IJSEAT International Journal of Science Engineering and Advance Technology

An Efficient and Scalable Cloud Approach for Managing the Data in the Cloud

Kancherla Rajasree[1], Yarlagadda Siva KoteswaraRao[2]
[1]M.Tech, Dept of CSE, Eluru College of Engineering and Technology, Eluru.
[2]Assistant Professor, Dept of CSE, Eluru College of Engineering and Technology, Eluru.

Abstract: In spite of late advances in dispersed Resource Description Frame work (RDF) information administration, preparing a lot of RDF information in the cloud is still extremely difficult. Disregarding its apparently straightforward information display, RDF really encodes rich and complex charts blending both case and construction level information. Sharding such information utilizing established systems or dividing the diagram utilizing conventional min-slice calculations prompts extremely wasteful dispersed operations and to a high number of joins. In this paper, we depict DiploCloud, a productive and adaptable conveyed RDF information administration framework for the cloud. In opposition to past methodologies, DiploCloud runs a physiological investigation of both occurrence and blueprint data preceding apportioning the information. In this paper, we depict the design of DiploCloud, its principle information structures, and additionally the new calculations we use to segment and disseminate information. We likewise exhibit a broad assessment of DiploCloud demonstrating that our framework is frequently two requests of greatness speedier than cutting edge frameworks on standard workloads.

Keywords: RDF, triple stores, cloud computing, Big data.

1. INTRODUCTION

The appearance of distributed computing empowers to effortlessly and efficiently arrangement registering assets, for instance to test another application or to scale a present programming establishment flexibly. The many-sided quality of scaling out an application in the cloud (i.e., adding new registering hubs to suit the development of some procedure) particularly relies on upon the procedure to be scaled. Regularly, the job needing to be done can be effectively part into an extensive arrangement of subtasks to be run freely and simultaneously. Such operations are usually called embarrassingly parallel. Embarrassingly parallel issues can be generally effectively scaled out in the cloud by propelling new procedures on new item machines. There are however many procedures that are a great deal more hard to parallelize, normally in light of the fact that they comprise of successive procedures (e.g., forms in view of numerical strategies, for example, Newton's technique). Such procedures are called characteristically successive as their running time can't be accelerated essentially paying little respect to the quantity of processors or machines utilized. A few issues, at long last, are not inalienably successive essentially but rather are hard to parallelize practically speaking as a result of the bounty of between process movement they create.

Scaling out organized information preparing frequently falls in the third class. Generally, social information preparing is scaled out by apportioning the relations and changing the inquiry arrangements to reorder operations and utilize dispersed forms of the administrators empowering intra-administrator parallelism. While a few operations are anything but difficult to parallelize (e.g., largescale, conveyed tallies), numerous operations, for example, circulated joins, are more intricate to parallelize due to the subsequent movement they conceivably produce.

While a great deal later than social information administration, RDF information administration has acquired numerous social strategies; Many RDF frameworks depend on hash-apportioning (on triple or property tables, see underneath Section 2) and on appropriated choices, projections, and joins. Our own Grid-Vine framework [1], [2] was one of the main frameworks to do as such with regards to expansive scale decentralized RDF administration. Hash apportioning has many preferences, including straightforwardness and successful load-adjusting. Notwithstanding, it likewise creates much between process movement, given that related triples (e.g., that must be chosen and after that joined) wind up being scattered on all machines.

In this article, we propose DiploCloud, an effective, appropriated and adaptable RDF information preparing framework for dispersed and cloud conditions. In spite of many conveyed frameworks, DiploCloud utilizes an undauntedly non-social stockpiling position, where semantically related information examples are mined both from the case level and the construction level information and get co-situated to limit internode operations. The principle commitments of this article are:

another half and half stockpiling model that proficiently and viably allotments a RDF diagram and physically co-finds related occurrence information (Section 3);
another framework design for taking care of fine-grained RDF parcels in huge scale (Section 4);
novel information situation procedures to co-find semantically related bits of information (Section 5); new information stacking and inquiry execution methodologies exploiting our framework's information allotments and files (Section 6);
a broad trial assessment demonstrating that our framework is regularly two requests of greatness quicker than cutting edge frameworks on standard workloads (Section 7).
DiploCloud expands on our past approach diplodocus ½RDF [3], an effective single hub triplestore. The framework was additionally stretched out in TripleProv [4], [5] to bolster putting away, following, and questioning provenance in RDF inquiry handling.

