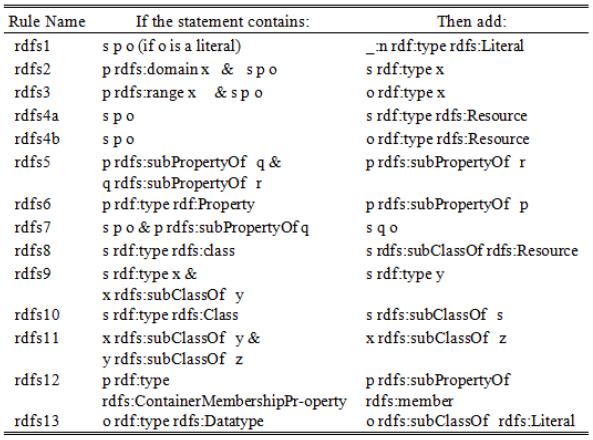
MapReduce (Dean and Ghemawat 2008) is a programming model for parallel and distributed processing of large datasets. Hadoop (Hadoop 2014) is a Java implementation of MapReduce, which supports data-intensive distributed applications on clusters of commodity machines. We present a novel representation method, Transfer Reasoning Tree (TRT) and Underived Assertional Triples (UAT), to store the incremental RDF triples more efficiently and simplify the reasoning process so that a user's online query can be answered timely. The update of TRT/UAT needs only minimum computation since the relationship between new triples and existing ones is fully exploited. Moreover, the inconsistency of ontologies can be detected and recovered effectively, thus avoiding errors in the following reasoning process.

To evaluate our methods, we have implemented a system using Hadoop and Amazon Cloud. Since Amazon Elastic Compute Cloud (EC2) can provide as much computation and storage capacity as needed in a pay-as-you-go manner, we deployed our system on EC2 to provide scalable computation. The experiments performed on BTC (Billion Triples Challenge) data show that IDRM outperforms other related ones. A real-world application on healthcare domain is also presented to validate the effectiveness of our method.

Background

The fundamental unit of RDF is a triple <*subject, predicate, object*>. RDF Schema (RDFS) (RDF-Schema 2014) is a set of classes with certain properties in RDF. RDF closure can be computed by applying all RDFS rules (RDF-Semantics 2004) iteratively on the input ontology until no new statements are derived. Fig. 1 shows 13 RDFS rules.



Rule Name	If the statement contains:	Then add:
rdfs1	s p o (if o is a literal)	_:n rdf:type rdfs:Literal
rdfs2	p rdfs:domain x & s p o	s rdf:type x
rdfs3	p rdfs:range x & s p o	o rdf:type x
rdfs4a	s p o	s rdf:type rdfs:Resource
rdfs4b	s p o	o rdf:type rdfs:Resource
rdfs5	p rdfs:subPropertyOf q & q rdfs:subPropertyOf r	p rdfs:subPropertyOf r
rdfs6	p rdf:type rdf:Property	p rdfs:subPropertyOf p
rdfs7	s p o & p rdfs:subPropertyOf q	s q o
rdfs8	s rdf:type rdfs:class	s rdfs:subClassOf rdfs:Resource
rdfs9	s rdf:type x & x rdfs:subClassOf y	s rdf:type y
rdfs10	s rdf:type rdfs:Class	s rdfs:subClassOf s
rdfs11	x rdfs:subClassOf y & y rdfs:subClassOf z	x rdfs:subClassOf z
rdfs12	p rdf:type rdfs:ContainerMembershipPr-operty	p rdfs:subPropertyOf rdfs:member
rdfs13	o rdf:type rdfs:Datatype	o rdfs:subClassOf rdfs:Literal

Fig. 1. RDFS Rules.

In order to distinguish the triples that may trigger the inference on RDFS rules, we divide them into ontological and assertional triples.

Definition 1. **Ontological Triples** are the triples from which significant inferences can be derived, i.e., the triples with predicate rdfs:domain, rdfs:range, rdfs:subClassOf, rdfs:subPropertyOf, and those with predicate rdf:type and object rdfs:Datatype, rdfs:Class or rdfs:ContainerMembershipProperty.

Definition 2. **Assertional Triples** are the triples that are not ontological triples.

Incremental Reasoning on RDF Datasets

This section presents the Incremental and Distributed Reasoning Method (IDRM) on big RDF datasets. The main steps of our proposed method are described in Fig. 2. The input of the system is incremental RDF data files. As our knowledge increases, new RDF data continuously arrive as commonly seen in practice. Then the Dictionary Encoding unit encodes the input triples into a unique and small identifier to reduce the physical size of input data and speed up the reasoning process. The encoded triples are then separated into the incremental ontological triples and incremental assertional triples, based on which the TRT/UAT Construction unit generates TRT and UAT, or the TRT/UAT Update unit updates relative TRT/UAT if this is not the first time that we run the system. The created or updated triples are stored in TRT and UAT storages, then the Query Processing unit reasons over the TRT/UAT and responses users' queries.

