Benchmarking Query Complexity between RDB and OWL

Chidchanok Choksuchat and Chantana Chantrapornchai

Department of Computing, Faculty of Science, Silpakorn University,
Nakhon Pathom, Thailand
cchoksuchat@hotmail.com, ctana@su.ac.th

Abstract. This paper describes how to benchmark relational database; RDB and web ontology language; OWL using query complexity concept on two difference speed machines. The domain was based on Hua Hin tourism. The purpose of this experiment was to benchmarking Semantic Web Knowledge Base Systems on relational perspective of query complexity. We use our tool to run on different speed machines to measure data complexity factors and the time of each activity. As a result, we conclude that if there is more data size and joined variables, the query complexity of RDB will increase but the ontology will reduce one. The advantage, the approach has been implemented and evaluated on improving the search engine of semantic web and reducing the expression complexity on a real archive.

Keywords: Query complexity, Ontology, OWL, RDF, RDB.

1 Introduction

The semantic web is an emerged technology. It was based on a representing, querying, and applying rules to data. The set of core standards are RDFS for structuring, RDF for representation, OWL for structuring and reasoning and SPARQL for querying. Differences from the relational databases are queried by SQL Language. The SPARQL query language [7] and protocol for RDF [10] are used as a standardized query API for providing access to datasets within enterprise settings and on the web. They aim at capturing domain knowledge and provide a commonly agreed understanding of a domain, which may be reused, shared, and run across applications and groups.

Tourism is a leading industry in the e-business. So, many projects with semantic web are already available [3], [4], [11] and [13]. The tourism enterprises will be interested in the use of ontology if they are evaluated well enough. So, the development of benchmarking ontology is an important task.

This paper contains the Hua Hin IT tourism domain covering benchmark query complexity between RDB and the knowledge base (KB) of semantic web represented by RDF/OWL. The benchmark was designed in accordance with two goals:

1. The benchmark allows comparing SQL and SPARQL query complexity across the same domain.

T.-h. Kim et al. (Eds.): FGIT 2010, LNCS 6485, pp. 352–364, 2010.

2. The benchmark is designed to measure SPARQL query performance against of OWL data.

This article makes the following hypotheses to the field of benchmarking Semantic Web technologies.

1. It complements the field with the tourism case driven benchmark.

2. It provides guidance to the developers by applying the benchmark to measure and improve the semantic web search engine by reduced the query complexity of the original RDB dataset.

The rest of the paper is structured as follows: Section 2 gives an existing benchmarks overview for Semantic Web technologies. Section 3 describes the benchmark design. Section 4 experimental presents the results of an experiment comparing the performance between RDB and OWL. Section 5 discusses the paper and outlines our next steps. Section 6 concludes the paper.

2 Literature Review

2.1 Benchmarking for RDB

Since a big part of web content is stored in RDB, the data storage becomes important. There are many types for benchmark about RDB in various domains.

In 1993, Gray [9] explained the benchmarks for DB and transaction systems. The quantitative comparison starts with the definition of a benchmark or workload that was measured as a transaction per second metric.

About the tourism domain, in 1998, a method of the performance evaluation of an information system [2] was presented in the evaluation of response time for a tourist agency's service system, result in a workload and concern to improve the system's behavior.

2.2 Benchmarking for RDF/OWL

T. Berners-Lee presented the future web concepts known as “Semantic Web” [15]. The purpose was to enable machines to comprehend semantic documents and data that are enriched by the conventional.

A key challenge for the Semantic Web is to acquire the capability to effectively query large KBs. There will be several competing systems. The benchmarks are needed that will objectively evaluate these systems. Development of effective benchmarks in an emerging domain is a challenging endeavor. For examples, LUBM: a benchmark for OWL knowledge base systems [16]. It was demonstrated with an evaluation of two memory based systems and two systems with persistent storage. It featured university domain ontology, synthetic OWL data scalable to an arbitrary size, fourteen extensional queries representing a variety of properties, and several performance metrics.

