© February 2017 | IJIRT | Volume 3 Issue 9 | ISSN: 2349-6002

INCREMENTAL ONTOLOGY INFERENCE FOR SEMANTIC WEB BASED ON MAPREDUCE APPROACH

T. Revathi[1], U. Uma Devi[2]
[1]*Senior Professor& Head, Dept of IT, Mepco Schlenk Engineering College, Sivakasi, Tamilnadu, India*
[2]*PG Scholar, Dept of IT, Mepco Schlenk Engineering College, Sivakasi, Tamilnadu, India*

***Abstract* — In the fast growing internet world, web content increase day by day. This demands the knowledge searching and reasoning in this big data. The knowledge is represented in the semantic web using web ontology languages. Existing methods take long time to derive inferences and also it performs full reasoning when new data stream arrives. In this paper an Incremental Ontology Inference (IOI) Method for handling large number of triples (subject, predicate, and object) is proposed. In IOI, the triples for each type are collected and a forest like data structure is built and then performs reasoning. The storage requirement is also reduced by merging the triple reasoned from other triple into a set of triples with the same values. Hence, it provides fast traversal of triples in the tree and retrieves the query results efficiently. MapReduce paradigm is used to implement the proposed approach. The results for user query are reasoned and retrieved effectively.**

***Index Terms*— Big data, Knowledge Searching, MapReduce, Ontology Inference, Semantic Web.**

I. INTRODUCTION

Currently, the web is the major source of giant data. Every day the flow of data increases at the web. This makes the challenge of identifying the useful things from the available data. The normal human effort is not enough to infer knowledge from such rich web resource. The Machine should also be having the knowledge to understand this data deluge and reason information. Semantic web [1] helps in making the machines realize the web. The resources on the web are expressed with the web ontology language, and this aid computer to identify the essential information from this current web. The major application of semantic web includes healthcare and life sciences [2], machine intelligence [3], and e-marketplace activities [4].

The statistics [5] shows that the size of the semantic web is approximately to contain 4.4 billion triples in 2009 and it is currently 20 billion triples. Its development rate is still increasing. Hence, this creates the problem of knowledge hunting over such big data.

Obtaining inference from incremental web resource faces three challenges: 1) Infer knowledge from correct triple is difficult due to its dispersed data; 2) Increasing Volume of data needs to be processed in a scalable manner; 3) Satisfying user query desires high-speed processing.

The fundamental description of information on the web is the Resource Description Framework (RDF) [6]. It is vital for the semantic web. Every statement on the web is symbolized to a triple. It expresses the relationship between the two resources. For example, Fig. 1 shows one way of representing the statement, "India is a country" in RDF is as the triple: "India" is the subject, "rdf:type" is the predicate, and "Country" is the object.

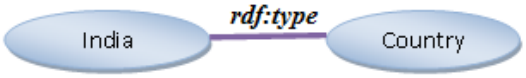

Fig.1 Representation Of Triple

RDF Schema (RDFS) provides vocabulary for describing how properties and classes are intended to be used together in RDF data. The Web Ontology Language (OWL) [7] extends RDF and RDFS. Its major intention is to bring the expressive and reasoning power of description logic to the semantic web.OWL consists of three languages with

increasing expressivity: OWL Lite, OWL DL and OWL Full. All the three languages allow us to describe classes, properties, and instances. The weaker languages have limitations on what can be declared or how it may be declared. The Inference is derived from this knowledge represented in the form of triples. Some triples in groups with others can provide new inference. This is done through the set of policies. Machines uses this policy set to derive the inference from the available triples.

II. LITERATURE REVIEW

These days reasoning semantic web have received much attention from both academic world and the business world. Lots of reasoning engines have been developed to hold the reasoning over semantic web.

Guo *et al*., [4] proposed a novel RuleXPM (Rule XML Product Map) method that infers the next generation electronic marketplace (e-marketplace) activities. The RuleXPM architecture supports the concept separation strategy and makes the designed RuleXPM inference engine generic and suitable for use in all types of e-marketplace. In this architecture, the inference engine is modular, i.e., each inference module is independent and reusable and the data in use can be dynamically generated, and is contextual.

Anagnostopoulos and Hadjiefthymiades [8], proposed fuzzy inference engines based on the knowledge-representation (KR) model to enhance the context inference. The capability of a context aware system is to classify context and infer specific situations can be facilitated by proper KR models. A Fuzzy set based model can accommodate the vagueness inherent in context capturing. A fuzzy set is used for representing imprecise context in a human understandable form. This methodology is generic and can be applied to different inference schemes in order to improve the inference capability of the classifier and deal with mutual exclusion inference. This model generates specific complementary fuzzy rules used for increasing the accuracy of the classification procedure for the well specified information in Semantic web. Applications can handle context as flexibly as their users would expect by using this method, but it is not suitable for all situations of the user.

