Novel Models and Architectures for Distributed Semantic Data Management

Kuldeep B.R. Reddy
Indian Institute of Technology Madras
Chennai, India
brkreddy@cse.iitm.ac.in

***Abstract*—Semantic data management refers to a range of techniques for the manipulation and usage of data based on its meaning and its rapid growth gives rise to the problems of building novel models and architectures for its distributed management allowing efficient query processing and reasoning. The first part of the work proposes an actor model for distributed semantic data management based on the concept of liquid architectures proposing an actor programming framework and execution environment to store, query and reason over structured RDF data. The motivation being to provide a low latency, high throughput distributed platform for semantic data. The second part of the work proposes a pay-as-you-go model and architecture for providing OWL-based semantics as a service including ontology construction, alignment and noise removal from text documents according to the query workload using hadoop map-reduce framework. The third part of the work proposes a query model, including four initial approaches, to generate interactive suggestions as an aid to the user for better formulation of SPARQL queries.**

***Keywords*-distributed semantic data management; actor-based systems; ontology learning; ontology noise removal; ontology alignment; hadoop map-reduce; interactive sparql querying**

I. Introduction

The goal of this paper is to present ideas for research projects on distributed semantic data management and its querying. The paper sketches initial approaches towards designing novel models and architectures for it.

Section II proposes an actor model for distributed semantic data managment based on the concept of liquid architectures. The actor model abstraction, including a programming model and run-time system, has been used to deploy web services on the cloud and this paper proposes to extend it to store semantic data which would allow decentralized query processing and reasoning. The motivation behind it is to develop a low latency, high throughput distributed platform specifically for semantic data. Sections A,B,C,D explain the actor model, proposed architecture, related work and future work respectively.

Section III proposes a pay-as-you-go model to provide semantics as a service which produces the research problems of query specific ontology construction from text, noise removal and alignment over hadoop map-reduce framework. Sections A,B,C,D explain the owl-based semantics as a service, related work, pay-as-you-go framework and architecture and future work.

Section IV introduces the problem of a query model to generate interactive SPARQL query suggestions over distributed semantic data to allow the user to formulate better queries and presents initial three approaches towards it. Sections A,B,C,D explain the initial four approaches including Ontology-based suggestions, Query-log based suggestions, cache based query reformulation suggestions and exploration based suggestion.

II. LiquidRDF : An Actor Model for Distributed Semantic Data Management

This part of the work proposes an Actor Model for Distributed Semantic Data Management. The foundation of the proposed model is based on the Actor Model of Computation. The motivation for the proposed model is to allow the development of low latency and high throughput platform and allow decentralized SPARQL query processing and reasoning. Actor based Distributed Systems are built on the concept that everyone are actors which are able to abstract away the individual hosts and do not share any memory, instead communicating only through messages. The work proposes an architecture of the model based actor programming framework and run-time system where each actor is a liquid RDF store providing a SPARQL endpoint, which for instance can be programmed in rdfstore-js, which is a JavaScript implementation of RDF stores, according to the concept of liquid architectures and a run-time system based on the one proposed in [8]. The following subsections sketch the actor model, programming model and the run-time system to store and query distributed semantic data.

A. Proposed Actor Model

The Actor model is based on the concept that everything is an Actor [3]. An Actor as a computational entity with a behavior such that in response to each message received can concurrently: Send a finite number of messages to other Actors, Create a finite number of new Actors and Designate the behavior to be used for the next message received. Communications with other Actors occur asynchronously. Actors abstract away the individual host. A number of actor languages such as STAGE [4], have been proposed to build actor based distributed systems.

The information workbench provides the front end user interface which allows the client to pose queries and develop applications. The information workbench, proposed in [5], is realized as a Web Application with AJAX-based front end and a pure Java back-end. At its core is a semantic data store which stores RDF data in triple stores that are accessed through Sesame [6] APIs. The FedX model, proposed in [7], implements the semantic store as a federation-based model of SPARQL endpoints. The proposed work makes use of the liquid architectures principles, as proposed in [8] to model each RDF store providing a SPARQL end-point as an actor allowing the development of semantic web applications on the distributed semantic data as well providing a decentralized query processing capabilities.

B. Proposed Architecture

In the proposed programming framework, each actor is a liquid RDF store, which can be programmed in rdfstore-js, that is a JavaScript implementation of RDF stores. A run-time framework is proposed which is based on the actor programming language execution environment which takes care of naming system for the actors. as well as enabling communication amongst them. The problem of changes in distributed storage, processes and the traffic load as a result of it is part of future work.