In 2007, Yuanbo et al. [8] explained to a requirements driven framework for benchmarking semantic web knowledge base systems (SWKBSs). Two major contributions were made to provide a list of requirements and organize collection of techniques and tools needed to develop such benchmarks.

In 2008, SP2Bench [12] was the language-specific SPARQL performance benchmark. It was settled in the DBLP scenario and comprised both a data generator for creating arbitrarily large DBLP-like documents and a set of designed benchmark queries. They applied existing engines as proof of concept and discussed the strengths and weaknesses from the benchmark results.

In 2009, the Berlin SPARQL Benchmark (BSBM) [5] was used for comparing the performance of native RDF stores with the performance of SPARQL-to-SQL rewriters across architectures based on e-commerce use case. The benchmark query mix emulated the search and navigation. The results of a benchmark experiment comparing the performance of four popular RDF stores (Sesame, Virtuoso, Jena TDB, and Jena SDB), two SPARQL-to-SQL rewriters (D2R Server and Virtuoso RDF Views) and two relational database management systems (MySQL and Virtuoso RDBMS).

2.3 Query Complexity Theory

Regarding to query complexity evaluation in tourism field, Abraham [1] presented the query evaluation along with business view that should not use much more of the mathematic formula and should use in tourism IT domain. The advantage of semantic web and capability of OWL reasoning in DL knowledge base is <T, A>. T is Terminological Box (TBox) constitutes the vocabulary of an application domain. A is Assertional Box (ABox) contains real world assertions of instances in vocabulary terms.

The Vardi's complexity concept [17] used ABox because ABox was viewed like RDB. It was important for the query evaluation model that was designed from KB. It meant the data model of logical query complexity itself was evaluated, rather than individual query languages. A more in-depth mathematical analysis was required for the query optimization issues that were covered by Calvanese. It takes use of DBMS techniques for both data representation, i.e., ABox assertions, and query answering via reformulation into SQL. Notably, in this case, the data complexity (ABOX size) of conjunctive query answering over ontologies is the one of First Order Logic (FOL) queries over DBs. The problematic LOGSPACE boundary was characterized. The fundamental results were presented on the data complexity of query answering in DLs. In particular, the FOL-reducibility boundary of the problem was concentrated. Query answering was no longer expressible as a FOL formula (SQL query) over the data.

There are three ways to measure the complexity of queries of evaluating queries in a specific language over a database. 1) The data complexity was given as a function of the size of the databases. 2) The expression complexity was given as a function of the length of the expressions. 3) The expression complexity was given as a function of the combined size of the expressions and the databases. It turns out that combined complexity is pretty close to expression complexity.

Vardi's definition for the complexity measure:

Definition 1: Let φ be a sentence of size s (a sentence represents a query). The φ has at most s variables. In order to evaluate φ on a database of size n, it suffices to cycle through at most n^s possible assignments of values from the database to the variables. Query complexity can be defined using formal logic as:

$$\exists\varphi\ ((\varphi \rightarrow s) \wedge (s \equiv n^s)) \quad (1)$$

For some sentence φ, the sentence has a size s, and s equals the number of possible values to variable assignments from the database that may be assigned to φ. Complexity of φ can therefore be expressed as the function: $s \equiv n^s$. The value of s (query complexity) can then be obtained by simply calculating: n^s.

Theorem 2: Query answering in DL is FOL reducible and therefore is in LOGSPACE with respect to data complexity [6].

3 Proposed Method

3.1 Methodology

This experiment makes the following procedure to the field of benchmarking.

1. We set up the dataset in RDB and the ontology in OWL type specified over Hua-Hin tourism. They contain the class of Attraction, Accommodation, Category, Rate, Facility, Location and Classification.

2. Set the difference levels of the complexity of conjunctive query: constant values, joining between sub-domains and variables and both of constant value and joined in the query expressions.

3. Prepare three forms of the queries: conjunctive query, SQL statement and SPARQL statement.

4. Use our tool for measuring query complexity by cover the Vardi's definition and Calvanese's theorem. We measured the query complexity both relational and knowledge-base models.

