, 40002.

[2]National Electronics and Computer Technology Center, Pathum Thani, Thailand,12120.

Received May 2014
Accepted September 2014

Abstract

Currently the amount of web data has increased excessively. Its metadata is widely used in order to fully exploit web information resources.This causes the need for Semantic Web technology to quickly analyze such big data. Resource Description Framework (RDF) is a standard for describing web resources. In this paper, we propose a method to exploit a NoSQL database, specifically MongoDB, to store and query RDF data. We choose MongoDB to represent a NoSQL database because it is one of the most popular high-performance NoSQL databases. We evaluate the proposed design and implementation by using the Berlin SPARQL Benchmark, which is one of the most widely accepted benchmarks for comparing the performance of RDF storage systems. We compare three database systems, which are Apache Jena TDB (native RDF store), MySQL (relational database), and our proposed system with MongoDB (NoSQL database). Based on the experimental results analysis, our proposed system outperforms other database systems for most queries when the data set size is small. However, for a larger data set, MongoDB performs well for queries with simple operators while MySQL offers an efficient solution for complex queries. The result of this work can provide some guideline for choosing an appropriate RDF database system and applying a NoSQL database in storing and querying RDF data.

Keywords : Semantic web, RDF storage system, RDF database system, SPARQL query processing

1. Introduction

Currently the amount of web data has increased excessively. Its metadata is widely used in order to fully exploit web information resources. The Semantic Web is a Web of data that the World Wide Web Consortium has the vision to provide a common framework that allows data to be shared and reused across applications and enterprises. Thus, there is the need for the definition of the relations among data that allows a better and automatic interchange of data. Resource Description Framework (RDF), which is one of the fundamental building blocks of the Semantic Web, gives a formation definition for the interchange of data. It is a standard for describing web resources.

RDF data is in the form subject-predicate-object which is called triples. The subject describes the resource while the predicate is the relation or property between the subject and the object. For example, one way to represent the notion

* Corresponding Tel.: +66-43-362-160; fax: +66-43-362-160
Email address: krunapon@kku.ac.th

"The woman has the sweets" in RDF is as the triple: a subject denoting "the woman", a predicate denoting "has", and an object denoting "the sweets."

Many types of storage engines are designed and evaluate for triples. One of those types is a triple store which is a purpose-built database for the storage and retrieval of triples. Queries on these triples are in SPARQL, which is a language designed specifically to query RDF databases. The efficiency of RDF data analysis depends on the performance of RDF storage and query engine.

Traditional RDF database systems query data from native RDF stores or from relational database systems. The motivation for such native RDF-specific stores is that the relational model is not particularly suitable towards storage and retrieval of RDF data because RDF is a graph data model. However, relational database systems are equipped with mature optimization techniques for storing and querying data.

NoSQL database is another type of database that is not relational database and not use SQL to query the data. NoSQL database has the data model that can divide into four types which are document database using JSON data format, key-value database, column store database, and graph database. NoSQL database has different characteristics from relational databases, such as schema-free and replication support. The motivation for this approach includes the simplicity of design and the horizontal scaling for supporting big data.

Recently, NoSQL databases have been more successful than traditional relational database systems for the ability inprocessing big data on the cloud effectively [1]. In NoSQL databases,to gain performance, ACID (Atomic, Consistency, Isolation, and Duration), which is a set of properties that guarantee that database transactions are processed reliably, is sacrificed [2]. However, the advocates of NoSQL databases argue that they should rather enforce the triple of requirements including consistency (C), availability (A) and partitioning tolerance (P), shortly CAP [1].

In this paper, we aim to search for the answer of the question how to process web data quickly. Thus, we propose a method to exploit a NoSQL database, specifically MongoDB, to store and query RDF. MongoDB is chosen because it is one of widely used NoSQL databases. The system first invokes NoSQL API to retrieve MongoDB data in JSON format. Then, the JSON parser module converts JSON data to RDF data. We evaluate our design and implementation by using the Berlin SPARQL Benchmark, which is one of the most widely accepted benchmarks for comparing the performance of three RDF storage systems which include Apache Jena TDB (native RDF store), MySQL (relational database), and MongoDB (NoSQL database).

