International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 6 | ISSN : 2456-3307

Design and implementation Framework of Energy Efficient and Scalable RDF data Query Processing to Multi Server Query Processor (MSQP) in ELearnAPP

P. Hariharan*[1], R. Prakash[2]

[1]Assistant Professor, PG & Research Department of Computer Science and Applications, Adhiparasakthi College of Arts and Science (Autonomous), G.B.Nagar, Kalavai, Vellore, Tamil Nadu, India

[2] M.Phil (CS) Research Scholar, PG & Research Department of Computer Science and Applications, Adhiparasakthi College of Arts and Science (Autonomous), G.B.Nagar, Kalavai, Vellore, Tamil Nadu, India

ABSTRACT

E-learning is learning new things through the use of technologies. It is growing at a rapid pace. Today more organizations are taking up e-learning. While e-learning technology developed extensively since its origin, there are numerous issues that experts find when come to executing e-learning Planning. One of the fundamental issues is the complexity of integrating these systems with content and with different type of business systems. RDF is a data model for representing labeled directed graphs, and it is an important building block of semantic web. Due to its flexibility and relevance, RDF has been utilized as a piece of e-learning. In these applications, large-scale graph datasets are extremely normal. Notwithstanding, existing techniques are not effectively managing them. We introduce a query processing system using Parallel Web Server, it consists of two noteworthy modules (1) The Master node and (2) Worker Nodes. The Master node investigates and analyzes the RDF data and places parts of data over multiple servers. The Worker Nodes parses the user query and distributes sub queries to cluster nodes. Also, the results of sub queries from various servers are gathered (and re-evaluated if necessary) and delivered to the user. Parallel Web Server goes for process queries by their deadlines, and preferred advantage high-level scheduling data to reduce the CPU energy consumption of a query-processing node. MSQP construct its decision on query efficiency predictors, estimating the processing volume and preparing time of a query. In e-learning ecosystem can assist organizations to achieve the advantages of an integrated approach to develop e-learning systems. It can be utilized for building a virtual environment for both educating and learning.

Keywords : Ranking, Review, Rating, Android Market, Search Rank Fraud, Malware Detection

I. INTRODUCTION

As of late, we have witnessed huge development and gigantic changes in the e-learning industry. A effective e-learning course requires thinking about the following contextual components:

- ✓ **Environment** - students require a specific environment (PC, Connection, software) and some preparation should be done to ensure that the student has that.
- ✓ **Teach skills** - students need to know something about how to utilize whatever learning framework exists.
- ✓ **Subject matter skills** - students need to have some essential skills to profit from the course.
- ✓ **Support** - there must be a system to get support when students keep running into issues.

- ✓ **Content** - must be designed for interaction/ collaboration.
- ✓ **Instructor** - mindful of students needs/concerns and involvement levels, endeavors to draw students into discussion early, organizes schedule, provides resources for students in needing of additional learning (remedial).
- ✓ **Technology** - should assume a servant role.
- ✓ **Organisation** - concentrated on learning, time and assets made accessible, students supported through help-desk.

Traditionally, social information processing is scaled out by partitioning the relations and rewriting the query plans to reorder operations and utilize disseminated versions of the operators empowering *intra-operator* parallelism. While some operations are easy to parallelize (e.g., large-scale, distributed counts), many operations, such as distributed joins, are more complex to parallelize in view of the subsequent movement they potentially generate.

While much more recent than social information management, RDF data management has obtained numerous relational techniques; Many RDF systems rely on hash-partitioning (on triple or property tables) and on dispersed choices, projections, and joins. Our own particular GridVine system was one of the first systems to do so in the context of large-scale decentralized RDF management. Hash partitioning has numerous advantages, including simplicity and effective load balancing. In any case, it additionally produces much inter-process traffic, given that related triples (e.g., that must be chosen and afterward joined) wind up being scattered on all machines.