2. RELATEDWORK

Many methodologies have been proposed to upgrade RDF stockpiling and SPARQL question handling; we list beneath a couple of the most well known methodologies and frameworks. We allude the peruser to late reviews of the field, (for example, [6], [7], [8], [9] or, all the more as of late, [10]) for a more thorough scope. Approaches for putting away RDF information can be comprehensively ordered in three subcategories: triple-table methodologies, property-table methodologies, and chart based methodologies. Since RDF information can be viewed as sets of subject-predicate-question triples, numerous early methodologies utilized a goliath triple table to store all information. Hexastore [11] recommends to list RDF information utilizing six conceivable lists, one for every stage of the arrangement of segments in the triple table. RDF-3X [12] and YARS [13] take after a comparable approach. BitMat [14] keeps up a three-dimensional piece solid shape where every cell speaks to a remarkable triple and the cell esteem indicates nearness or nonattendance of the triple. Different strategies propose to accelerate RDF inquiry preparing by considering structures grouping RDF information in view of their properties. Wilkinson et al. propose the utilization of two sorts of property tables: one containing groups of qualities for properties that are regularly co-got to together, and one abusing the sort property of subjects to bunch comparative arrangements of subjects together in a similar table. Owens et al. propose to store information in three B+-tree lists. They utilize SPO, POS, and OSP changes, where each record contains all components of all triples. They separate an inquiry to fundamental chart designs which are then coordinated to the put away RDF information. Various further methodologies propose to store RDF information by exploiting its diagram structure. Yan et al. propose to separate the RDF diagram into subgraphs and to assemble auxiliary files (e.g., Bloom channels) to rapidly distinguish whether some data can be found inside a RDF subgraph or not. Ding et al. recommend to part RDF information into subgraphs (atoms) to all the more effectively track provenance information by reviewing clear hubs and exploiting a foundation metaphysics and utilitarian properties. Das et al. in their framework called gStore arrange information in nearness list tables. Every vertex is spoken to as a passage in the table with a rundown of its active edges and neighbors. To file vertices, they manufacture a S-tree in their contiguousness list table to diminish the pursuit space. Brocheler et al. propose an adjusted double tree where every hub containing a sub diagram is situated on one circle page.
Dispersed RDF question handling is a dynamic field of research. Past SPARQL leagues approaches (which are outside of the extent of this paper), we refer to a couple of well known methodologies beneath. Like an expanding number of late frameworks, The Hadoop Distributed RDF Store (HDRS)1 utilizes MapReduce to prepare dispersed RDF information. RAPID+ augments Apache Pig and empowers more effective SPARQL inquiry handling on MapReduce utilizing an option question algebra. Their stockpiling model is a settled hash-delineate. Information is gathered around a subject which is a first level key in the guide i.e. the information is co-situated for a common subject which is a hash an incentive in the guide. The settled component is a hash delineate predicate as a key and protest as an esteem. Sempala expands on top of Impala stores information in a wide bound together property tables keeping one star-like shape per push. The writers split SPARQL questions to straightforward Basic Graph Patterns and modify them to SQL, tailing they figure a characteristic join if necessary. Jena HBase2 utilizes the HBase prevalent wide-table framework to actualize both triple-table and property-table circulated stockpiling. Its information model is a segment oriented, sparse, multi-dimensional sorted guide. Sections are gathered into segment families and timestamps add an extra measurement to every cell. Cumulus RDF3 utilizes Cassandra and hash-dividing to appropriate the RDF tiples. It stores information as four records [13] (SPO, PSO, OSP, CSPO) to bolster a total file on triples and queries on named charts (settings). We as of late took a shot at an observational assessment to decide the degree to which such noSQL frameworks can be utilized to oversee RDF information in the cloud4 [25].
Our past GridVine [1], [2] framework utilizes a triple-table stockpiling methodology and hash-apportioning to disseminate RDF information over decentralized P2P systems. YARS2,5 Virtuoso6, 4store, and SHARD hash parcel triples over numerous machines and parallelize the inquiry processing. Virtuoso by Erlin et al. stores