Paulheim and Bizer, [9] studied the problem of inference with noisy data and presented the SDType method based on the statistical distribution of types in RDF datasets to deal with noisy data. Milea *et al*., [10] presented a temporal extension of the OWL for expressing time-dependent information. These ontology-reasoning methods are conducted on a single machine or local cluster. The reasoning speed is directly dependent on the scale of the ontology, which is not suitable for a large ontology base.

To deal with a large base, some researchers moved to distributed reasoning methods. Weaver and Hendler, [11] presented a method for materializing the complete finite RDFS closure. It is the first method to provide RDFS inference on such large data sets in such low times and scalable manner. This maintains soundness and completeness without requiring any cumbersome preparation of the data. This method increases the processing speed by means of parallel inference. It lacks with scalability and expressivity. Urbani *et al*., [12] proposed a scalable distributed reasoning method by some nontrivial optimisations for encoding the RDFS ruleset in MapReduce and exploited the MapReduce framework for efficient large-scale Semantic Web reasoning and implements on the top of Hadoop. This reasoning technique performs quick reasoning using Hadoop Distributed File System (HDFS) and high data correlation. The drawback of using this method is it does not focus on quality of reasoning. Schlicht and Stuckenschmidt, [13] highlighted the drawback of the MapReduce-based reasoning and then introduced a Mapresolve method for more expressive logics. It adapts the standard method for distributed resolution that avoids repetition of resolved inferences. For the limited expressivity of RDFS, the repetition can be avoided because every MapReduce job is executed only once Dean and Ghemawat, [14]. For each step, the clause sets are parsed and written to disc, generates needless overhead.

Still, these techniques don't consider the effect of incremental data volume, and does not show the processing of users' queries. To answer the demands on a user query, they need to obtain the entire RDF closure by reasoning and save them to hard disk. The data volume of RDF closure is ordinarily larger than original RDF data. The storage of RDF closure is thus not a small amount and the query on it takes nontrivial time. Furthermore, as the data volume increases and the ontology base is

updated, these methods require the re-computation of the entire RDF closure every time when new data arrive. To avoid such time-consuming process, incremental reasoning techniques are proposed.

Urbani *et al*., [15] proposed a scalable parallel inference using MapReduce. This method calculates the RDF closure for large scale RDF dataset by adopting algorithms to process the statements based on input data as incremental reasoning. This technique identifies the accurate status, which either existing or newer one does not provide the relationship between the newly arrived and existing data. Grau *et al*., presented an incremental reasoning approach based on modules that can reuse the information obtained from the previous versions of ontology [16]. This method is used for OWL reasoning speed is a huge problem while using this method.

Bo Liu et al., proposed an Incremental and Distributed inference method based on Mapreduce and Hadoop [17]. This method speeds up the updating process with newly arrived data and fulfills the requirements of end users for online queries that leverage the old and new data to minimize the updating time and reduce the reasoning time when facing big RDF datasets. Though this inference method speeds up the updating and reasoning, it concentrates only on the RDF and does not consider other web definition languages like OWL.

In this paper, we propose an IOI method that reasons out from the ontology describing web contents considering the OWL set operator elements, Inverse element and RDFS elements. To make the reasoning in the faster way Hadoop framework [18] can well control over the existing and newly derived triples. This reduces the inference time issues faced by large web contents.

III. ONTOLOGY INFERENCE BASED ON OWL AND RDF ELEMENTS

This section presents the IOI over large scale web contents described with web ontology description languages include RDF and OWL. Fig. 2 shows the architectural layout for the inference, presenting the steps involved in the ontology reasoning.

The input to the IOI method is in the form of triples. The URLs in the triple are encoded with the corresponding hash code initially. Then OWL+RDF reasoning step process the encoded triples and obtain new reasoned triple incrementally. After that Forest Creation (FC)/Effective storage (ES) modules are performed to construct tree based storage of triples. The query processing step takes the user's query and answers them with results obtained after reasoning.

A. Hash Coding

The web page contains many statements and they are represented in the semantic web as the sequence of long URIs to uniquely identify each web resource. This makes the processing and reasoning over the URIs complexity. To solve this issue, the hash code for each URI in the triple is generated. A unique numeric identifier is assigned to each one of them.

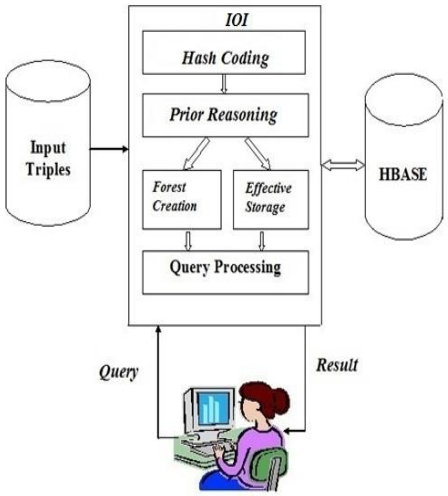

Fig. 2 Incremental Ontology Inference

Triple file is given as the input to the MapReduce Program. It splits the triples into <subject, predicate, object> line by line in the map function. Then the reduce function calculates the hash code for each unique URI emitted from the map.