What we will investigate is as follows:

1.The results of query complexity evaluation include: number of terms, number of the first expression answers, number of the second expression answer, calculate query complexity of all expressions, degree of query complexity and percent of OWL reduced the degree of query complexity RDB.

2. The usage time of 1) on different speed machine.

3.2 Prepare the Dataset

The benchmark is settled in the Hua Hin tourism portal scenario. Figure 1 gives an overview of the dataset and the properties of each class. For RDB, the Accommodation domain was based on the structure of a normalized relational data model of http://www.huahin.go.th (2009).

After that, we created the ontology by Protégé 3.3.1 to Hua Hin tourism OWL file for SPARQL query. The DL expressivity of this ontology is SOIN (D). In OWL, classes can be described by a class identifier and can benchmark for classes [14]. The group of class and class hierarchies' benchmark contains ontologies that describe classes and class hierarchies. These ontologies include classes that are a subclass of value restrictions, cardinality restrictions on properties, and class intersections. In this group, vocabulary terms of both RDF(S) and OWL2 are used: rdfs:subClassOf, owl:Class, owl:Restriction, owl:onProperty, owl:someValuesFrom, owl:allValuesFrom, owl:cardinality, owl:maxCardinality, owl:minCardinality, owl:intersectionOf.

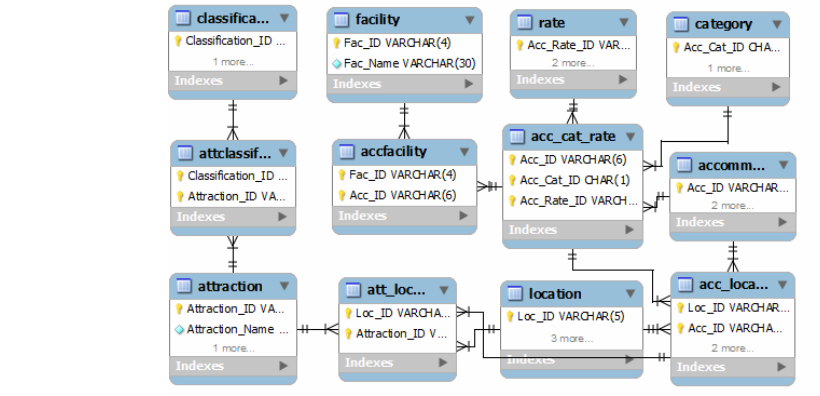

Fig. 1. The data model structure

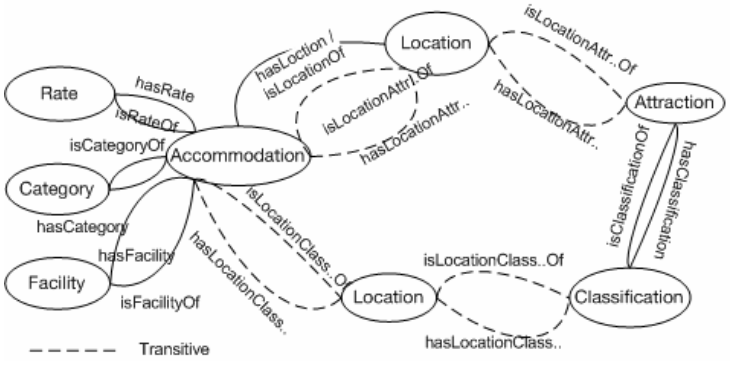

Fig. 2. OWL Ontology graph of Hua Hin tourism

In our OWL, sub-classes were use under the Accommodation class. For examples,

```
<! -- http://www.owl-ontologies.com/HuaHinProj.owl
#Category-Bangalow -->
<owl:Class rdf:about="#Category-Bangalow">
    <owl:equivalentClass><owl:Restriction>
    <owl:onProperty rdf:resource="#hasCategory"/>
    <owl:hasValue rdf:resource="#Category_Bangalow"/>
    </owl:Restriction></owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Accommodation"/>
    <owl:disjointWith rdf:resource="#Category-
    GuestHouse"/>
    <owl:disjointWith rdf:resource="#Category-Hotel"/>
    <owl:disjointWith rdf:resource="#Category-Resort"/>
</owl:Class>
```

The overview of Hua Hin tourism ontology graph shows in Figure 2. The OWL contains the class, properties and transitive properties between classes.