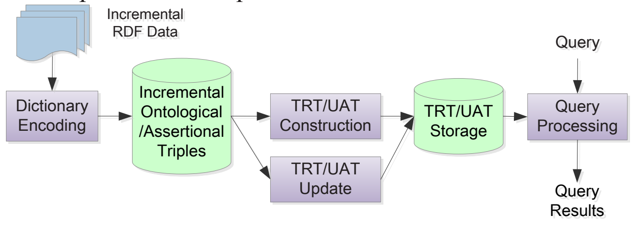


Fig. 2. Main steps of IDRM.

Transfer Reasoning Tree

As seen in Fig. 1, some key elements are more easily to trigger other inferences, and the triples related to these elements have strong correlation with each other. In order to use a more efficient method to store these triples and minimize the changes to the entire ontology base at every update, we construct them to transfer reasoning tree.

Definition 3. **Transfer Reasoning Tree (TRT)** is a set of directed trees constructed by the triples whose predicates are rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain and rdfs:range.

It is further divided into PTRT, CTRT and DRTT.

Definition 4. **Property Transfer Reasoning Tree (PTRT)** is a set of directed trees constructed based on all the triples that have predicate rdfs:subPropertyOf, or have predicate rdf:type and object rdfs:Container-MembershipProperty. Each node in a tree stands for a subject or object, and the directed link between them presents their sub-property relation.

Definition 5. **Class Transfer Reasoning Tree (CTRT)** is a set of directed trees 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 presents their sub-class relation.

Definition 6. **Domain/Range Transfer Tree (DRTT)** is a set of directed trees 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 presents the domain or range relation between the node pair.

The construction of PTRT contains 2 steps:

1) Abstract the triples whose predicates are rdfs:subPropertyOf; for each triple, build a directed graph from nodes A to B, in which A stands for its subject and B stands for its object.

2) Abstract the triples whose predicate is rdf:type and object is rdfs:Container-MembershipProperty; following the 12th RDFS rule, build a directed graph from nodes A to B, in which A is its subject and B is rdfs:member.

Similarly, the construction of CTRT is as follows:

1) Abstract the triples whose predicates are rdfs:subClassOf; for each triple, build a directed graph from nodes A to B, in which A stands for its subject and B stands for its object.

2) Abstract the triples whose predicate is rdf:type and object is rdfs:class; following the 8th RDFS rule, build a directed graph from nodes A to B, in which A is its subject and B is rdfs:Resource.

3) Abstract the triples whose predicate is rdf:type and object is rdfs:Datatype; following the 13th RDFS rule, build a directed graph from nodes A to B, in which A is its subject and B is rdfs:Literal.

The construction of DRTT has 2 steps:

1) Abstract the triples whose predicate is rdfs:range; for each triple, build a directed graph from nodes A to B, in which A stands for its subject and B stands for its object.

2) Abstract the triples whose predicate is rdfs:domain; for each triple, build a directed graph from nodes A to B, in which A stands for its subject and B stands for its object. The arrow connecting A and B is dotted line.

An example of building PTRT and DRTT is shown in Fig. 3 and 4, respectively.

The forward and reverse paths are then defined for reasoning over TRT. In each tree, the **Forward Path of node *n* or edge *r*** is a route starting from *n* or *r* to an endpoint following the sequence of the directed links. An endpoint means a node that has no links starting from it. E.g., in Fig. 3, the forward path of edge p5→p6 in PTRT is: p6→p8; while the forward path of node p5 is: p5→p7 and p5→p6→p8.

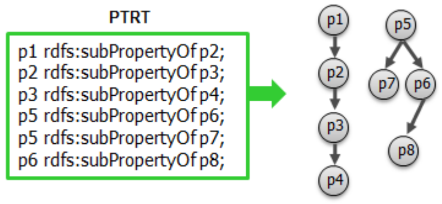

PTRT
p1 rdfs:subPropertyOf p2;
p2 rdfs:subPropertyOf p3;
p3 rdfs:subPropertyOf p4;
p5 rdfs:subPropertyOf p6;
p5 rdfs:subPropertyOf p7;
p6 rdfs:subPropertyOf p8;
p1
p2
p3
p4
p5
p7
p6
p8

Fig. 3. PTRT construction.

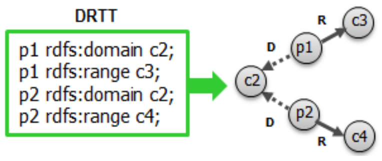


Fig. 4. DRTT construction.

In each tree, the **Reverse Path of node *n* or edge *r*** is a route starting from *n* or *r* to an endpoint following the reverse sequence of the directed links. In Fig. 3, the reverse path of edge p2→p3 in PTRT is: p1→ p2; while the reverse path of node p7 is: p7→p5.