Benchmarking has been a core topic of RDF data management research. Bizer and Schultz [3] proposed the Berlin SPARQL Benchmark (BSBM) for comparing the performance of native RDF stores (Sesame, Virtuoso, Jena TDB, and Jena SDB), SPARQL-to-SQL rewriters (D2R Server and Virtuoso RDF Views), and relational database management systems (MySQL and Virtuoso RDBMS). The rewriting approach outperformed native RDF storage with the increasing dataset. The other important result was that relational database management systems were faster than the SPARQL-to-SQL rewriters. The authors of this related paper explained that RDF stores might not have a mature optimization technique as SQL query engines had. Our paper uses the BSBM benchmark to evaluate RDF storage systems but we also propose the approach to use and evaluate NoSQL database as a RDF data query processing system.

There has been some work on querying RDF data from NoSQL databases [4-6]. Cudre-Mauroux et al. [4] made the first attempt at characterizing and comparing NoSQL stores and native RDF stores for RDF processing. They used the Berlin SPARQL Benchmark and the DBpedia SPARQL Benchmark to evaluate and compare a native RDF store (4store) with four NoSQL databases which included Jena+H-Base, Hive+HBase, CumulusRDF, and Counchbase. All experiments were performed on the Amazon EC2 Elastic Compute Cloud infrastructure. Based on the experimental results, NoSQL systems, such as Jena+HBase, processed simple SPARQL queries more efficiently than native RDF stores, such as 4store. On the other hand, for more complex SPARQL queries requiring several many joins and filters, NoSQL systems took longer time than 4store. Although both this related work and our work compare NoSQL systems and native RDF systems, but our paper also evaluates the performance of a relational base database system as well.

Angles and Gtierrez studied the RDF model from a database perspective and compared it with other database models [5]. However, they did not implement and evaluate a graph database for storying and querying RDF data like we do. Lately, Bendar et al. [6] performed the comparison of RDF databases, NoSQL databases, and relational databases for the Semantic Web applications with their own developed benchmark. However, they did not provide the analysis the types of queries for which each database was suitable.

Sequeda and Miranker [7] chose to execute SPARQL queries on RDF representation of the legacy relational data by implementing the system called Ultrawrap. Ultrawrap encoded a logical representation of the database as an RDF graph using SQL views and a translation of SPARQL queries to SQL queries. To improve query execution time, detection of unsatisfiable conditions and self-join elimination could be applied to the SQL from the translations of SPARQL queries.

Alexaki et al. [8] presented the ICS-FORTH RDFSuite, a suite of tools for RDF validation, store,and querying. They proposed the design of a persistent RDF store (RSSDB) for loading resource descriptions in an Object Relational Database Management System (ORDBMS) by using RDF schema knowledge. They also presented RQL as a declarative language for querying both RDF descriptions and schemas. However, they did not compare their proposed system with other database systems and did not use a standard benchmark like BSBM.

Several researchers have attempted to design and develop RDF storage and query engine using relational DBMSs [9-11]. Harris et al. [9] proposed 3store as a RDF storage and query engine and extended it to support SPARQL query interface [10]. However, 3store had not been evaluated and compared with other systems [9-10]. Jena1 [11] and Jena2 [12] are popular Semantic Web programmers' toolkits that have been downloaded for several thousand times. Jena1 is an open-source project, implemented in Java, and available for download for free. Its core is the capability in manipulating RDF graphs. Jena2 was extended to support multiple and flexible presentations of RDF graphs and to provide a simple minimal list view of the RDF graph to the application programmers.

There are several works about scalable RDF engines for storing, indexing, and querying [13-16]. The main focus of Jena2 was to improve the performance and scalability due to these problems: too many joins, single statement table, reification storage bloat, and query optimization [13]. To address these issues, the Jena2 schema design supported a denormalized schema for storing resource URIs and simple literal values directly in the statement table. In

addition, to improve performance through locality and caching, Jena2 also supported the use of multiple statement tables.

Sesame [14] was one of the first architectures which its aim was for efficient storing and querying a large amount of RDF data. However, there were some unsupported operations, such as aggregates [15]. Also, implementing triple store directly in PostgresSQL was faster than that of Sesame's interfaces and SeRQL [15]. Abadi et al. [15] proposed the approach of vertically partitioning the RDF data. The results showed that vertical partitioning achieved similar performance to the property table technique proposed to reduce the number of self-joins.

The RDF-3X (RDF Triple eXpress) [16], designed and implemented from scratch specifically for the management and querying of RDF data, outperformed the previously best alternative [15] by one or two orders of magnitude.