In this article, we propose **MSQP**, an efficient, distributed and scalable RDF data processing system for distributed and cloud conditions. s opposed to numerous distributed systems, **MSQP** utilizes a resolutely non-relational storage format, where semantically related data patterns are mined from both the instance-level and the schema-level data and get co-located to minimize inter-node operations. The principle contributions of this article are:

- A new hybrid storage model that efficiently and effectively partitions an RDF graph and physically co-locates related instance data.
- A new system architecture for dealing with fine-grained RDF partitions in large-scale.
- Novel information placement procedures to co-locate semantically related pieces of data.
- New information loading and query execution procedures exploiting our system's data partitions and indices.

II. RELATED WORK

Query on a specific geographic region by user to constrain to search results in a Geographic web listes. This geographic search technology is called local search and is being executed with noteworthy interest in realsearch engines. Academic research was conducted for extracting geographic knowledge from the web. Combination of content and spatial information preparing is utilized as a part of such geographic web search engines. This research focus on geographic search engines and it is converged with general web query processing. Each page in such search engine additionally has a geographic area of relevance associated with it, called the geographic footprint of the page It extracts geographic information. for example, city names, addresses, or references to point of interests, from the pages and then maps these to positions using external geographic databases. Footprints are represented as polygons and bitmap based structures. These search engines can be divided into crawling, data mining, list construction, and query processing. In this system, it focuses on Germany and crawl the "de" domain; in cases where the coverage area does not correspond well to any set of domains, focused crawling

strategies that may be needed to find the relevant pages. Overall ranking function might be of the form. Fundamental commitments in this paper are as per the following:

- ✓ Discuss and formally examine the query-processing problem in geographic web search engines.
- ✓ Describe a few proficient algorithms for query processing in geographic search engines.
- ✓ Integrate the algorithms into an existing high performance query processor for a scalable search engine, and assess them on a large web crawl and queries derived from a real query trace.

There are search engines with quick query processing and results about are additionally required quicker where as other search engines with large batches of queries are submitted for different web mining and system enhancement tasks that do not require an immediate response and such search engines are called as batch processing search engines. Here, conclusion is that significant cost diminishments are conceivable by utilizing specific mechanisms for executing batch queries in Web search engines.

Three categories of Web query languages can be distinguished, according to the format of the data they can retrieve: XML, RDF and Topic Maps. This article introduces the spectrum of languages falling into these categories and summarizes their salient aspects.

Various formalisms have been proposed for representing Semantic Web meta-information, specifically RDF, Topic Maps, and OWL (formerly known as DAML+OIL). These formalisms typically enable one to describe relationships between data items, such as concept hierarchies and relations between concepts. Semantic Web is an integrated access to the data on the Web that is spoken to in any of the previously mentioned formalisms.

The accompanying three questions are at the heart of building up a query language:

1. What are the core data retrieval capabilities of each query language?
2. to what extent, and what forms of reasoning do they offer, and
3. How are they realized?

It focuses on introducing and comparing languages designed primarily for providing efficient and effective access to data on the Web and Semantic Web. Specifically, it avoids the following types of languages: Programming language tools for XML, Reactive languages, Rule languages, OWL query languages.

III. STORAGE MODEL

Our storage framework in MSQP can be viewed as a hybrid structure broadening a few of the ideas from above. Our framework is built based on three principle structures: RDF AtomicStructure clusters (which can be viewed as hybrid structures getting from both property tables and RDF subgraphs), Model records (storing literals in compact records as in a column-oriented database framework) and an efficient key list listing URIs and literals based on the clusters they belong to. Contrary to the property-table and column-oriented approaches, our system based on templates and AtomicStructure is more flexible, as in every Models can be adjusted progressively, for example following the addition of new information or a shift in the workload, without requiring to alter the other Models or AtomicStructure. Also, we present a unique combination of physical structures to handle RDF data both horizontally (to flexibly co-locate entities or values related to a given instance) as well as vertically (to co-locate series of entities or values attached to similar instances).