The steps after calculating the hash code are all based on only with the encoded triple format obtained from hash coding to reduce the storage and to speed up the upcoming inference process.

B. Prior Reasoning with OWL Elements

The main inference exists in the triples having the predicate as property and class. Therefore, before reasoning out from the RDF elements like

rdfs:subProperty, rdfs:subClassOf, rdfs:domain, and rdfs:range, we have to reason out of the OWL elements. Because the triples infers from owl:unionOf, owl:intersectionOf, and owl:inverseOf can also trigger other inference of class, domain, and range. Therefore, prior reasoning with OWL elements is done to make the reasoning process better.

Two reasoning steps consider here are reasoning from Set Operators and Inverse. OWL supports set operators like union and intersection elements. Algorithm 1 shows the reasoning over set operator and Algorithm 2 shows the reasoning over Inverse element.

Algorithm 1 - Reasoning over Set operator
Input: All Triples T with predicate owl:unionOf and owl:intersectionof
{
for each triple t <s,p,o> in T
  if p is owl:unionOf
    add triple <o,rdfs:subClassOf,s>to R
  if p is owl:intersectionOf
    add triple <s,rdfs:subClassOf,o>to R
return R
}
Output: Reasoned rdfs:subClassOf Triples R

Algorithm 2 - Reasoning over Inverse
Input: All triples T having predicate as owl:inverseOf
{
for each triple t<s,p,o> in T
  if p is owl:inverseOf
    for each triple t in T
      if <s,rdfs:domain,o1> exists in T
        add triple <o, rdfs:domain, o1> to R
      if <s,rdfs:range,o1> exists in T
        add triple <o, rdfs:range, o1> to R
return R
}
Output: Reasoned rdfs:domain & rdfs:range triples R

The owl:unionOf property relates a class to a set of class descriptions. The inference exist with this is, if a class X is the owl:unionOf a set of classes, say A, B, and C, then each of A, B, and C, is rdfs:subClassOf X. For example SweetFruit and NonSweetFruit are the unionOf the Fruit class, then we can infer that both SweetFruit and NonSweetFruit are the rdfs:subClassOf the class Fruit.

Similar to the owl:unionOf property owl:intersectionOf also relates a class to a set of class descriptions. The inference exist with this is, if a class X is the owl:intersectionOf a list of classes, say A, B, and C, then X is rdfs:subClassOf of each of A,B, and C. For example, if WhiteWine is *exactly* the intersection of the class Wine and the set of things that are white in color, then we can infer that WhiteWine is an rdfs:subClassOf Wine.

Properties always have a direction, from domain to range. It is difficult to define relations in both directions: persons own cars, cars are owned by persons. The owl:inverseOf construct can be used to define such an inverse relation between properties. The inference exist here is, If A is owl:inverseOf B and A domain is U Then B domain U can be inferred, and similarly if A range is U Then B range is also U.

C. Forest Creation

In order to efficiently handle the inference and avoid the searching over entire ontology base the forest data structure is maintained. The forest may consist of one or multiple trees. The tree gets updated when incremental triples occur. Each node in a tree stands for a subject or object, and the directed link between them shows their sub-property relation. FC is further divided into Property FC (PFC), Class FC (CFC), and Domain/Range FC **(**DRFC).

PFC is a directed forest constructed based on all the triples that have predicate rdfs:subPropertyOf, or have predicate rdf:type and object rdfs:ContainerMembershipProperty. Fig. 3 shows the PFC Creation, In this ‘hasSon’ is the rdfs:subPropertyOf ‘hasChild’. So the directed graph from ‘hasSon’ to ‘hasChild’ is drawn. Similarly for all the triples with predicate rdfs:subPropertyOf(RPO) is drawn.

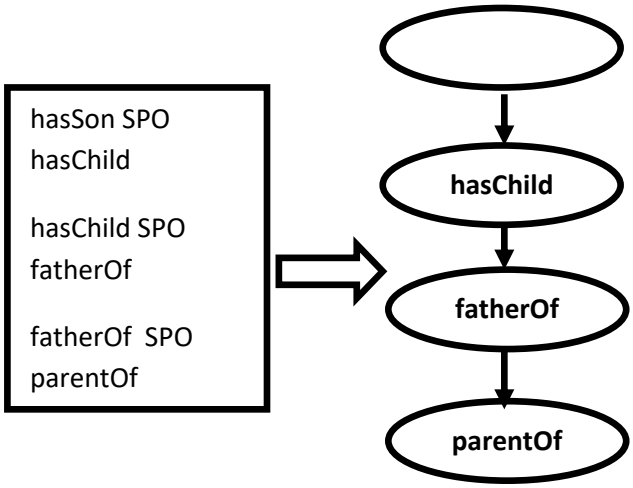

Fig. 3 PFC Creation

CFC is a directed forest constructed based on all the triples that have predicate rdfs:subClassOf, or have predicate rdf:type and object rdfs:Datatype or rdfs:Class. Each node in a tree stands for a subject or object, and the directed link between them shows their sub-class relation. Fig. 4 shows the CFC Creation, In this 'Europe' is the rdfs:subClassOf 'Country'. So the directed graph from 'Europe' to 'Country' is drawn. Similarly for all the triples with predicate rdfs:subClassOf (SCO) is drawn.