3.3 Selection the Queries

The key to this research was the selection of queries that accurately reflects the structure of web queries. Since this is tourism domain from Muang Hua Hin Municipality (version 2009), the user can query the accommodation by select from category and review the destination by the location. Therefore we set up the query in select case with joining complexity. This experiment started from a basis of level-1 to level-5 that was conducted with five conjunctive queries.

Table 1. Selection of Queries

No.	Size	Basic Domain	Other Domain	Depth
1	+	/	-	0
2	++	/	Location	1
3	+++	/	Location, Attraction	2
4	+++	/	Attraction, Classification	2
5	++++	/	Location, Attraction, Classification	3

The queries used in the experiment are presented in Table1. It was derived by the level, data size, domain, number of joined and ontology graph depth. In the detail, we represent the conjunctive query as an ontology concept.

Test 1. Set the basic concept include the domain of Accommodation with Category, Rate and Facility. Select the Accommodation with constant value. Return the name of Accommodation from constant Rate, Category and Facility.

```
((Category ⊓  ( ∃ hasRate{ Room  Rate })
           ⊓  ( ∃ hasFacility{ Facility A })
           ⊓  ( ∃ hasFacility{ Facility B })
           ⊓  ( ∃ hasFacility{ Facility C }))
```

Test 2. Select the basic concept joined with the Location and constant values. Return the name of Accommodation from constant Rate, Category, Facility and Location.

```
((Category ⊓ ( ∃ hasRate{ Room Rate })
           ⊓ ( ∃ hasFacility{ Facility A })
           ⊓ ( ∃ hasFacility{ Facility B })
           ⊓ ( ∃ hasFacility{ Facility C })
           ⊓ ( ∃ hasLocation{ Loc_Name }))
```

Test 3. Select the basic concept joined with the Location, Attraction and constant values. Return the Accommodation name from constant Rate, Category, Facility, Location and Attraction.

```
((Category Π ( ∃ hasRate { Room Rate })
           Π ( ∃ hasFacility { Facility A })
           Π ( ∃ hasLocation { Loc_Name })
           Π ( ∃ hasAttraction { Attraction A })
           Π ( ∃ hasAttraction { Attraction B }))
```

Test 4. Select the basic concept joined with the Location, Classification, and constant values. Return the Accommodation name from constant Rate, Category, Facility, Location and Classification.

```
((Category Π (∃ hasLocation { Loc_Name })
           Π (∃ hasRate { Room Rate })
           Π (∃hasFacility { Facility A })
           Π (∃ hasFacility { Facility B })
           Π (∃ hasClassification {Classification A}))
```

Test 5. Select the basic concept joined with the Location, Attraction, Classification, and constant values. Return the Accommodation name from constant Rate, Category, Facility, Location, Attraction and Classification.

```
((Category Π(∃ hasLocation{ Loc_Name })
           Π(∃ hasRate { Room Rate})
           Π(∃ hasFacility { Facility A })
           Π(∃ hasFacility { Facility B })
           Π(∃ hasAttraction {Attraction A })
           Π(∃ hasClassification { Classification A })
           Π(∃ hasClassification { Classification B }))
```

3.4 Measuring Query Complexity

For measuring query complexity, we used Vardi's definition as a basis for the complexity measure. Then we used Calvanese's theorem to concern about the cycle times of the actual degree of a query's computational complexity in LOGSPACE.