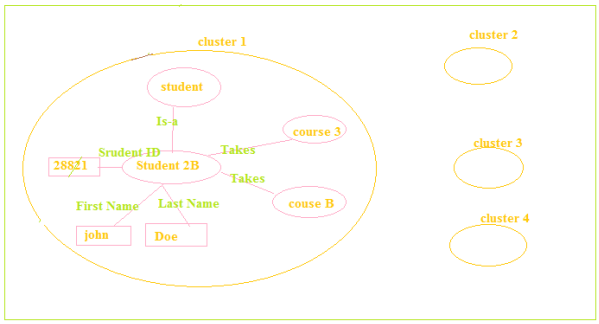

Figure 1. The two main data structures in MSQP: AtomicStructure clusters, storing in this case RDF subgraphs about students, and a Model records, storing a list of literal values corresponding to student IDs.

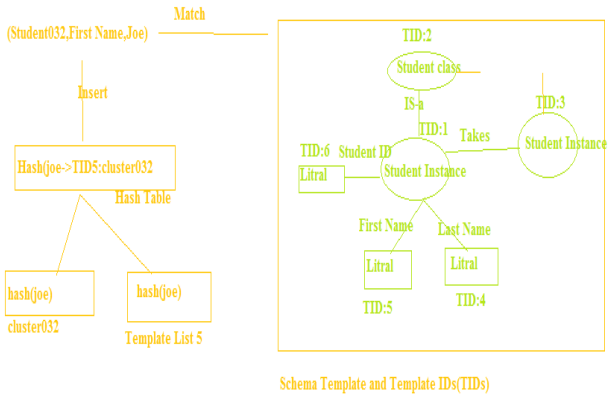

Figure 2. An insert using Models: an incoming triple (left) is matched to the current RDF Model of the database (right), and inserted into the hash-table, a cluster, and a Model list.

Figure 1 gives a simple example of a few AtomicStructure clusters— storing information about students—and of a Model list— compactly storing lists of student IDs. AtomicStructures can be seen as *horizontal* structures storing information about a given instance in the database (like rows in relational systems). Model lists, on the other hand, store *vertical* lists of values corresponding to one attribute (like columns in a relational system).

3.1 Key List

The Key List is the focal list in MSQP; it utilizes a lexicographical tree to parse every incoming URI or literal and assign it a unique numeric key value. It then stores, for every key and every Model ID, an ordered list of all the clusters IDs containing the key (e.g., "key 10011, corresponding to a Course object [Model ID 17], appears in clusters 1011, 1100 and 1101"; see also Figure 2 for another example). This may seem like a peculiar way of listing values, but we show below that this actually allows us to execute numerous queries very efficiently simply by reading or intersecting such Records in the hash-table directly.

3.2 AtomicStructures

MSQP uses physiological RDF partitioning and AtomicStructure patterns to efficiently co-locate RDF data in distributed settings. Figure 3 (ii) gives an example of Atomic Structure. Atomic Structures have three key advantages in our context:

- ✓ Atomic Structures speak to the perfect tradeoff between co-location and degree of parallelism when partitioning RDF data. Partitioning RDF data at the triple-level is problematic due to the numerous joins it generates; Large graph partitions are imperfect also, since all things considered an excessive number of related triples are co-located, in this way inhibiting parallel processing
- ✓ All AtomicStructures are Model-based, and consequently store data extremely compactly;
- ✓ Finally, the Atomic Structures are defined in order to *materialize* frequent joins, for example between an entity and its corresponding values (e.g., between a student and his/her first name), or between two semantically related entities (e.g., between a student and his/her counselor) that are frequently co-accessed.

When accepting another triple the framework embeds it in the corresponding Atomic Structure(s).

In case the corresponding Atomic Structure does not exist yet, the framework creates a new Atomic Structure cluster, embeds the triple in the Atomic Structure, and embeds the cluster in the list of clusters it maintains. Figures 3 give a Model illustration that co-locates information relating to Student instances along with a case of an Atomic Structure for Student123.