1) Programming Framework: Actors In the proposed model, each actor is a liquid RDF store providing a SPARQL end-point. The proposed architecture treats each actor both as a semantic service provider and consumer. Each actor has a service description describing the contents of the liquid RDF store written in the form of SPARQL graph patterns. Actors communicate with each other in the RDF data format. The work also plans to add a feature to actors which would allow them to take control of a set of actors and coordinate them to attain specific goals.

Actor Script To begin with, the work plans to use rdfstore-js [9], which is a JavaScript implementation of RDF store with support for SPARQL queries, to program the liquid RDF stores in the proposed architecture.

2) Run-Time Framework: Theater A theater represents the execution environment of the actors. Each system has a theater running on it, which has multiple threads or processes of frames. The runtime should be able to compile each actor as an independent entity but executed on a separate frame on a separate thread or process.

Manager The manager is the module which runs on all the systems and locates the actors and allows communication of messages between them.

Migration & Load Balancing This module will take care of actor allocations on different frames and their migration across the systems.

C. Related Work

An architecture of a worldwide computing framework was proposed in [10], which consists of a actor programming language, a distributed run-time system and a middleware architecture for load balancing. The proposed model in this work can be considered as an adaptation of this framework for the semantic web that would allow leveraging the over-arching standards of the web and the semantic web via RDF, SPARQL, and JavaScript to devise a viable platform for application development. There has been related work on storing RDF data on existing distributed platforms as in RDF on hadoop [23], p2p systems [21] and agent based systems [22]. This work on the other hand proposes a distributed systems designed specifically for semantic data and comparing it with schemes to store on existing distributed platforms is part of future work.

D. Future Work

1. Devising SPARQL query optimization strategies on the proposed architecture for decentralized SPARQL query processing in the proposed model where each actor evaluates a part of the SPARQL query and transmits the partially bound SPARQL query to other actors. Possible approaches towards optimization can be based on the reputation-based message routing model.
2. Devising RDFS reasoning strategies on the proposed architecture to enable inferring new information and presenting additional results for the SPARQL queries including optimizations for forward chaining and backward chaining approaches.
3. Application Development on the proposed architecture.

III. Pay-as-you-Go framework and architecture for OWL-based semantics as a service

This part of the work proposes a Pay-As-You-Go framework and architecture for providing OWL-based Semantics as a Service in the cloud. The model is related to the Data as a Service model which is based on the concept that the data is treated as a product and can be provided as a service, whereas in this model the meaning of the data based on OWL ontologies is treated as a product and provided as a service. Schemes to interpret words have traditionally been based on ontologies and the proposed work addresses the problem of OWL ontology management on the cloud in a pay-as-you-go fashion and offering the interpretation of words based on them as a service. The documents are stored in the cloud and the ontologies are built from them in a pay-as-you-go manner. The ontologies are gradually refined and aligned as needed by the query workload. The following subsections sketch the proposed service, pay-as-you-go model and architecture and the associated associated research problems.

A. OWL-based Semantics as a Service

Cloud computing has emerged as a paradigm which delivers hosted services over the internet [11]. Cloud computing relies on sharing of resources to achieve economies of scale. These services have traditionally been classified into three categories : Infrastructure as a Service, Platform as a Service, Software as a Service.

Here, the cloud provider provides OWL-based Semantics as a Service to the client. Semantics refers to the study of interpretation of words and the idea behind the service is that the meaning of words can be provided as a service over the cloud. Schemes to interpret words have traditionally been based on ontologies. An ontology formally represents knowledge as a set of concepts within a domain, and the relationships between pairs of concepts. The work presents the problem of OWL ontology management on the cloud including the building and maintaining of OWL ontologies in a pay-as-you-go fashion and offering the interpretation of words based on them as a service. The proposed service also looks to provide the user with an option to construct an ontology using the ontologies already present in the web using map-reduce.

Related Work Ontology as a Service was proposed in [12], which is based on sub-ontology extraction and merging, whereby multiple sub-ontologies are extracted from various source ontologies, and then these extracted sub-ontologies are merged to form a complete ontology to be used by the user. The proposed work on the other hand proposes a pay-as-you-go framework where the ontologies are constructed and merged from the text documents as per the queries of the user. In addition, the proposed work looks to remove noise in the ontologies and provides a SPARQL query engine in the cloud and therefore represents a more comprehensive solution to the problem.