3.5 The Machine Specification

The experiments ran on two different speed machines as follows:

Machine#1: The experiment was conducted on a processor: Intel (R) CPU T2050 1.60 GHz; 798 MHz; memory: 0.99 GB hard disks: 80GB 32-bit Operating System running Window XP Professional.

Machine#2: The experiment was conducted on a processor: Intel (R) Core(TM) i5 CPU M430 2.27GHz; memory: 4GB hard disks: 320GB 64-bit Operating System running Window 7 Home Premium.

Software were installed in 2 machines: Apache Web Server 2.2.8, MySQL Database 5.0.51b, phpMyAdmin Database Manager 2.10.3. Protégé 3.3.1.RacerPro 1.9.0 reasoner. Java 1.6.0_18 was used.

4 Experiment Results

The experiment result was represented in Table2. We measure the factors include:

1. # Terms: Number of terms from conjunctive query.
2. # Joined: Number of equi joined between the tables.
3. Xs: the first parameter from Vardi's concept.
4. Vs: the second parameter from Vardi's concept.
5. Query complexity: Xs* Vs.
6. # Answer: the number of the answers of query.
7. Complexity Ranking: the depth of OWL graph.
8. Degree of query complexity in LOGSPACE.
9. Percent of reduced the degree of query complexity.
10. The ratio of the #answers and query complexity.
11. Usage time in millisecond on machine#1.
12. Usage time in millisecond on machine#2.
    (Usage time = load time + execution time)

Table 2. Results of benchmarking query complexity

Measuring	RDB1	OWL1	RDB2	OWL2	RDB3	OWL3	RDB4	OWL4	RDB5	OWL5
1) #Terms	7	6	8	7	9	7	9	7	11	9
2) #Joined	0	0	1	0	2	0	2	0	3	0
3) X^s	2	2	2	2	4	2	5	2	6	5
4) V^s	-	-	4	-	5	-	222	-	1110	-
5) QC	2	2	8	2	20	2	1110	2	6660	5
6) #Answer	2	2	2	2	2	2	2	2	5	5
7) Ranking	-	-	1	1	1	2	1	2	1	3
8) Deg QC	0.30	0.30	0.90	0.30	1.3	0.30	3.0	0.30	3.8	0.70
9)%Reduce of QC	0		67		77		90		82	
10) #Ans/QC	1.00	1.00	0.25	1.00	0.10	1.00	0.00180	1.00	0.00075	1.00
11) M#1(ms.)	23	4.3	24	5.4	24	5.7	26	6.8	48	8.8
12) M#2(ms.)	9.5	4.8	11	4.9	12	5.7	19	5.8	32	6.6

5 Result Analysis

We analyzed the result by the issues as follows:

Issue1: About usage time of RDB and OWL. The machine#2 can reduce usage time more than the machine#1. And the usage time of OWL less than the usage time of RDB in both machines. The differences of the usage time of OWL in the both machines less than the usage time of RDB. That means whether speed of machines, OWL still use less time than RDB clearly.

Issue2: The number of terms and joins between RDB and OWL are related as Figure 4. The RDB and OWL expression terms increase along with the more depth of graph. But the amount of OWL expression terms still increase less than RDB terms because sub-class in OWL can reduce the expression terms. We used the subclass of Accommodation class in OWL.