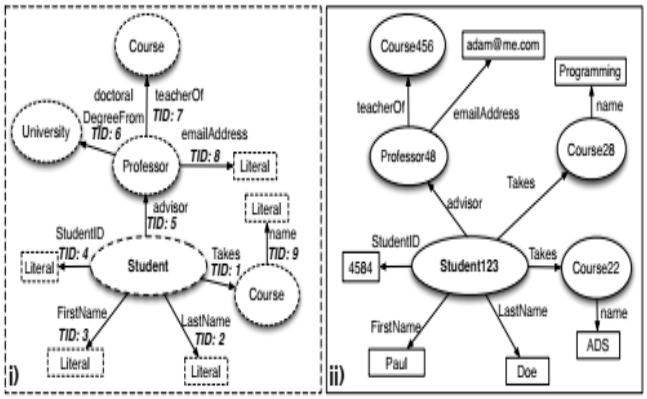

Figure 3. A Atomic Structure Model (i) along with one of its RDF Atomic Structures (ii)

3.3 Auxiliary Lists

While creating Atomic Structure Models and Atomic Structures identifiers, our framework likewise take considerations of two extra data gathering and analysis tasks. First, it inspects both the schema and instance data to determine all subsumption (subclass) relations between the classes, and keeps up this information in a compact *type hierarchy*. We allot to every key the most particular type possible in order to avoid having to materialize the type hierarchy for every instance, and handle type inference at query time by looking up types in the type hierarchy. In case two unrelated types are assigned to a given instance, the partition manager makes a new virtual type composed of the two kinds and assigns it to the instance. Finally, we maintain *statistics* on each Model, counting the number of instances for each vertex (instance / literal) and edge (property) in the Models.

3.4 Master Node

The Master node is made out of three principle subcomponents: a key list in charge of encoding URIs and literals into minimized framework identifiers and of interpreting them back, a partition manager in charge of apportioning the RDF data into recurring subgraphs, and a distributed query executor responsible for parsing the incoming query, rewriting the query plans for the Workers, gathering lastly restoring the outcomes to the client. Note that the Master node can be replicated at whatever point important to insure proper query load balancing and fault-tolerance. The Master can also be duplicated to scale out the key list for extremely large datasets, or to replicate the dataset on the Workers utilizing diverse partitioning schemes (all things considered, each new instance of the Master is responsible for one partitioning scheme).

3.5 Worker Nodes

The Worker nodes hold the partitioned data and its corresponding local indices, and are in charge of running subqueries and sending results back to the Master node. Thoughtfully, the Workers are considerably less complex than the Master node and are built on three main data structures: i) a sort list, clustering all keys based on their types ii) a series of RDF *Atomic Structures*, storing RDF data as very compact subgraphs, and iii) a Atomic Structure list, storing for each key the list of Atomic Structures where the key can be found.

3.6 Query Processing

Query processing in MSQP is very different from past approaches to execute queries on RDF data, because of the three peculiar data structures in our system: a key list associating URIs and literals to Model IDs and cluster lists, clusters storing RDF Atomic Structures in a very compact fashion, and Model lists storing compact lists of literals. All queries made out of one Basic Graph Pattern (star-like queries) are executed absolutely in parallel, autonomously on all Workers

without any central coordination thanks to the Atomic Structures and their Lists. For queries that still require some level of distributed coordination—regularly to deal with distributed joins—we resort to adaptive query execution strategies. We mainly have two different ways of executing distributed joins: at whatever the intermediate result set is small (i.e., up to a few hundred tuples according to our Statistics components), we send all results to the Master, which finalizes the join centrally. Otherwise, we fall back to a distributed hash-join by distributing the smallest result set among the Workers. Distributed joins can be stayed away from as a rule by falling back on the dispersed data partitioning and data co-location schemes described above. Algorithm 1 gives a high-level description of our distributed query execution process featuring where specific tasks are performed in our framework.

IV. ALGORITHM

Algorithm 1 High Level Query Execution Algorithm

1: Master: isolate query based on Atomic Structure scopes to obtain subqueries
2: Master: send sub-queries to workers
3: Workers: execute sub-queries in parallel
4: Master: gather moderate results
5: Master: perform distributed join whenever necessary.