The contributions of this paper are as following:1) applying MongoDB to store and query RDF data; 2) using the standard Berlin SPARQL benchmark to compare all three kinds of database systems: native RDF store, relational database, and NoSQL database. The analysis of the comparison can be a guideline for choosing an appropriate database system for different kinds of applications. For example, relational databases are suitable for applications with complex queries while NoSQL databases should be used for applications with simple queries.

2. Research methodology

In this section, we describe dataset description, query description, and experimental settings.

2.1 Dataset description

The dataset was adapted from the Berlin SPARQL Benchmark (BSBM) [1]. The BSBM consists of dataset generators and queries mix that can be used for comparing the performance of RDF storage and querying engines. The benchmark was built around an e-commerce use cases in which a set of products was offered by different vendors, consumers, and comments. The benchmark dataset consists of the following classes: product, product type, product feature, producer, vendor, review, and person. The BSBM has been chosen because it can simulate real-world enterprise application scenarios. In addition, the BSBM dataset is provided in the RDF data format, which simulates the Semantic Web data setting. In our tests, five different sizes of the dataset were generated and varied by the number of products: 1360, 2785, and 5544 products. The numbers of generated triples were 500K, 1M, and 2M triples respectively.

2.2 Query description

The query was adapted from the BSBM[1]. The query set consists of query for testing join, regular expression, comparison function, negation, sort result, skip result, and limit the result.

The description and characteristics of twelve BSBM queries are described as follows.

Query 1 : Find products for a given set of generic features (characteristics: simple filters, LIMIT, ORDER BY, and DISTINCT)

Query 2 : Retrieve basic information about a specific product for display purposes (characteristics: more than 9 patterns, and OPTIONAL)

Query 3 : Find products having some specific features and not having one feature (characteristics: simple filters, negation, OPTIONAL, LIMIT, and ORDER BY).

Quer 4 Find products matching two different sets of features (characteristics: simple filters, more than 9 patterns, LIMIT, ORDER BY, and UNION).

Query 5 : Find products that are similar to a given product (characteristics: complex filters, LIMIT, ORDER BY, and DISTINCT).

Query 6 : Find products having a label that contains a specific string (characteristics: complex filters and REGEX).

Query 7 : Retrieve in-depth information about a product including offers and reviews (characteristics: simple filters, more than 9patterns, and OPTIONAL).

Query 8 : Give me recent English language reviews for a specific product (characteristics: simple filters, more than 9 patterns, OPTIONAL, LIMIT, and ORDER BY).

Query 9 : Get information about a reviewer (characteristics: simple filters, and DESCRIBE).

Query 10 : Get cheap offers which fulfill the consumer's delivery requirements (characteristics: simple filters, LIMIT, ORDER BY, and DISTINCT).

Query 11 : Get all information about an offer (characteristics: unbound predicates and UNION).

Query 12 : Export information about an offer into another schema (characteristis: CONSTRUCT).

We translate the SPARQL query to MongoDB query by using functions that have the same effect, such as ORDER BY in SPARQL is translated to $sort operation in MongoDB to sort the result set. However, for functions that cannot be translated directly, we will use the functions that have a similar logic to get the same result, such as UNION in SPARQL is translated to $or and DISTINCT translate to $group operation instead to get the same result.

Table 1 and Table 2 give an overview of the chracteristics of the MongoDB queries corresponding to the BSBM benchmark queries.

Table 1 Characteristics of the Mongo DBQ1-Q6

Characteristics		Q1	Q2	Q3	Q4	Q5	Q6
Result manipulation	$sort	/		/	/	/	
	$limit	/	/	/	/	/	
	$skip				/		
	$unwind		/		/		
	$group	/			/	/	
Comparison Condition	Math($gt, $lt)	/		/	/	/	
	$or			/	/		
	$regex						/
	$all	/		/	/		
	$in					/	
	$ne					/	
	$eq		/				
	use _id		/			/	
Join(DBRef)			/				

Table 2 Characteristics of MongoDBQ7-Q12

Characteristics		Q7	Q8	Q9	Q10	Q11	Q12
Result manipulation	$sort		/		/		
	$limit		/		/		
	$skip						
	$unwind						
	$group				/		/
Comparison Condition	Math($gt, $lt)	/			/		
	$or						
	$regex						
	$all						
	$in						
	$ne						
	$eq	/			/		
	use _id	/		/		/	/
Join(DBRef)		/	/	/			/

2.3 Experimental settings

All tests were performed on a machine with the following specifications: Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz, and 4GB main memory size. The machine was running on Ubuntu 12.04.5 LTS (GNU/Linux 3.2.0-69-generic x86_64) and Java Development Kit version 1.7 with maximum heap size 1GB. In each test, we measured the average system response time. The following provides the version numbers and configurations of the tested database system.