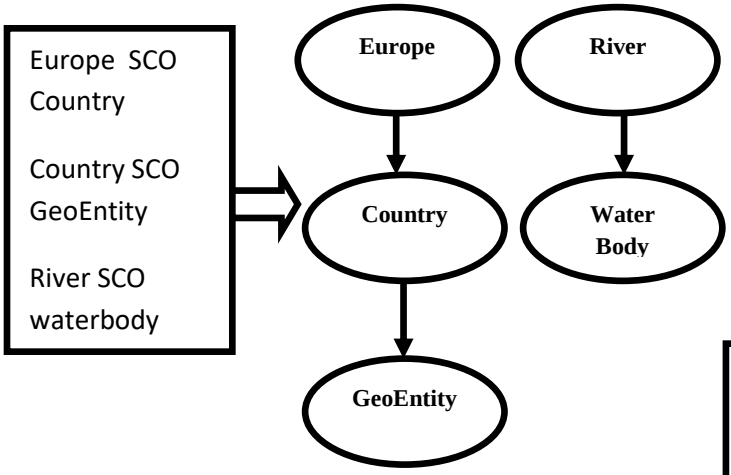

Fig. 4 CFC Creation

DRFC is a directed forest constructed based on the triples that have predicates rdfs:domain or rdfs:range, in which each node in the tree stands for a subject or object and the directed link shows the domain or range relation between the node pair. Fig. 5 shows the DRFC Creation, In this 'NorthCorner' has the rdfs:range as 'Location'. So the directed graph from 'NorthCorner' to 'Location' is drawn. To differentiate the rdfs:domain (R) and rdfs:range (D) is marked with the dotted line and dark line respectively. Similarly for all the triples with predicate rdfs:range and rdfs:domain are drawn.

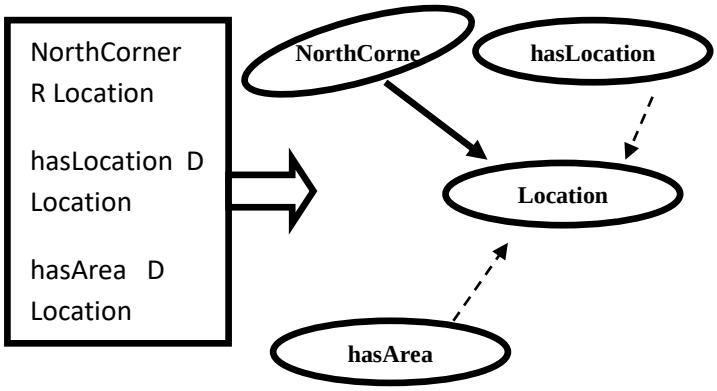

Fig. 5 DRFC Creation

D. Reasoning over FC

The next thing after FC is to perform reasoning. Since FC contains set of trees, Traversing in the tree is by two ways, either in the forward or reverse direction of the tree. If we start from root or endpoint to search node, then it is called "forward path" otherwise if we traverse from search node to the root or endpoint then it is called "reverse path".

Reasoning FC is also done in three ways similar to the creation process. An Algorithm 3, 4, and 5 shows the Reasoning over PFC, CFC, and DRFC respectively. The input to the Algorithm is Assertional Triples (AT). AT is a triple not having the predicate as subProperty, subClassOf, domain and range.

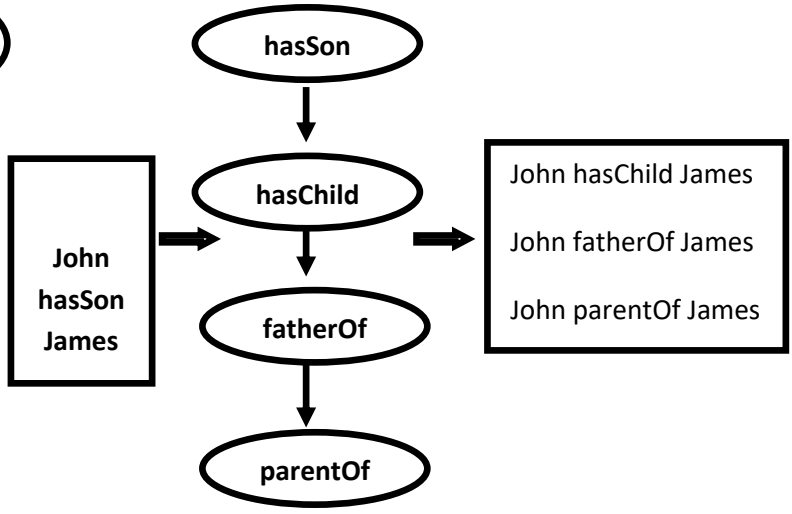

Fig. 6 Reasoning Of PFC

Algorithm 3 describes that for the given AT if its predicate exists in the PFC then new triple be generated. Fig. 6 shows the example for reasoning