B. Pay-as-you-go framework

Pay-As-You-Go framework has traditionally been used for data management [13]. The idea behind the approach has been to provide some services immediately and gradually form tighter integrations as needed. The factors that lead to the concept have been very large volumes of the information that would require significant cost in order to integrate them upfront. This framework combines the process of integration with the query processing and iteratively forms the connections and refines them as per the query workload.

The proposed work utilizes this concepual framework for managing OWL ontologies in the cloud. The documents are stored in the cloud and the ontologies are built from them in a pay-as-you-go manner. The ontologies are gradually refined and aligned as needed by the query workload. The benefit of this approach is that the entire ontology need not be built upfront thus saving costs and the query processing time is also reduced as only the parts of the ontology which are frequently accessed are learned as a result the query is processed over a much smaller part of the ontology.

Figure 1 illustrates the pay-as-you-go framework for OWL ontology management in the cloud. The user interface accepts the query as input from the user. The parser module parses the query converting its constituent terms

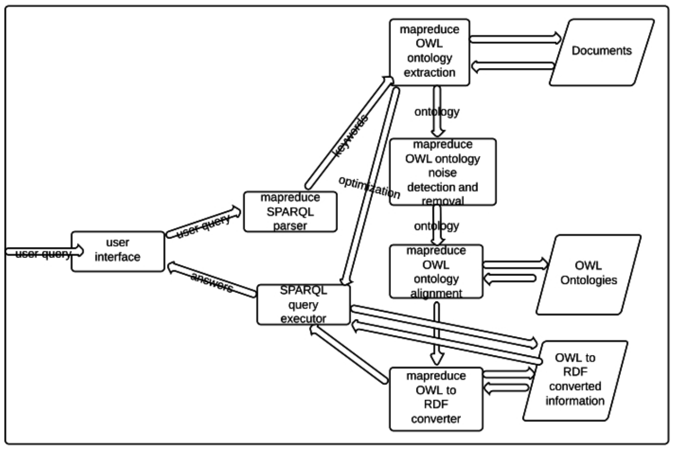
mapreduce OWL ontology extraction
Documents
ontology
keywords
mapreduce OWL ontology noise detection and removal
optimization
mapreduce SPARQL parser
user query
user query
user interface
answers
ontology
mapreduce OWL ontology alignment
OWL Ontologies
SPARQL query executor
OWL to RDF converted information
mapreduce OWL to RDF converter

Figure 1. Pay-As-You-Go Framework for OWL ontology management in the cloud

into keywords. The ontology construction module takes the keywords as input and uses it to set a context for which the ontology is built from the documents. It also refers the existing ontologies to avoid constructing the part which has already been built earlier. The built ontology is aligned and merged with the existing ontologies using the alignment module, the noise removal module is invoked here to detect and remove the inconsistencies and unsatisfiable concepts. The RDF converter module converts the extracted ontology to the RDF and merges it with existing RDF graph stores it across the nodes. The RDF graphs are repartitioned according to the query workload if necessary. The SPARQL query executor module executes the SPARQL over the converted RDF graph and returns the answers back to the user.

C. Pay-as-you-go architecture

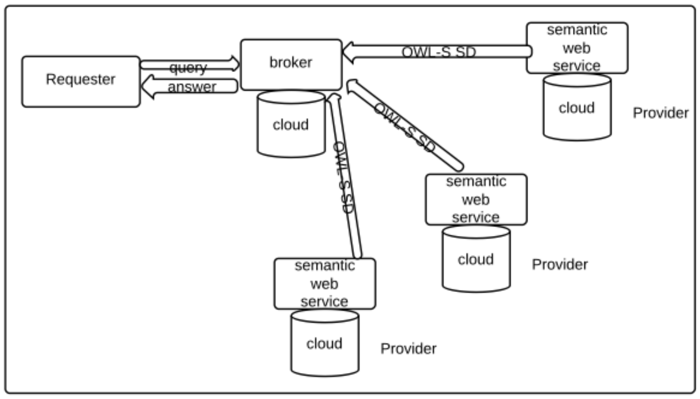

Figure 2. Pay-As-You-Go Architecture of the proposed system

Figure 2 illustrates the pay-as-you-go architecture of the proposed system. Each node illustrates a semantic web service which is implemented using a pay-as-you-go framework in the cloud as described earlier. Each web service releases a service description using OWL-S [14] specification. The OWL-S ontology has three main parts: the service profile, the process model and the grounding. The service profile is used to describe what the service does. The process model describes how a client can interact with the service. The service grounding specifies the details that a client needs to interact with the service. A Broker has emerged as an important component in web services infrastructure by facilitating the discovery and mediation amongst the web services for the client. The proposed architecture implements the broker in the cloud incorporating the idea that the OWL-S service descriptions of the web services are refined over time in a pay-as-you-go manner as per the query workload.