1. Apache Jena TDB from Jena framework version 2.11.0 using RDF datasets in Turtle format, and SPARQL query language from the Berlin benchmark.

2. MongoDB version 2.6.4 with RDF datasets that are converted to json-ld format and parsed to MongoDB by using MongoDB query API.

3. MySQL version 5.5.38-0ubuntu0.12.04.1, with SQL datasets and queries of the Berlin SPARQL benchmark.

The performance metric used in the study is the number of queries executed per second which is the number of queries that have been executed successfully in one second.

3. Research results and discussion

The results of the performance evaluation of these systems (Jena TDB, MongoDB, and MySQL) are described in Table 3-5.

Table 3 The number of queries executed per second for 500K data set

500K	TDB	MongoDB	MySQL
Q1	*44.016*	**167.102**	132.610
Q2	78.177	*71.642*	**266.249**
Q3	*123.360*	**379.534**	240.884
Q4	*104.439*	**397.888**	303.361
Q5	*84.024*	229.700	**314.595**
Q6	*44.601*	220.212	**330.874**
Q7	*134.284*	212.973	**341.772**
Q8	*153.232*	**456.150**	320.177
Q9	*145.424*	**438.912**	389.298
Q10	*162.509*	**480.830**	406.432
Q11	*176.823*	**700.641**	395.720
Q12	*144.956*	**517.715**	395.796

In Table 3,MySQL outperforms MongoDB for queries Q2, and Q5-Q7 because Q2 has many operations such as join (DBRef), $eq to compare two values in the same document of MongoDB, and $unwind to separate the array object in MongoDB. As a result, MongoDB has to execute many steps and finish fewer queries in one second. For Q5 and Q7, in MongoDB, these queries require many sub-queries; thus, MongoDB is slower than MySQL. For Q6, MySQL performs very well since it has a sophisticated

optimizer for handling a regular expression. For queries Q2, and Q5-Q7, these queries require many joins, which a relational database system has an advantage because of its mature optimization techniques. MySQL also performs better than other systems for queries with complex filters and join.

Jena TDB performs poorly for all queries probably because it has not yet successfully implemented effective optimization techniques as relational databases have.

MongoDB performs better than other two kinds of database systems for most queries, especially queries with simple operators.

Table 4 The number of queries executed per second for 1M data set

1M	TDB	MongoDB	MySQL
Q1	*43.536*	**164.128**	131.352
Q2	77.010	*68.040*	**260.600**
Q3	*116.258*	**367.184**	235.679
Q4	*101.687*	**353.265**	300.617
Q5	*52.305*	168.181	**314.335**
Q6	*28.533*	137.625	**327.403**
Q7	*122.143*	209.694	**341.415**
Q8	*152.318*	**434.222**	314.225
Q9	*132.400*	**401.634**	377.606
Q10	*131.775*	337.155	**392.301**
Q11	*173.851*	**673.549**	392.568
Q12	*118.884*	**498.579**	393.642

In Table 4, The experimental results of all three database systems when the dataset is 1M are similar to those when the dataset is 500K except that for Q10, MySQL has higher throughput than MongoDB. It is likely that MySQL performs better for larger data sets due to the efficiency of indexing. On the other hand, for Q10, MongoDB uses $eq and join operation which require a larger amount of time for a larger result set.

Table 5 The number of queries executed per second for 2M data set

2M	TDB	MongoDB	MySQL
Q1	*40.377*	**158.492**	129.957
Q2	*62.858*	67.625	**260.721**
Q3	*94.764*	**290.004**	235.086
Q4	*97.984*	264.681	**301.325**
Q5	*36.108*	123.581	**312.722**
Q6	12.246	80.014	**320.887**
Q7	*96.904*	112.829	**337.532**
Q8	*106.941*	**423.473**	310.835
Q9	*94.691*	**397.862**	376.822
Q10	*105.837*	274.942	**385.128**
Q11	*121.693*	**633.726**	387.576
Q12	*90.216*	**496.011**	386.086

In Table 5, The experimental results when the dataset is 2M are similar to those when the dataset is 1M. However, for Q4, MySQL has higher throughput than MongoDB due to more efficient indexes to handle the queries with the high number of patterns of a larger volume of data. MySQL can handle most queries with more than 9 patterns (Q2, Q4, and Q7). On the other hand, MongoDB is suitable for queries with negation (Q3), unbound predicates (Q11), or some operators (Q9 and Q12).