over PFC. Given the triple <john hasSon james>, the predicate hasSon exist in the PFC. Therefore, traverse through the forward path of PFC. The path is hasChild→ fatherOf → parentOf. Then for each node in the forward path generate the triple by replacing the predicate with that node. Thus the final results are {<John hasChild James>, <John fatherOf James>, <John ParentOf James>}.

Algorithm 3 - Reasoning over PFC
Input: All AT and PFC
{
for each node p in PFC
F<- forward path of p
for each node f in F
add triple t <s,p,o> to R
return R
}
Output: Reasoned triple R

Algorithm 4 describes that for the given AT and the DRFC, new triples can be reasoned. For the given triple, if its predicate is rdfs:domain and object 'o' exist in the DRFC then new triple be generated. Similarly, if its predicate is rdfs:range and subject 's' exists in the DRFC then new triple be generated. Reasoning over DRFC is done prior to the CFC since it infers triples that trigger the reasoning of CFC.

Algorithm 4 - Reasoning over DRFC
Input: All AT and DRFC
{
for each node p in DRFC
if p has a domain edge linked to node c
add triple <s,rdf:type,c> to R
if p has a range edge linked to node c
add triple <o,rdf:type,c> to R
return R
}
Output: Reasoned triples R that triggers CFC Inference

Algorithm 5 describes that for the given AT if its predicate is rdf:type and object o exist in the CFC then new triple be generated. Fig. 7 shows the example for reasoning over CFC. Given the triple <Austria rdf:type Europe>, the object Europe exist in the CFC. Therefore, traverse through the forward path of CFC. The forward path is Europe→Country → Geographic Entity. Then for each node in the forward path generate the triple by replacing the object with that node. Thus the final results are {<Austria rdf:type Country> , <Austria rdf:type Geo Entity> }.

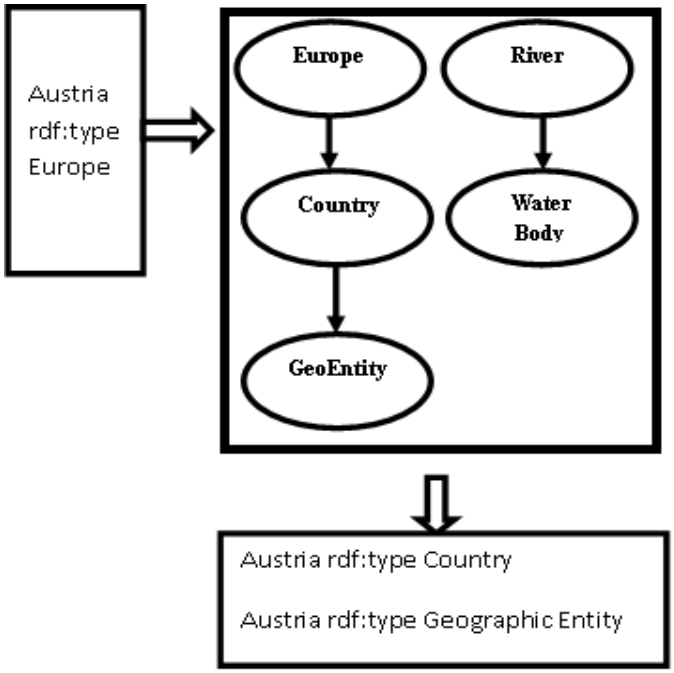

Fig.7 Reasoning CFC

Algorithm 5 - Reasoning over CFC
Input: All AT and CFC
{
for each node o in CFC
F <-- forward path of o
for each node f in F
add triple t<s,rdf:type,f> to R
return R
}
Output: Reasoned rdf:type triples R

E. ES Making

The AT we obtained from reasoning can be derived from others, so we can avoid storing all of them to make the storage efficient. For example, given two triples AT1: <*s*1, *hasSon*, *o*1> and AT2: <*s*1, *fatherOf*, *o*1> and the PFC in the derived triples for T1 are <*s*1, *hasSon*, *o*1>, <*s*1, *hasChild*,*o*1>, <*s*1, *fatherOf*, *o*1>, and <*s*1, *parentOf*, *o*1>, and those for T2 are <*s*1, *fatherOf*, *o*1> and <*s*1, *parentOf*, *o*1> which are all duplicated ones.

ES reduce the storage of AT that can be inferred from others. It is subdivided into two types according to FC as Property Effective Storage (PES)

and Class Effective Storage (CES). The information that cannot be derived is alone stored in ES.