Algorithm 2 Query Execution Algorithm with Join on the Master Node

1. procedure EXECUTEQUERY (a; b)
2. for all BGP in QUERY do ⊳BGP - Basig Graph Pattern
3. if BGP.subject then
4. Atomic Structures ←GetAtomic Structure (subject)
5. else if BGP.object then
6. Atomic Structures← GetAtomic Structures (object)
7. end if
8. for all Atomic Structures do
9. check if the Atomic Structure matches the BGP
10. for all TP in BGP do TP - Triple Pattern
11. if TP.subject != Atomic Structure.subject then
12. nextAtomic Structure
13. end if
14. if TP.predicate! = AtomicStructure.predicate then
15. nextAtomicStructure
16. end if
17. if TP.object! = AtomicStructure.object then
18. nextAtomicStructure
19. end if
20. end for
21. the AtomicStructure matches the BGP, so we can retrieve entities
22. resultBGP←GetEntities (AtomicStructure,BGP)
23. end for
24. results ←resultBGP
25. end for
26. SendToMasterNode (results)
27. end procedure
28. On the Master do Hash Join

V. ELEARNAPP

The ELearnAPP is a domain that consolidates planning, building and evaluation of the learning/educational process and covers the tools for creating, arranging and consolidating content parts. An improved structure of the ELearnAPP as an essential part of the EP is shown in Figure 1, while distinctive sorts of users are interacting with it. The auxiliary units, interacting with each other, have been given the name modules because the environment does not claim to be equipped for playing out every one of the capacities that can be relegated to a very completely finished e-Learning framework. In such a framework, similar units perform essentially more complex functions and are called specialists.

The Learner profile module is in charge of making and storing the profiles of the students (looked over various choices or determined after some tests). The module advances Students details to the repository called User profiles. It is used to store the users' identification details: user name, password, level, objectives, interests, etc. The Data delivery module serves as a mediator between the different EP modules when there is an information search query.

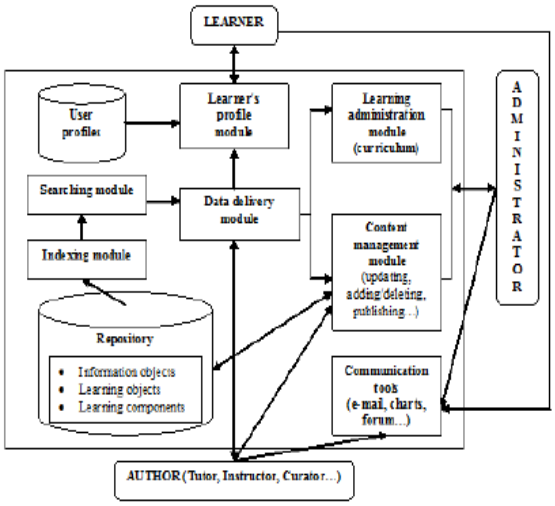

Figure 4. ELearnAPP

The Searching module forms the queries. It does not have direct access to the Repository and uses the Listing module for the assets stored there. This module is in charge of setting a extraordinary number for every resource and in this manner encourages their aggregation in the repository. The Learning organization and Content management modules support tasks identified with information control: adding/deleting, updating and publishing. The main module supports DL process structure (the curriculum) and the second one – content of the subjects in the Repository. The structure of the materials in the Repository is hierarchically organized: discipline, textbook (learning articles), module, section, lesson (learning objects) and term (information objects). The structure of the items and the articles themselves with a few attributes should be stored in separate files, in order to accelerate the processing, to achieve a greater independence and security of the data. For the students the content of every exercise ought to be spoken to by methods for a standard internet browser. At the point when a client picks the web address of the EP, the questions of the program he/she utilizes are gotten and handled by the server. It advances them to the LCMS, which figures out what could possibly be done that.