MongoDB scales worst for queries with $eq operator. For all queries with $eq, which are Q2, Q7, and Q10, MongoDB does not perform well with a data size that is at least 1 MB. In addition, MongoDB is not suitable for queries with other kind of comparisons, such as Q5 and Q7 which use math($gt, $lt) and use _id. For queries with $regex operator, such as Q6, MongoDB also performs and scales poorly.

4. Conclusion

In this paper, we design the framework that uses MongoDB, a document based NoSQL database

to store and query RDF data. In addition, we compare the triple store (Jena TDB), NoSQL (MongoDB), and the relational database system(MySQL) by using the Berlin SPARQL Benchmark. Based on the experimental results, it has been found that Jena TDB performs poorly on most queries due to the ongoing optimization techniques. For most queries in a small data set, MongoDB finishes more number of queries than other database system. However, for a larger data set, MongoDB performs well for queries with negation, unbound predicates, or some simple operators. On the other hand, MySQL performs well for queries with many patterns and complex filters.

Future work should consider functions of the framework to automatically parse and import RDF data to MongoDB. There is also a need for a systematic system for the translation from SPARQL query to MongoDB query.

5. Acknowledgements

The financial support from Young Scientist and Technologist Programme, NSTDA (YSTP: SP-56-NT03) is gratefully acknowledged.

6. References

[1] Pokorny J. NoSQL databases: a step to database scalability in web environment. International Journal of Web Information Systems. 2013;9(1):69-82.

[2] Stonebraker M. SQL databases v. NoSQL databases. Communications of the ACM, 2010; 53(4):10-1.

[3] Bizer C, Schultz A. The berlin sparql benchmark. International Journal on Semantic Web and Information Systems (IJSWIS). 2009; 5(2):1-24.

[4] Cudré-Mauroux P, Enchev I, Fundatureanu S, Groth P, Haque A, Harth A, Wylot M. Nosql databases for rdf: An empirical evaluation. In :The Semantic Web–ISWC 2013:Springer Berlin Heidelberg; 2013. p. 310-25.

[5] Angles R, Gutierrez C. Querying RDF data from a graph database perspective. The Semantic Web: Research and Applications. Springer Berlin Heidelberg. 2005;346-60.

[6] Bednar P, Sarnovsky M, Demko V. RDF vs. NoSQL databases for the semantic web applications. In: Applied Machine Intelligence and Informatics (SAMI), IEEE 12th International Symposium; 2014. p. 361-4.

[7] Sequeda JF, Miranker DP. Ultrawrap: Sparql execution on relational data. Web Semantics: Science, Services and Agents on the World Wide Web. 2013;22:19-39.

[8] Alexaki S, Christophides V, Karvounarakis G, Plexousakis D, Tolle K. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In SemWeb; 2001. p. 310-25.

[9] Harris S, Gibbins N. 3store: Efficient bulk RDF storage. In: Proceedings of the1st InternationalWorkshop on Practical and Scalable Semantic Systems (PSSS'03); 2003. p. 1-20.

[10] Harri S, Shadbolt N. SPARQL query processing with conventional relational database systems. In: Web Information Systems Engineering–WISE 2005 Workshops: Springer Berlin Heidelberg; 2005. p. 23-44.

[11] McBride B. Jena. IEEE Internet Computing. July/August, 2002.

[12] Carroll JJ, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K. Jena: implementing the semantic web recommendations. In: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters: ACM; 2004. p. 74-83.

[13] Wilkinson K, Sayers C, Kuno HA, Reynolds D. Efficient RDF Storage and Retrieval in Jena2. In:SWDB; 2003. p. 131-50.

[14] 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 Berlin Heidelberg; 2002. p. 54-68.

[15] Abadi DJ, Marcus A, Madden S, Hollenbach KJ. Scalablesemantic web data management using vertical partitioning. In: VLDB; 2007. p. 411–22.

[16] Neumann T, Weikum G. The RDF-3X engine for scalable management of RDF data. The VLDB Journal. 2010;19(1):91-113.