PES concentrate on the AT that is having the same subject and object, but the predicates are different for each of AT. Similarly CES concentrate on the AT that is having the same subject and the predicate should be of rdf:type, but the object field of AT is different for each of them. Algorithm 6 and 7 are aimed to build PES and CES respectively.

The complexity of Algorithm 6 and 7 is O(mn) where 'm' represents the number of triples in T and 'n' represents the number of nodes in PFC/CFC.

Algorithm 6 - Making PES

Input: PFC and a set of triples that have the same subject and object T = {$<S_i, P_1, O_i>$, $<S_i, P_2, O_i>$,. }

{
T← {$<S_i, P_1, O_i>$, $<S_i, P_2, O_i>$,. . . }
P = {P_1,P_2,..} ←all the predicates in T
for each P_j in P
Q← p_j in the forward path of PFC
for each q_i in Q
if q_i exists in P
Remove q_i from P
return PES = {S_i ,O_i,P}
}

Output: PES = {S_i ,O_i, $<P_i>$}

Algorithm 7 - Making CES

Input: CFC and a set of triples that have the same subject and predicate is rdf:type, T= { $<S_i, rdf:type, O_1>$, $<S_i, rdf:type, O_2>$,… }

{
T← {$<S_i, rdf:type, O_1>$, $<S_i, rdf:type, O_2>$,… }
O= {O_1, O_2...} ←all the objects in T
for each O_j in O
Q← O_j in the forward path of CFC
for each C_i in Q
if C_i exists in O
remove C_i from O
return CES = {s_i,O}
}

Output: CES = {S_i,$<O_i>$}

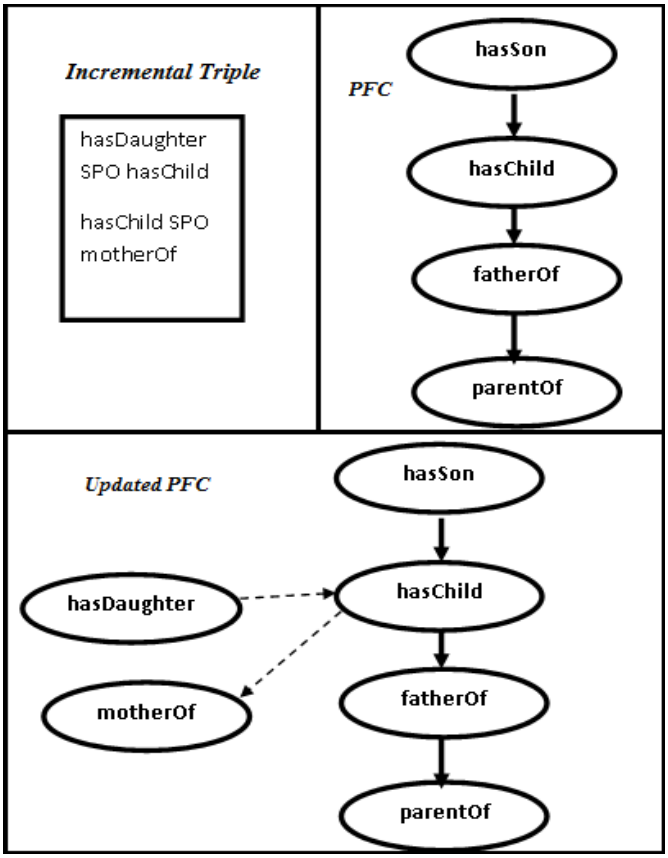

Fig. 8 Incrementing FC and ES

F. Updating FC and ES

New triples are added to the tree without reconstructing the full tree. It was done by simply updating the already existing tree structure. When new triples arrive, new edges are added to the existing FC. Now we have two kinds of edges, i.e., existing edges referring to the triples that exist in the original FC, and incremental ones to those who's subject or object or both do not exist in the FC.

Fig. 8 shows the incremental update of FC. Dark lines shows the already existing edges and dotted edges represent the incremental edges that are updated.

The steps for updating the FC and ES are as follows.

1. Generate new PFC by adding new edges to the existing PFC.
2. Generate incremental PES based on the input triples, add the incremental PES to the existing PES, and run Algorithm 6 to generate new PES.

3. Generate incremental DRFC based on the incremental triples and add incremental DRFC to the existing DRFC.
4. For the PES the predicate is in the reverse path of the incremental edges though the forward path of the incremental edge contains nodes in DRFC, generate the AT using Algorithm 4.
5. Generate new CFC by adding new edges to the existing CFC.
6. Generate the incremental CES based on the incremental AT and the triples generated in step 4, add the incremental CES to the existing CES, and run Algorithm 7 to make new CES.

G. Query Retrieval

The main goal of Calculating FC and ES is to reduce the query time. In order to search the node in the tree, traversing is possible in three ways. They are Forward, Backward, and Reachable node search. Based on these ways there exist six methods to retrieve the query result.