D. Future Work

1. Query-based learning of OWL ontology from documents in the cloud.
The proposed work aims to build a module to learn OWL ontology from the text documents for the given query in the cloud. The idea is to build the OWL ontology in stages as the queries fed as input to the system using the map-reduce framework. The benefit of this approach is that only the relevant parts of the OWL ontology are constructed thus saving the query processing costs. Similar approach to contruct relevant parts of the ontology to the given query was proposed in [15], which involves the user and case-based reasoning. The goal of this module is to devise an approach which constructs the relevant ontology automatically by identifying the topic from the keywords in the queries without involving the user using the map-reduce framework in hadoop and also recognizes that a part of the OWL ontology required for the new query has already been constructed earlier and therefore builds only the required parts.

2. Query-based noise removal in OWL ontology in the cloud.
The proposed work aims to detects and removes noise while the ontology using the information present in the query in the cloud. Noise considered in the work are the inconsistencies and the unsatisfiable concepts present in the ontologies. The existing approaches to detect and remove them have been presented in [16]. The goal of this part of the work is to devise a scheme which treats the additional structural information present in the query as valuable and uses it during the noise removal process to resolve inconsistencies and detect erroneous concepts in the OWL ontology using the map-reduce framework in hadoop.

3. Query-based OWL ontology alignment in the cloud.
The proposed work aims to map and align the ontologies in the cloud taking the help of the additional structural information present in the query. The current approach presented in [17] is based on particle swarm optimization. An ontology alignment is defined as a set of correspondences between ontological entities, i.e. classes, properties, and individuals, of two ontologies. The goal of this module is to devise an approach which looks to utilize the additional structural knowledge present in the query during the alignment process using the map-reduce framework in hadoop.

4. SPARQL query engine for OWL ontology in the cloud.
The proposed work aims to build a SPARQL query engine for querying OWL ontology using the map-reduce framework in hadoop. One of the approaches to query OWL ontology using SPARQL on a stand-alone machine works by computing closure of the entire OWL ontology then converting it to RDF graph and then processing SPARQL queries on the resulting RDF graph. The goal of this module is to devise an approach to identify only a portion of the OWL ontology related to the query, compute its closure and convert it to the RDF graph on the cloud and merge it with the existing RDF graphs and process the query on it. The proposed approach will reduce the execution time considerably as the query is executed on a smaller OWL ontology. The RDF graphs can be partitioned using the METIS graph partitioning algorithm and stored accross different nodes and they can be modified as per the query workload.

5. Pay-as-you-go cloud-based OWL-S broker for semantic web services.
The proposed work aims to build a cloud-based OWL-S semantic web services broker. The model here is that the service descriptions of the web services in the OWL-S format are continously being refined and updated as per the query workload. An architecture of the broker was presented in [18], the proposed work aims to extend the broker implementation to the cloud while making changes to the broker protocol of advertisement and mediation which would allow taking into account the pay-as-you-go refinements of OWL-S service descriptions.

IV. Query Model for Interactive SPARQL query suggestions over distributed semantic data

This part of the paper presents an overview of the research project dealing with generating suggestions to the SPARQL queries given by the user which are executed over distributed semantic data. We present four approaches for it. In the first approach, we recognize that a triple pattern is erroneous in the query by comparing it with the RDFS/OWL ontology. The suggestions to correct the errors in the triple patterns are presented to the user. In our second approach, we make use of the SPARQL query log to produce suggestions. In our third approach, we propose to suggest modifications to the SPARQL query being executed to the user in order to speed up its execution. In the fourth approach, we explore the distrubuted semantic data surrounding the entities discovered during the query execution process looking for paths which are equivalent to the a predicate in the query and present them as suggestions to the user.