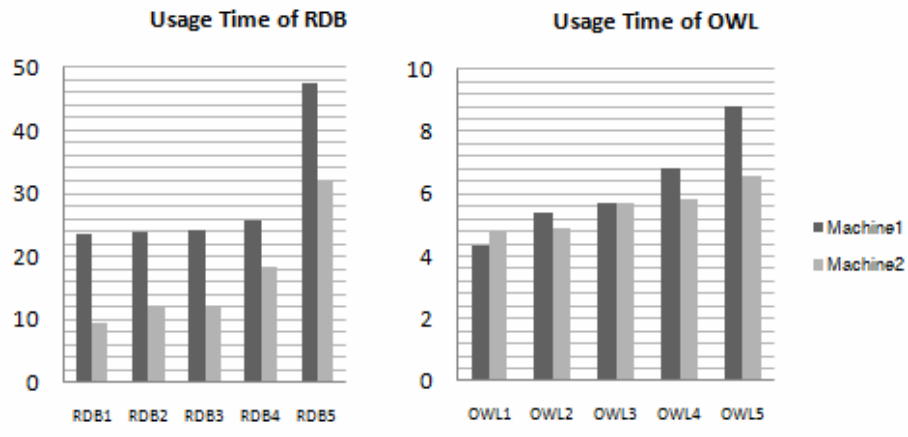

Fig. 3. Comparison between usage times of RDB and OWL

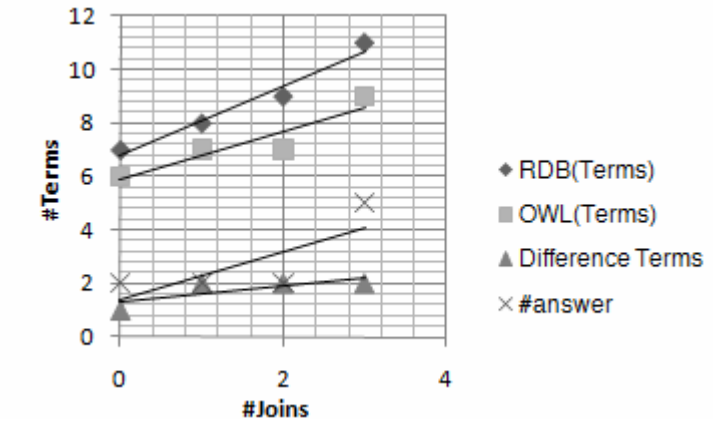

Fig. 4. Number of terms and joins of RDB and OWL

When the SPARQL were queried, it can be found out within the sub-class constantly. We can imply the OWL expression by subclass terms instead the terms of class. We can show you clearly by the conjunctive query. For example, Conjunctive query of OWL-2:

```
Q(X)<-Category-Bangalow(X) ∧ hasLocation(X,A) ∧
hasRate(X,B) ∧ hasFacility(X,C) ∧
hasAccommodationFacility(X,D) ∧
hasAccommodationFacility(X,E) ∧A=HuaHin-Takiab Road ∧
B= Room_Rate_2 ∧ C =Beach ∧ D= Refrigerator ∧ E= Air
Conditioning
```

Whereas the conjunctive query of RDB-2 have to join between Accommodation and Category classes as:

```
Q(X)<- Accommodation(X) ∧ hasLocation(X,A) ∧
hasRate(X,B) ∧ hasCategory(X,C) ∧
hasAccommodationFacility(X,D) ∧
hasAccommodationFacility(X,E) ∧ hasFacility(X,F) ∧
A=HuaHin-Takiab Road ∧ B= Room_Rate_2 ∧ C=
Category_Bangalow ∧ D=Beach ∧ E= Refrigerator ∧ F= Air
Conditioning
```

This is the reason why the number of terms of OWL less than RDB terms in the Table 2.

In addition, by study the graph in Figure4, we can see the difference between the terms of RDB and OWL slight wider, when the number of terms and joins increase. Because transitive properties were add in OWL. For instance, the use of transitive property hasLocationAttraction in the Accommodation ontology eliminated the number of joins for a relational model, and also reduced the number of query terms. In the next line, you can see the Accommodation class connects Location class with hasLocation property. Location class connects to Attraction class with hasAttraction Property.

Accommodation →Location →Attraction

We can formulate easier if we use transitive property hasLocationAttraction to connect between Accommodation and Attraction as below:

Accommodation →Attraction

For the other transitive properties, we show in Figure 2. As a result, OWL3, OWL4 and OWL5 can reduce the expression terms from relational model. In addition, the transitive property use of reduce the query complexity as well.