Method 1: Input: <subject, object> pair, Output: predicate list

Search for <subject, object> pair in the PES and obtain the predicate list P_i. Here i is the count of different predicate obtained. Check for all the predicate values of i in the PFC and emit the list of nodes that are available in the forward path.

Method 2: Input: <subject>and predicate <rdf:type>, Output: object list

Search <subject> in the CES and obtain the object list O_i . Here i is the count of different objects obtained. Check for all the subject values of i in the CFC and emit the list of nodes that are available in the forward path.

Method 3: Input: <predicate>, Output: <subject, object> pair

Search for the <predicate> in the reverse path of the PFC and obtain the list of nodes P_i in that path. Here i is the count of different predicate obtained. Check for all predicate values of i in the PES and emit the <subject, object> pair.

Method 4: Input: predicate <rdf:type> and <object>, Output: <subject>

Search for the <object> in the reverse path of the CFC and obtain the list of nodes O_i in that path. Here i is the count of different objects obtained. Check for all object values of i in the CES and emit the <subject>.

Method 5: Input: <subject> and predicate <rdfs:subPropertyOf>, Output: object list

Search for the <subject> in the PFC. If exists in the tree, then emit all the nodes in the forward path.

Method 6: Input: <subject> and predicate <rdfs:subClassOf>, Output: object list

Search for the <subject> in the CFC. If exists in the tree, then emit all the nodes in the forward path.

Basically eight types of query statements, including *<?x ?y ?z>*, *<?x p ?z>*, *<s ?y ?z>*, *<?x ?y o>*, *<?x p o>*, *<s p ?z>*, *<s ?y o>*, and *<s p o>* are available. For each type the procedure to execute the query is as follows.

1) **<?x ?y ?z>:**Get all the <subject,object> pairs in the PES. Give them as input to Method1. Get the entire <subject> from CES and give them as input to Method 2. Output the entire result obtains from both the methods.
2) **<?x p ?z >:**If 'p' is rdf:type then give it as input to method 2. If 'p' is rdfs:subpropertyOf then give it as input to method 5. If 'p' is rdfs:subClassOf then give it as input to method 6. Otherwise if 'p' doesn't satisfy any of the above if condition then gives the 'p' as the input to method 3. It outputs all the <?x ?z> pairs as the result.
3) **<s ?y o>:**Give <s,o> as input to method1 and emit the predicate result. Also give the <o> as input to method 4, if's' exist in the result, then predicate <rdf: type> is also emitted as output.
4) **< s ?y ?z>:**Get all the <subject, object>pair in PES for the 's'. Then execute the query type 3 for the obtained <subject,object>pair. Also check whether's' exists in CES and obtain the object list 'O' by running method 2. Emit the result as <s rdf:type O>.
5) **<?x ?y o>:** Get all the <subject,object> pair in PES with 'o'. Then execute the query type 3 for the obtained <subject,object>pair. Also check whether 'o' exists in CES and obtain the subject list S by running the method 4. Emit the result as <S rdf:type o>.
6) **<?x p o>:**Execute the query type 2 and filter the result based on 'o'.

7) **<s p ?z>:**Execute the query type 2 and filter the result based on 's'.
8) **<s p o>:**Execute the query type 2 and filter the result with both 's' and 'o'.

If for all the query type there is no result obtained means, then return empty to the user.

IV. RESULTS & ANALYSIS

To implement the proposed approach, Hadoop framework is used which enable the MapReduce technology. We use the Hadoop-2.6.0 and Hbase-0.98.9 for our system. The System is configured with 4 GB memory and 500 GB storage. Hadoop is an open source Java based implementation which allows the distributed processing of large scale datasets.

The triples for the experiment are taken from linked open vocabularies [19]. The geography dataset OWL file is converted to the triples format. It results into 16776 triples. The main core of the IOI system is to process these triples and derive inference from a set of MapReduce programs which are written by the algorithms described in this paper. The Hadoop platform supports the HBase [20] for storage of input and intermediate processing of the triples.

To efficiently compress the input triple elements Hash Coding is done initially to perform the reasoning. Then prior reasoning over OWL elements is done to gather the newly reasoned triples which can be given as input to the next step of IOI. To construct the FC, the matched triples which can be given as input to the creation process are collected in the Map function and it is emitted to the Reduce function. In Reduce the actual construction process is carried out. Since programs are split and executed this performance efficiency is helpful in implementing the FC construction. After performing the FC and reasoning process, the result of triples count is 31074. Then ES is computed to reduce the AT storage that can be inferred from other triples. This resulted in a set of triples with the same subject and object, but with different predicate list i.e. $\{S_i,O_i,P\}$ pattern is 1968 and sets of triples with the same subject and a different object list, i.e.$\{S_i,<O_i>\}$ pattern is 2106. Thus storage space is largely reduced. Finally the result of various query type of user is retrieved from the FC and ES efficiently. This reduces the query processing time.