information as RDF quads comprising of the accompanying components: chart, subject, predicate, and protest. Every one of the quads are continued in one table and the information is divided in light of the subject. Virtuoso executes two records. The default file (set as an essential key) is GSPO (Graph, Subject, Predicate, Object) and a helper bitmap file (OPGS). A comparable approach is proposed by Harris et al., where they apply a basic stockpiling model putting away quads of (model, subject, predicate, question). Information is parceled as nonoverlapping sets of records among portions of equivalent subjects; fragments are then disseminated among hubs with a round-robin calculation. They keep up a hash table of charts where every passage focuses to a rundown of triples in the diagram. Furthermore, for each predicate, two radix tries are utilized where the key is either subject or protest, and individually question or subject and diagram are put away as sections (they consequently can be viewed as customary P:OS and P:SO records). Literals are ordered in a different hash table and they are spoken to as (S,P, O/Literal). SHARD keeps information on HDFS as star-like shape revolving around a subject and all edges from this hub. It presents a condition emphasis calculation the primary thought of which is to emphasize over all provisions and incrementally tie factors and fulfill compels.

3 STORAGE MODEL

Our capacity framework in DiploCloud can be viewed as a half breed structure broadening a few of the thoughts from above. Our framework is based on three primary structures: RDF particle groups (which can be viewed as cross breed structures acquiring both from property tables and RDF subgraphs), layout records (putting away literals in conservative records as in a segment situated database framework) and a proficient key list ordering URIs and literals in light of the bunches they have a place with. As opposed to the property-table and segment situated methodologies, our framework in view of layouts and particles is more versatile, as in every format can be changed progressively, for instance taking after the addition of new information or a move in the workload, without requiring to adjust alternate layouts or atoms. What's more, we present a one of a kind mix of physical structures to deal with RDF information both on a level plane (to adaptably co-find elements or qualities identified with a given case) and additionally vertically (to co-find arrangement of elements or qualities joined to comparable examples).

Atom bunches are utilized as a part of two courses in our framework: to legitimately amass sets of related URIs and literals in the hashtable (in this manner, pre-registering joins), and to physically co-find data identifying with a given question on circle and in mainmemory to decrease plate and CPU reserve latencies. Format records are essentially utilized for investigation and total questions, as they permit to prepare extensive arrangements of literals proficiently.

3.1 Key Index

The Key Index is the focal record in DiploCloud; it utilizes a lexicographical tree to parse every approaching URI or strict and appoint it a remarkable numeric key esteem. It then stores, for each key and each format ID, a requested rundown of the considerable number of groups IDs containing the key (e.g., "key 10011, comparing to a Course protest [template ID 17], shows up in bunches 1011, 1100 and 1101". This may seem like a quite impossible to miss method for ordering values, yet we appear underneath this really enables us to execute many inquiries productively just by perusing or converging such records in the hashtable specifically.

The key file is in charge of encoding all URIs and literals showing up in the triples into a one of a kind framework id (key),and back. We utilize a custom-made lexicographic tree to parse URIsand literals and dole out them a one of a kind numeric ID. The lexicographic tree we utilize is essentially a prefix tree part the URIs or literals in light of their regular prefixes (since numerous URIs have the same prefixes) with the end goal that every substring prefix is put away once and just once in the tree. A key ID is put away at each leaf, which is made out of a sort prefix (encoding the kind of the component, e.g., Student or xsd : date) and of an auto-increased case identifier. This prefix trees enable us to totally stay away from potential crashes (created for example when applying hash works on extensive datasets),and additionally let us minimally co-find both sort and case ids into one minimized key. A moment structure makes an interpretation of the keys once again into their unique frame. It is made out of an arrangement of reversed lists (one for each sort), each relating an example ID to its comparing URI/exacting in the lexicographic tree keeping in mind the end goal to empower proficient key look-ups.