VI. RESULT

6.1 SCREENSHOT

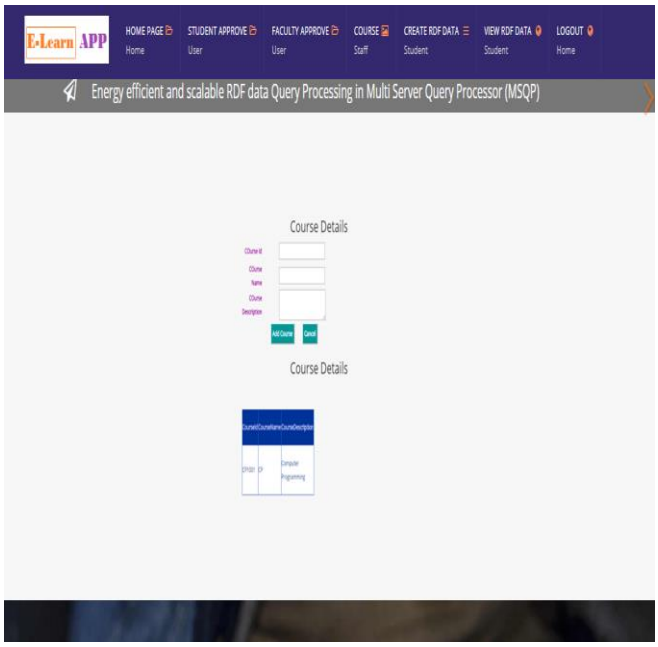

Figure 5. Add Course Details

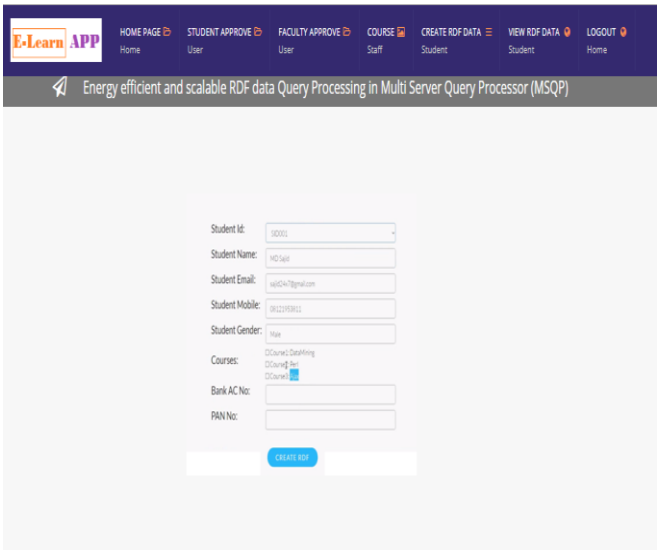

Figure 6. Student Registrations

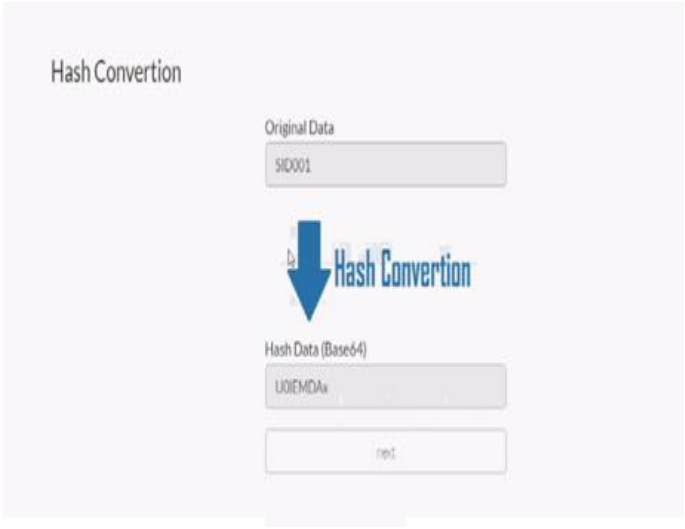

Figure 7. Hash Conversions

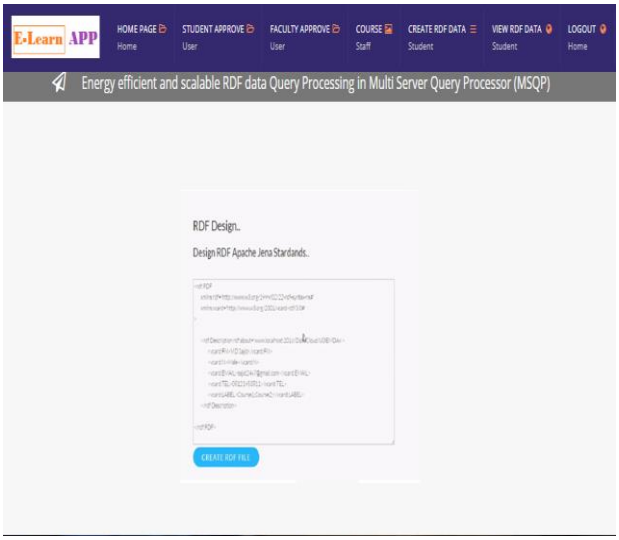

Figure 8: Create RDF Data

VII. CONCLUSIONS

MSQP is an efficient and adaptable framework for for managing RDF data in the cloud. From our viewpoint it strikes an optimal balance between intra-operator parallelism and data co-location by considering repeating, fine-grained physiological RDF partitions, distributed data allocation schemes, leading however to potentially bigger data (excess presented by higher extensions or versatile AtomicStructures), to more complex inserts, and updates. MSQP is especially suited to clusters of commodity machines and cloud environments where network latencies can be high, since it deliberately attempts to avoid all complex and distributed operations for query execution Our exploratory assessment demonstrated that it positively thinks about to best in class frameworks in such conditions. We intend to keep creating MSQP in a few bearings: First, we intend to incorporate some further pressure systems. We intend to chip away at a programmed Models revelation in light of incessant examples and untyped components.

VIII. FUTURE ENHANCEMENT

In Future, we plan take a shot at integrating an inference engine into MSQP to help a larger set of semantic constraints and queries natively. At last, we are as of now testing and extending our framework with with a few accomplices keeping in mind the end goal to oversee to a great degree vast scale, disseminated RDF datasets about bioinformatics applications

IX. REFERENCES

[1]. J. Huang, D. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. In VLDB, pages 1123–1134. ACM, 2011.

[2]. D. Kossmann. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.

[3]. G. Ladwig and T. Tran. Linked data query processing strategies. In ISWC, pages 453–469. Springer, 2010.

[4]. A. Langegger, W. W¨oß, and M. Bl¨ochl. A semantic web middleware for virtual data integration on the web. In ESWC, pages 493–507. Springer, 2008.