Issue3: the Comparison of Query Complexity of RDB and OWL. By studying Figure 5, we can see the difference query complexity values between RDB and ontology with the same data size. The RDB represents by SQL query language, find out the answer from the joins among the table, data complexity will increase the query complexity follow by the data size. On the contrary, the OWL represents by SPARQL query language can query the answer along the transitive property and subclass without joins. The OWL query complexity does not increase that much.

Issue4: The percentage of OWL implied the reduction in the query complexity. Figure 6 shows the reduction in percent of OWL query complexity degree to that of RDB.

We can see the growth of OWL reduced the degree of query complexity from RDB when the query complexity increases. In the small data complexity case such as query 1, we cannot see the difference. But for more query complexity, the percent of OWL reduced the query complexity degree from RDB more than 80%.

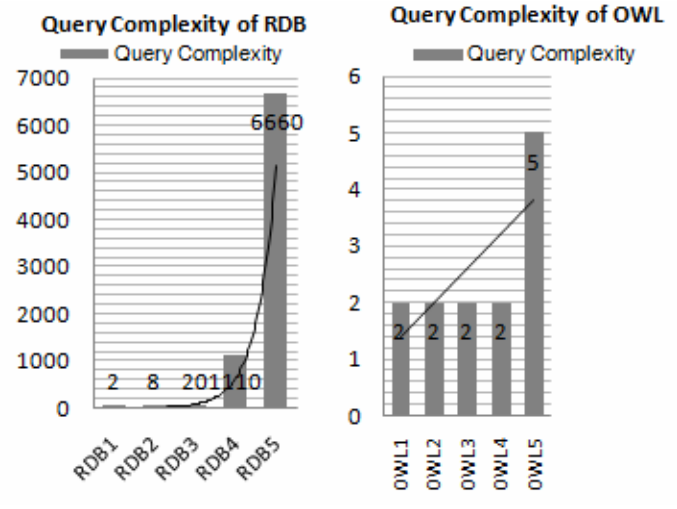

Fig. 5. Comparison of Query Complexity of RDB and OWL

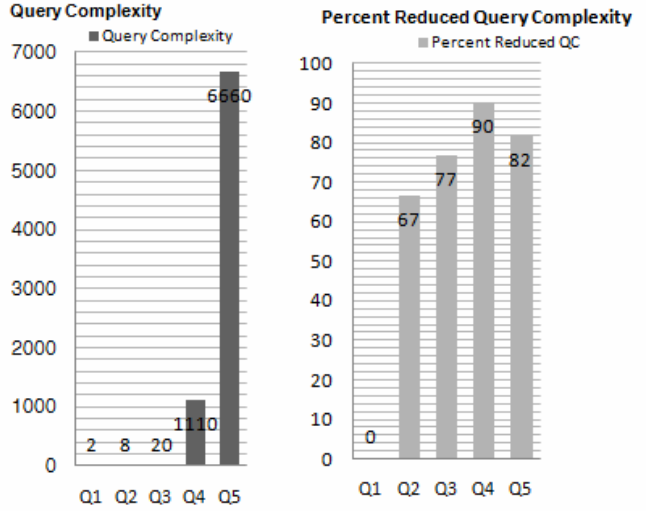

Fig. 6. Percent of reduction of OWL on the query complexity from RDB

Issue5: The ratio of number of the query answer to query complexity. We can see clearly in RDB that the query used to find the answer was much more complex. Because the number of answers were equal to 5 and query complexity as 6600. The ratio was less than 0.00076. This value was compared with the ratio equal to 1 from OWL side. When there is a small number of the answer, it is easier to find.