3.2 Templates

One of the key developments of DiploCloud rotates around the utilization of decisive stockpiling designs [36] to productively assemble huge accumulations of related values on plate and in principle memory. At the point when settingup another database, the database head may give DiploCloud a couple implies in the matter of how to store the information on circle: the manager can give a rundown of triple examples to indicate the root hubs, both for the format records and the atom bunches (see for example Fig. 1, where "Understudy" is the root hub of the particle, and "StudentID" is the root hub for the layout list). Group roots are utilized to figure out which bunches to make: another group is made for each occurrence of a root hub in the database. The bunches contain all triples withdrawing from the root hub while navigating the chart, until another example of a root hub is crossed (in this way, one can join groups in view of their root hubs). Format roots are utilized to figure out which literals to store in layout records.

In light of the capacity designs, the framework handles two fundamental operations in our framework: i) it keeps up a blueprint of triple formats in principle memory and ii) it oversees layout records. At whatever point another triples enters the framework, it partners layout IDs comparing to the triple by considering the sort of the subject, the predicate, and the kind of the protest. Each unmistakable rundown of "(subject-sort, predicate, question sort)" characterizes another triple format. The triple formats assume the part of a case based RDF mapping in our framework. We don't depend on the express RDF pattern to characterize the formats, since an extensive extents of requirements (e.g., areas, reaches) are regularly precluded in the blueprint (as it is for instance the case for the information we consider in our trials, see Section 7). On the off chance that another format is distinguished (e.g., another predicate is utilized), then the layout chief updates its in-memory triple format blueprint and embeds new format IDs to mirror the new example it found. In the event of exceptionally inhomogeneous informational indexes containing a large number of various triple layouts, special cases can be utilized to regroup comparable formats (e.g., "Understudy - likes - *"). Take note of this is exceptionally uncommon practically speaking, since all the datasets we experienced up until this point (even those in the LOD cloud) regularly consider a couple of thousands triple layouts at most.

A while later, the framework embeds the triple in one or a few atoms. In the event that the triple's protest compares to a root layout list, the question is additionally embedded into the format list relating to its format ID. Formats records store exacting qualities alongside the key of their comparing group root. They are put away minimalistically and sectioned in sub records, both on plate and in primary memory. Layout records are commonly sorted by considering a lexical request on their strict qualities—however different requests can be determined by the database chairman when he pronounces the format roots. In that sense, format records are reminiscent of sections in a segment situated database framework.

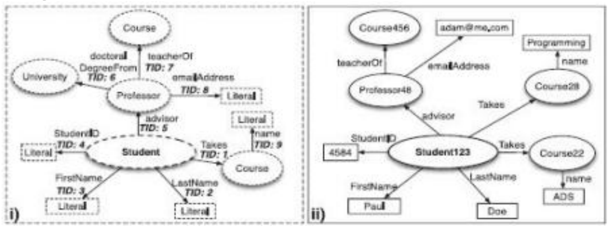

Fig. 1. A molecule template (i) along with one of its RDF molecules (ii).

4. SYSTEM OVERVIEW

Fig. 1 gives an improved design of Diplo-Cloud. Diplo-Cloud is a local, RDF database framework. It was intended to keep running on bunches of ware machines with a specific end goal to scale out effortlessly when taking care of greater RDF datasets. Our framework configuration takes after the engineering of numerous current cloudbased circulated frameworks, where one (Master) hub is in charge of collaborating with the customers and coordinating the operations performed by the other (Worker) hubs.