A. *Ontology-based suggestions*

In the first approach, we make use of RDFS/OWL ontology to generate suggestion to correct possible errors in the query triple patterns. The approach works by taking each predicate at level i in the SPARQL BGP and comparing its domain and range with the domains and ranges of its preceding and subsequent predicate at levels i-1 and i+1. If their intersection results is empty set, we recognize this predicate as the one which requires correction. We then look to replace the predicate with the predicates from the ontology being used by the user whose domain and range matches that of the preceding and subsequent predicates in the query. With this approach, we get a number of queries with different predicates chosen from the ontology all of which are presented to the user as suggestions. Future work also consists of how the quality of service can be maintained.

B. *Query-log based Suggestions*

This section addresses the problem of generation SPARQL query suggestions as they are being partially entered by the user using the query log. We represent the set of queries entered by the user in the form of a single directed graph. The nodes of this graph contain the triple patterns and the edges contain weights which represent the number of times the triple pattern has been given as part of a query. We normalize the different variable names given in queries into a single set, with each variable representing the nodes at each stage in the query log graph. We maintain a list containing the different combinations of the two variables in the descending order of their occurrance for each node in the query log graph.

When the new query is being entered by the user, the triple patterns entered are matched with the nodes in the query log graph. To produce suggestions, the outgoing edges for the matched node in the graph are collected and sorted in the decreasing order of their weights and the triple patterns which are attached to the edges presented to the user as the next possible triple pattern and the two variables for the triple pattern are picked from the node's associated variable list. We also propose the use of an ontology during the process of generation of suggestions by re-ordering the triple patterns based on how close they are semantically to the user's original query. The semantic distance is computed using the least common ancestor method between two concepts in the ontology.

C. *Cache based Query Reformulation suggestions at Run-time*

This section presents an approach to generate suggestions in order to speed up the execution of SPARQL queries. Our approach proceeds by suggesting the modifications to the query at run-time to the user. The query is modified by either replacing the predicates in it with another set of predicates chosen from a query which was issued earlier or adding new set of predicates from an earlier query whose results are present in the cache. This allows the remaining results for a part of the query to be picked from the cache itself.

In order to replace or add the new predicates in the current query, we take a sample of the intermediate results of the query and compare it with sample of the intermediate results of previous queries during the query execution [19]. If the number of matches exceeds a pre-determined threshold we generate the new query and suggest the modification to the user to either replace it with the new predicates or add the new ones instead, such that the intermediate results are now picked from the cache. The scheme poses the question of how to access the relevant samples of intermediate results of previous query executions efficiently which we plan to address in the future.

D. *Query reformulations suggestion based on exploration*

In the fourth approach, we explore the distributed semantic data looking for paths which are equivalent to the concerned predicate in the query and present them as suggestions to the user. We consider an equivalent path as one which connects the same pair of entities which the concerned predicate in the query connects. Of course, there will be many sets of equivalent paths which requires heuristics to determine the set which can be used to expand the query. The heuristic we use in this paper computes the number of times the query predicate and an probable equivalent path occur together. If the number of times a probable equivalent path occuring with the query predicate exceeds a certain threshold, we confirm it as an equivalent path and use it to expand the query. We also propose the use of an ontology to optimize the search process of semantically equivalent paths. The semantic distance is computed using the least common ancestor method between two concepts in the ontology. Future work also consists of making use of statistics and ontology matching techniques to precisely determine the equivalent path.

E. *Conclusions*

In this part of the paper, we presented an overview of the research project dealing with generating suggestions to SPARQL queries given by user which are executed over the web of linked data. We also proposed four approaches for this problem. Related work is being done in the field of traditional databases to generate suggestions to SQL queries [20] and we believe extending it to SPARQL taking into consideration the semantic web concepts of reasoning and implicit information represents a new research direction worth exploring.

References

[1] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data – the story so far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009.

[2] O. Hartig and J.-C. Freytag, "Foundations of traversal based query execution over linked data." in *HT*, E. V. Munson and M. Strohmaier, Eds. ACM, 2012, pp. 43–52. [Online]. Available: http://dblp.uni-trier.de/db/conf/ht/ht2012.htmlHartigF12

[3] C. Hewitt, "Actor model of computation: Scalable robust information systems," Tech. Rep. v24, July 2012, cite arxiv:1008.1459Comment: improved syntax. [Online]. Available: http://arxiv.org/abs/1008.1459v24

[4] J. Ayres and S. Eisenbach, "Stage: Python with Actors," in *International Workshop on Multicore Software Engineering (IWMSE)*, April 2009. [Online]. Available: http://pubs.doc.ic.ac.uk/actors-in-python/

[5] P. Haase, M. Schmidt, and A. Schwarte, "The information workbench as a self-service platform for linked data applications." in *COLD*, ser. CEUR Workshop Proceedings, vol. 782. CEUR-WS.org, 2011. [Online]. Available: http://dblp.uni-trier.de/db/conf/semweb/cold2011.htmlHaaseSS11

[6] J. Broekstra, A. Kampman, and F. van Harmelen, "Sesame: A generic architecture for storing and querying RDF and RDF Schema," in *The Semantic Web – ISWC 2002: First International Semantic Web Conference Sardinia, Italy*, 2002, pp. 54–68.