We compare our reasoning result with the IDRM [17] implemented by Bo Liu et al., and analyze the result. The result of analysis in Table 1 shows that IOI reasons out more number of triples when compared with IDRM. For IDRM reasons only the RDF property, class, domain and range elements. But IOI also infers the three OWL elements such as union, intersection and inverse which again results in the RDFS elements like subClassOf, domain and range. These inferred triples were also considered in constructing the FC and ES. Hence IOI based reasoning provides better results of nodes in the CFC and DRFC. This helps in answering the user query with more reasoning ability of IOI.

Table1. Comparison Result

Geography OWL dataset	**IDRM**	**IOI**
Number of Triples with 'subClassOf' predicate	3941	6345
Number of Triples with 'domain/range' predicate	1904	2884
Node Count in CFC	3794	4037
Node Count in DRFC	1470	1582
Triples after reasoning	14644	31074
Set of triples with $\{S_i,O_i,P\}$ pattern	1251	1968
Set of triples with $\{S_i,<O_i>\}$ pattern	1519	2106

V. CONCLUSION AND FUTUREWORK

In this paper, reasoning over the semantic data is performed. Large volume of data makes the reasoning process a challenging one. To avoid the complexity of reasoning, IOI method is proposed. The experiment is conducted using both IOI and IDRM [17]. The results show that IOI infers more number of triples than other reasoning methods comparatively because it infers the set and inverse elements also. In future, the method can be enhanced with more properties of OWL other than that considered here. Also the efficiency of the results obtained by the user query can be further improved.

REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American,284(5):34-43, 2001.

[2] M. S. Marshall *et al.*, "Emerging practices for mapping and linking life sciences data using RDF—A case series," *J. Web Semantics*, vol. 14, pp. 2–13, Jul. 2012.

[3] M. Nagy and M. Vargas-Vera, "Multiagent ontology mapping framework for the Semantic Web," *IEEE Trans. Syst., Man, Cybern. A, Syst.,Humans*, vol. 41, no. 4, pp. 693–704, Jul.2011.

[4] J. Guo, L. Xu, Z. Gong, C.-P. Che, and S. S. Chaudhry, "Semantic inference on heterogeneous e-marketplace activities," *IEEE Trans. Syst.,Man, Cybern. A, Syst., Humans*, vol. 42, no. 2, pp. 316–330, Mar. 2012.

[5] Linking Open Data on the Semantic Web [Online].Available: http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics

[6]P.Hayes,(ed.) RDF Semantics. W3C Recommendation, 2004.

[7] P. Patel-Schneider, P. Hayes, and I. P. Horrocks, "Web Ontology Language (OWL) abstract syntax and semantics," W3C Recommendation,2004.

[8] C. Anagnostopoulos and S. Hadjiefthymiades, "Advanced inference in situation-aware Sep. computing," *IEEE Trans. Syst., Man, Cybern. A, Syst.,Humans*, vol. 39, no. 5, pp. 1108–1115,2009.

[9] H. Paulheim and C. Bizer, "Type inference on noisy RDF data," in *Proc.ISWC*, Sydney, NSW, Australia, 2013, pp. 510–525.

[10] V. Milea, F. Frasincar, and U. Kaymak, "tOWL: A temporal web ontology language," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42,no. 1, pp. 268–281, Feb. 2012.

[11] J. Weaver and J. Hendler, "Parallel materialization of the finite RDFS closure for hundreds of millions of triples," in *Proc. ISWC*, Chantilly, VA, USA, 2009, pp. 682–697.

[12] J. Urbani, S. Kotoulas, E. Oren, and F. Harmelen, "Scalable distributed reasoning using mapreduce," in *Proc. 8th Int. Semantic Web Conf.*, Chantilly, VA, USA, Oct. 2009, pp. 634–649.

[13] A. Schlicht and H. Stuckenschmidt, "MapResolve," in *Proc. 5th Int. Conf. RR*, Galway, Ireland, Aug. 2011, pp. 294–299.

[14] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[15] J. Urbani, S. Kotoulas, J. Maassen, F. V. Harmelen, and H. Bal, "WebPIE: A web-scale parallel inference engine using mapreduce," *J. Web Semantics*, vol. 10, pp. 59–75, Jan. 2012.

[16] B. C. Grau, C. Halaschek-Wiener, and Y. Kazakov, "History matters: Incremental ontology reasoning using modules," in *Proc. ISWC/ASWC*,Busan, Korea, 2007, pp. 183–196.

[17] Bo Liu, Member, IEEE, Keman Huang, Jianqiang Li, and MengChu Zhou, "An Incremental and Distributed Inference Method for Large-Scale Ontologies Based on Mapreduce Paradigm, IEEE Trans. on cybernetics, vol. 45, no. 1, january 2015

[18] Hadoop [online], Available: https://hadoop.apache.org/

[19] Linked Open Vocabulary [online], Available: http://lov.okfn.org/dataset/lov/vocabs/

[20] HBase [online], Available: https://hbase.apache.org/