4.1 Master Node

The Master hub is made out of three principle subcomponents: a key list (c.f. Segment 3.1), accountable for encoding URIs and literals into minimal framework identifiers and of interpreting them back, a segment administrator (c.f. Segment 5), in charge of apportioning the RDF information into repeating subgraphs, and a conveyed question agent (c.f. Segment 6.3), in charge of parsing the approaching inquiry, modifying the question gets ready for the Workers, gathering lastly giving back the outcomes to the customer. Take note of that the Master hub can be duplicated at whatever point important to guarantee appropriate question stack adjusting and fault tolerance. The Master can likewise be copied to scale out the key file for to a great degree huge datasets, or to repeat the dataset on the Workers utilizing diverse dividing plans (all things considered, each new occurrence of the Master is in charge of one apportioning plan).

4.2 Worker Nodes

The Worker hubs hold the divided information and its relating neighborhood records, and are in charge of running subqueries and sending comes about back to the Master hub. Theoretically, the Workers are substantially less complex than the Master hub and are based on three fundamental information structures: i) a sort record, grouping all keys in view of their sorts ii) a progression of RDF atoms, putting away RDF information as extremely smaller sub charts, and iii) a particle list, putting away for each key the rundown of particles where the key can be found.

5 DATA PARTITIONING AND ALLOCATION

As specified in Section 2, triple-table and property-table hash-partitionings are right now the most widely recognized apportioning plans for conveyed RDF frameworks. While basic, such hash-partitionings methodicallly infers some dispersed coordination overhead (e.g., to execute joins/way traversals on the RDF chart), consequently making it unseemly for most huge scale groups and distributed computing situations showing high system latencies. The other two standard social dividing strategies, (tuple) round-robin and range parceling, are likewise imperfect for the information and setting we consider, since they would segment triples either aimlessly or in view of the subject URI/sort, subsequently truly constraining the parallelism of most administrators (e.g., since many occurrences having a similar sort would wind up on a similar hub). Apportioning RDF information in view of standard diagram dividing strategies (correspondingly to what proposes) is likewise from our point of view improper in a cloud setting, for three fundamental reasons:

Loss of semantics: standard diagram parceling instruments, consider unlabeled charts generally, and thus are absolutely rationalist tothe lavishness of a RDF chart including classes of hubs and edges.

Loss of parallelism: apportioning a RDF diagram based, for example, on a min-slice calculation will prompt exceptionally coarse parcels where a high number of related occurrences (for example connected to a similar sort or sharing connections to similar items) will be co-found, subsequently definitely constraining the level of parallelism of numerous administrators (e.g., projections or choices on specific sorts of instances).Limited versatility: at long last, endeavoring to segment vast RDF charts is improbable in cloud environments, given that cutting edge chart dividing strategies are inalienably brought together and information/CPU concentrated (as a narrative proof, we needed to get a capable server and let it keep running for a few hours to segment the biggest dataset we use in Section 7 utilizing METIS). DiploCloud has been considered starting from the earliest stage to bolster circulated information parceling and co-area plots in an effective and adaptable way. DiploCloud receives a middle arrangement between tuple-dividing and diagram apportioning by settling on a repeating, finegrained chart parceling procedure exploiting atom layouts. DiploCloud's atom formats catch repeating designs happening in the RDF information normally, by reviewing both the occurrence level (physical) and the construction level (coherent) information, consequently the expression physiological9 dividing.

5.1 Physiological Data Partitioning

We now characterize the three fundamental molecule-based information dividing procedures bolstered by our framework: Scope-k particles. The most straightforward strategy is to physically characterize various layout sorts (of course the framework considers assorted types) filling in as root hubs for the molecules, and then to co-find every single further hub that are specifically or in a roundabout way associated with the roots, up to given extension k.Scope-1 particles, for instance, co-situate in the atoms all root hubs with their immediate neighbors (cases or literals) as characterized by the formats. Scope-2 or 3 particles link perfect formats from the root hub (e.g.,ðstudent; takes; courseþ and ðcourse; hasid; xsd : integerþ) recursively up to profundity k to appear the joins around each root, to the detriment of quickly expanding stockpiling costs since much information is commonly recreated all things considered (see Section 7). The extent of the particles is characterized for this situation physically and includes information duplication. All information above Scope-1 is copied; this is the cost to pay keeping in mind the end goal to profit by pre-figured joins inside the particles, which essentially builds inquiry execution as we show in the accompanying. Manual dividing. Root hubs and the best approach to link the different formats can likewise be determined by hand by the database manager, who simply needs to compose a setup record indicating the roots and the way layouts ought to be connected to characterize the non specific state of every atom sort. Utilizing this system, the overseer fundamentally determines, in view of asset sorts, the correct way taking after which atoms ought to be physically augmented.