[7] K. Hose, R. Schenkel, M. Theobald, and G. Weikum, "Database foundations for scalable RDF processing," in *Reasoning Web*, 2011, pp. 202–249.

[8] D. Bonetta and C. Pautasso, "Towards liquid service oriented architectures." in *WWW (Companion Volume)*. ACM, 2011, pp. 337–342. [Online]. Available: http://dblp.uni-trier.de/db/conf/www/www2011c.htmlBonettaP11

[9] A. Garrote. (2011) antoniogarrote / rdfstore-js. [Online]. Available: https://github.com/antoniogarrote/rdfstore-js

[10] T. Desell, K. E. Maghraoui, and C. Varela, "Load balancing of autonomous actors over dynamic networks," in *In Proceedings of the Hawaii International Conference on System Sciences, HICSS-37 Software Technology Track*, 2004, pp. 1–10.

[11] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, ser. NCM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–51. [Online]. Available:

[12] A. Flahive, D. Taniar, and W. Rahayu, "Ontology as a service (oaas): a case for sub-ontology merging on the cloud," *The Journal of Supercomputing*, pp. 1–32, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11227-011-0711-4

[13] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspaces: a new abstraction for information management," *SIGMOD Rec.*, vol. 34, no. 4, pp. 27–33, Dec. 2005. [Online]. Available: http://doi.acm.org/10.1145/1107499.1107502

[14] D. Martin, M. Burstein, E. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic Markup for Web Services," Tech. Rep., Nov. 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[15] N. B. Mustapha, H. B. Zghal, M.-A. Aufaure, and H. H. B. Ghzala, "Semantic search using modular ontology learning and case-based reasoning." in *EDBT/ICDT Workshops*, ser. ACM International Conference Proceeding Series. ACM, 201. [Online]. Available: http://dblp.uni-trier.de/db/conf/edbtw/edbtw2010.htmlMustaphaZAG10

[16] P. Haase and J. Vlker, "Ontology learning and reasoning – dealing with uncertainty and inconsistency," in *Uncertainty Reasoning for the Semantic Web I*, ser. LNCS. Springer Berlin / Heidelberg, 2008, vol. 5327, pp. 366–384.

[17] J. Bock, A. Lenk, and C. Dänschel, "Ontology Alignment in the Cloud," in *Proceedings of the 5th International Workshop on Ontology Matching (OM-2010)*, vol. 689. http://ceur-ws.org: CEUR Workshop Proceedings, November 2010, pp. 73–84.

[18] M. Paolucci, J. Soudry, N. Srinivasan, and K. Sycara, "A broker for owl-s web services," in *First International Semantic Web Services Symposium, AAAI Spring Symposium Series*, 2004.

[19] M. Yang and G. Wu, "Caching intermediate result of sparql queries." in *WWW (Companion Volume)*. ACM, 2011, pp. 159–160.

[20] J. Fan, G. Li, and L. Zhou, "Interactive sql query suggestion: Making databases user-friendly," in *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*. IEEE Computer Society, 2011, pp. 351–362.

[21] Filali. Imen, Bongiovanni. Francesco, Huet. Fabrice and Baude. Francoise, "A Survey of Structured P2P Systems for RDF Data Storage and Retrieval." in *Transactions on Large-Scale Data and Knowledge-Centered Systems III*. Springer Berlin / Heidelberg, volume 6790, 2011.

[22] Fuhr. Norbert, Klas. Claus-Peter, "Combining RDF and Agent-Based Architectures for Semantic Interoperability in Digital Libraries" in *Proceedings of the DELOS Workshop on Interoperability in Digital Libraries*.2001.

[23] Husain. Mohammad Farhan, Doshi. Pankil, Khan. Latifur, Thuraisingham. Bhavani, "Storage and Retrieval of Large RDF Graph Using Hadoop and MapReduce" in *Cloud Computing*. . Springer Berlin / Heidelberg, volume 5931, 2009.