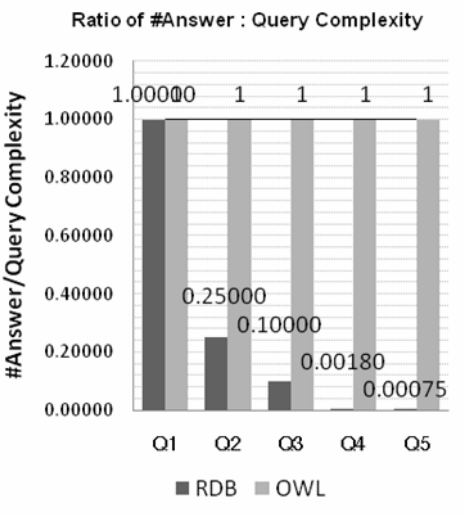

Fig. 7. The ratio of number of the answers to query complexity

6 Conclusion

Overall, we proposed the method of benchmarking query complexity between RDB and OWL. That can separate to the overview; measuring the usage time by two different speed machines. It was clear that the faster 4GBRAM machine#2 could run RDB and OWL in less usage time than the other one. The gap of time in RDB fell clearly. But in the OWL case, the Machine#2 dropped a few usage times for the small data. For the large amount of data, it tended to decrease clearly.

We also evaluated query complexity in five level queries. The RDB represented by SQL language that could query by joins between the tables. As the result, the more number of joins, the more query complexity. Whereas OWL ontology represented by SPARQL language. There were the sub-class hierarchy and transitive properties which used to reduce the query complexity. Thus, in the future work, we can use them continuously to improve the search engine in Semantic Web technologies.

Acknowledgments. This experiment is the part of thesis: Improving search engine using semantic web case study Hua-Hin tourism information. Thank you to the Muang Hua Hin Municipality for giving the http://www.huahin.go.th (2009) database structure.

References

1. Abrahams, B.: Tourism Information Systems Integration and Utilization within the Semantic Web (2006), http://wallaby.vu.edu.au/adt-VVUT/uploads/approved/adt-VVUT20070514.125504/public/02Chapter1-3.pdf

2. Zgrzywa, A.: The evaluation of the response time for a tourist agency's service system. Information and Software Technology 40, 37–44 (1998)
3. Legrand, B.: Semantic Web Methodologies and Tools for Intra-European Sustainable Tourism. JITT (2004)
4. Robert, B., Christina, F., Birgit II, P., Christoph, G., Hannes, W.: Covering the semantic space of tourism: an approach based on modularized ontologies. In: Proceedings of the 1st Workshop on Context, Information and Ontologies, ACM, Heraklion (2009)
5. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems - Special Issue on Scalability and Performance of Semantic Web Systems 5, 1–24 (2009)
6. Calvanese, D., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of KR 2006, pp. 260–270 (2006)
7. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF.W3C Recommendation (2008), http://www.w3.org/TR/rdf-sparql-query/
8. Yuanbo, G., Abir, Q., Zhengxiang, P., Jeff, H.: A Requirements Driven Framework for Benchmarking Semantic Web Knowledge Base Systems. IEEE Educational Activities Department, vol. 19, pp. 297–309 (2007)
9. Gray, J.: Database and Transaction Processing Performance Handbook. In: The Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, San Francisco (1993)
10. Clark, K.G., Feigenbaum, L., Torres, E.: SPARQL Protocol for RDF.W3C Recommendation (2008), http://www.w3.org/TR/rdf-sparql-protocol/
11. Dell'Erba, M., Fodor, O., Höpken, W., Werthner, H.: Exploiting Semantic Web Technologies for Harmonizing E-Markets. Journal of Information Technology and Tourism 7, 201–219 (2005)
12. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL Performance Benchmark. In: IEEE 25th International Conference on Data Engineering, ICDE 2009, pp. 222–233 (2009)
13. Foder, O., Werther, H.: Harmonise: A Step Toward an Interoperable E-Tourism Marketplace. International Journal of Electronic Commerce 9, 11–39 (2005)
14. Castro, R.G.: Benchmarking Semantic Web technology. Facultad de Informática, Doctoral Thesis. Universidad Politécnica de Madrid (2008)
15. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web: Scientific American. Scientific American (2001)
16. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web 3, 158–182 (2005)
17. Moshe, Y.V.: The complexity of relational query languages (Extended Abstract). In: Proceedings of the fourteenth annual ACM symposium on Theory of computing. ACM, San Francisco (1982)