The framework then consequently copies information taking after the director's determination and pre-processes all joins inside the particles. This is commonly the best answer for moderately stable datasets and workloads whose primary components are notable. Versatile apportioning. At last, DiploCloud's most adaptable dividing calculation begins by characterizing degree 1 atoms of course, and after that adjusts the layouts taking after the inquiry workload. The framework keeps up a sliding-window w following the current history of the workload, and related measurements about the quantity of joins that must be performed and the implicating edges (e.g., missing colocation amongst understudies and courses bringing on an expansive number of joins). At that point at each time age , the framework: i) grows one atom layout by specifically connecting the edges (controls) that are in charge of the most signs up to a given edge for their maximal profundity and ii) diminishes (up to degree 1) every augmented particle whose expansions were not questioned amid the last age. In that way, our framework gradually adjusts to the workload and appears visit ways in the RDF chart while keeping the general size of the particles little. Thus to the two past strategies, when the extent of a particle is expanded, the framework copies the applicable bits of information and pre-processes the joins. The benefit of this strategy is that it starts with moderately basic and minimized information structures and afterward consequently adjusts to the dynamic workload by expanding and diminishing the extent of particular atoms, i.e., by including and expelling pre-registered ways in light of layout determinations. On account of an exceptionally dynamic workload, the framework won't adjust the structures keeping in mind the end goal to maintain a strategic distance from continuous revising costs that would not by effortlessly amortized by the change in inquiry handling.

6 COMMON OPERATIONS

Algorithm 2. Query Execution Algorithm with Join on the Master Node

```
1: procedure EXECUTEQUERY(a,b)
2: for all BGP in QUERYdo BGP - Basig Graph Pattern
3: if BGP.subject then
4: molecules GetMolecule(subject)
5: else if BGP.object then
6: molecules GetMolecules(object)
7: end if
8: for all molecules do
9: check if the molecule matches the BGP
10: for all TP in BGP do "TP - Triple Pattern
11: if TP.subject != molecule.subject then
12: nextMolecule
13: end if
14: if TP.predicate != molecule.predicate then
15: nextMolecule
16: end if
17: if TP.object != molecule.object then
18: nextMolecule
19: end if
20: end For
21: the molecule matches the BGP, so we can retrieve entities
22: resultBGP GetEntities(molecule,BGP)
23: end For
24: results result<-BGP
25: end for
26: SendToMasterNode(results)
 27: end procedure
28: On the Master do Hash Join
```

7. CONCLUSION

DiploCloud is a proficient and versatile framework for overseeing RDF information in the cloud. From our point of view, it strikes an ideal harmony between intra-administrator parallelism and information colocation by considering repeating, fine-grained physiological RDF parcels and conveyed information allotment plans, driving however to conceivably greater information (repetition presented by higher extensions or versatile particles) and to more mind boggling supplements and updates. DiploCloud is especially suited to bunches of ware machines and cloud situations where arrange latencies can be high, since it efficiently tries to stay away from all mind boggling and dispersed operations for inquiry execution. Our exploratory assessment demonstrated that it positively thinks about to cutting edge frameworks in such conditions. We plan to keep creating DiploCloud in a few bearings: First, we plan to incorporate some further pressure systems (e.g., HDT). We plan to deal with a programmed formats disclosure in light of regular examples and untyped components. Additionally, we plan to chip away at coordinating a surmising motor into DiploCloud to bolster a bigger arrangement of semantic imperatives and questions locally. Finally, we are presently trying and developing our framework with a few accomplices keeping in mind the end goal to oversee to a great degree expansive scale, circulated RDF datasets with regards to bioinformatics applications.