[5]. S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. STR: A simple and efficient algorithm for R-Tree packing. In ICDE, pages 497–506. IEEE Computer Society, 1997.

[6]. Y. Li and J. Heflin. Using reformulation trees to optimize queries over distributed

heterogeneous sources. In ISWC, pages 502–517. Springer, 2010.

[7]. T. Neumann and G. Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In ICDE, pages 984–994. IEEE Computer Society, 2011.

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

[9]. B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In ESWC, pages 524–538. Springer, 2008.

[10]. K. Stocker, D. Kossmann, R. Braumandl, and A. Kemper. Integrating semi-join-reducers into state of the art query processors. In ICDE, pages 575–584. IEEE Computer Society, 2001.

[11]. China IDC, 2012. Data center power will double in the next five years. Online]. Available:⟨http://tech.idcquan.com/pro/34910.shtml⟩ .(accessed 2016).

[12]. Do, J., Kee, Y.S., Patel, J.M., et al., 2013. Query processing on smart SSDs: opportunities and challenges. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM: New York. pp. 1221–1230.

[13]. Dokeroglu, T., Bayir, M.A., Cosar, A., 2015. Robust heuristic algorithms for exploiting the common tasks of relational cloud database queries. Appl. Soft Comput. 30 (C), 72–82.

[14]. Global action plan, 2007. An inefficient truth. Global action plan report. Online]. Available: http://www.global action plan.org.uk/⟩.(accessed 2016).

[15]. Graefe, G., 2008. Database servers tailored to improve energy efficiency. In: Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management. New York: ACM. pp. 24–28.

[16]. D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in Proc. ISCA, 2011, pp. 319–330.

[17]. C. D. Manning, P. Raghavan, and H. Sch¨utze, Introduction to Information Retrieval. Cambridge University Press, 2008.

[18]. M. Catena, C. Macdonald, and I. Ounis, "On inverted list compression for search engine efficiency," in Proc. ECIR, 2014, pp. 359–371.

[19]. J. Dean, "Challenges in building large-scale information retrieval systems: Invited talk," in Proc. WSDM, 2009.

[20]. S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," Found. Trends Inf. Retr., vol. 3, no. 4, pp. 333–389, Apr. 2009.