REFERENCES

[1] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. van Pelt, “GridVine: Building Internet-scale semantic overlay networks,” in Proc. Int. Semantic Web Conf., 2004, pp. 107–121.

[2] P. Cudre-Mauroux, S. Agarwal, and K. Aberer, “GridVine: An infrastructure for peer information management,” IEEE Internet Comput., vol. 11, no. 5, pp. 36–44, Sep./Oct. 2007.

[3] M. Wylot, J. Pont, M. Wisniewski, and P. Cudre-Mauroux. (2011). dipLODocus[RDF]: Short and long-tail RDF analytics for massive webs of data. Proc. 10th Int. Conf. Semantic Web - Vol. Part I,pp. 778–793 [Online]. Available: http://dl.acm.org/citation.cfm?id=2063016.2063066

[4] M. Wylot, P. Cudre-Mauroux, and P. Groth, “TripleProv: Efficient processing of lineage queries in a native RDF store,” in Proc. 23rd Int. Conf. World Wide Web, 2014, pp. 455–466.

[5] M. Wylot, P. Cudre-Mauroux, and P. Groth, “Executing provenance- enabled queries over web data,” in Proc. 24th Int. Conf.World Wide Web, 2015, pp. 1275–1285.

[6] B. Haslhofer, E. M. Roochi, B. Schandl, and S. Zander. (2011). Europeana RDF store report. Univ. Vienna, Wien, Austria, Tech.Rep. [Online]. Available: http://eprints.cs.univie.ac.at/2833/1/europeana_ts_report.pdf

[7] Y. Guo, Z. Pan, and J. Heflin, “An evaluation of knowledge base systems for large OWL datasets,” in Proc. Int. Semantic Web Conf.,2004, pp. 274–288.

[8] Faye, O. Cure, and Blin, “A survey of RDF storage approaches,”ARIMA J., vol. 15, pp. 11–35, 2012.

[9] B. Liu and B. Hu, “An Evaluation of RDF Storage Systems for Large Data Applications,” in Proc. 1st Int. Conf. Semantics, Knowl.Grid, Nov. 2005, p. 59.

[10] Z. Kaoudi and I. Manolescu, “RDF in the clouds: A survey,” VLDB J. Int. J. Very Large Data Bases, vol. 24, no. 1, pp. 67–91, 2015.

[11] C. Weiss, P. Karras, and A. Bernstein, “Hexastore: sextuple indexing for semantic web data management,” Proc. VLDB Endowment,vol. 1, no. 1, pp. 1008–1019, 2008.

[12] T. Neumann and G. Weikum, “RDF-3X: A RISC-style engine for RDF,” Proc. VLDB Endowment, vol. 1, no. 1, pp. 647–659, 2008.

[13] A. Harth and S. Decker, "Optimized index structures for querying RDF from the web," in Proc. IEEE 3rd Latin Am. Web Congr., 2005, pp. 71–80.
[14] M. Atre and J. A. Hendler, "BitMat: A main memory bit-matrix of RDF triples," in Proc. 5th Int. Workshop Scalable Semantic Web Knowl. Base Syst., 2009, p. 33.
[15] K. Wilkinson, C. Sayers, H. A. Kuno, and D. Reynolds, "Efficient RDF Storage and Retrieval in Jena2," in Proc. 1st Int. Workshop Semantic Web Databases, 2003, pp. 131–150.

AUTHORS PROFILE:

Kancherla Rajasree is pursuing M.Tech from department of Computer Science and Engineering at Eluru College of Engineering and Technology, Eluru.

Mr. Yarlagadda Siva Koteswararao completed M.TECH, presently working as Assistant Professor in the department of Computer Science and Engineering at Eluru College of Engineering and Technology, Eluru, with 7 years of experience.