Given an assertional triple <*s*,*p*,*o*> and TRT, Algorithms 1-3 are given for reasoning over PTRT, DRTT and CTRT.

```
Algorithm 1. Reasoning over PTRT
1: For each node n in PTRT,
2:     N ← forward path of n
3:     For each node m in N,
4:         Add triple <s, m, o> to R
5: Return R
```

```
Algorithm 2. Reasoning over DRTT
1: For each node n in DRTT,
2:     If n has a domain edge linked to node m
3:         Add triple <s, rdf:type, m> to R
4:     If n has a range edge linked to node m
5:         Add triple <o, rdf:type, m> to R
6: Return R
```

```
Algorithm 3. Reasoning over CTRT
1: For each node n in CTRT,
2:     N ← forward path of n
3:     For each node m in N,
4:         Add triple <s, rdf:type, m> to R
5: Return R
```

Because Algorithm 2 may generate new triples with predicate rdf:type, which may influence the reasoning of CTRT, the three algorithms should be run from 1 to 2 to 3. Fig. 5 shows an example of PTRT reasoning.

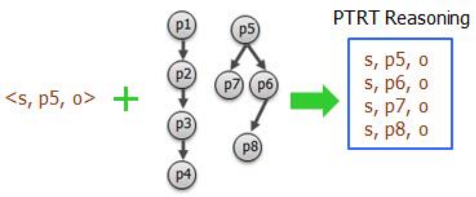


Fig. 5. Reasoning of PTRT.

Underived Assertional Triple

Since some assertional triples can be derived by the others, not all of them need to be stored.

Definition 7. Underived Assertional Triple (UAT) refers to the assertional triples that cannot be derived from others. UAT is separated into PUAT and CUAT according to the type of TRT.

Definition 8. Property Underived Assertional Triple (PUAT): Given PTRT and a set of assertional triples with the same subject and object but different predicates {<s_i, p_1, o_i>,< s_i, p_2, o_i>……}, if triple <s_i, p_j, o_i> cannot be derived from any others, we name it a PUAT for subject-object pair <s_i, o_i>.

Definition 9. Class Underived Assertional Triple (CUAT): Given CTRT and a set of assertional triples with the same subject and predicate rdf:type {<s_i, *rdf:type*, o_1>, <s_i, *rdf:type*, o_2>……}, if triple <s_i, *rdf:type*, o_j> cannot be derived from any others, we name it a CUAT for subject s_i.

To compute PUAT and CUAT, Algorithms 4 and 5 are proposed as follows.

```
Algorithm 4. Calculating PUAT
Input: PTRT, and a set of triples that have the same
 subject and object $T$={<$s_i$, $p_1$, $o_i$>,<$s_i$, $p_2$, $o_i$>……}
Output: PUAT={$s_i$, $o_i$,<$p_i$>}
1: $P$={$p_1$, $p_2$ ...}←all the predicates in $T$
2: For each $p_j$ in $P$,
3:     $Q$ ← forward path of $p_j$ in PTRT
4:     For each $q_i$ in $Q$,
5:         If $q_i$ exists in $P$
6:             Remove $q_i$ from $P$
7: Return PUAT={$s_i$, $o_i$, $P$}
```

```
Algorithm 5. Calculating CUAT
Input: CTRT, and a set of triples that have the same
 subject and the predicate is rdf:type,
$T$={<$s_i$, rdf: type, $o_1$>,<$s_i$, rdf: type, $o_2$>……}
Output: CUAT={$s_i$, <$o_i$>}
1: $O$={$o_1$,$o_2$ ...}←all the objects in $T$
2: For each $o_j$ in O,
3:     Q ← forward path of $o_j$ in CTRT
4:     For each $c_i$ in Q,
5:         If $c_i$ exists in O
6:             Remove $c_i$ from O
7: Return CUAT={$s_i$, $O$}
```

Incremental Update of TRT and UAT

The advantages of constructing TRT/UAT focus on two aspects, one is to reduce the storage as we only store the core and minimum information that cannot be derived, the other and more important one is to provide an efficient way for updating the knowledge base since updating TRT/UAT takes much fewer efforts than changing the entire ontology and re-computing RDF closure.

When new RDF files arrive, new edges are added to the existing TRT. Basically, there are two kinds of edges: **Existing Edges** refer to the triples that exist in the original TRT, and **Incremental Edges** refer to those whose subject or object or both do not exist.

The process of updating PTRT is shown in Fig. 6. The update of CTRT and DRTT is similar to that of PTRT. The process of updating UAT is presented as below.

(1) Generate new PTRT by adding incremental edges to the existing PTRT. (2) Generate incremental PUAT based on the incremental ontological triples, add the incremental PUAT to existing PUAT, and execute Algorithm 4 to obtain new PUAT. (3) Generate incremental DRTT based on the incremental ontological triples and add incremental DRTT to the existing DRTT. (4) For the PUAT with a predicate in the reverse path of the incremental edges while the forward path of the incremental edge contains nodes in DRTT, generate the assertional triples by executing Algorithm 2. (5) Generate new CTRT by adding incremental edges to the existing CTRT. (6) Generate the incremental CUAT based on the incremental assertional triples and the triples generated in Step 4, add the incremental CUAT to the existing CUAT, and execute Algorithm 5 to compute new CUAT.

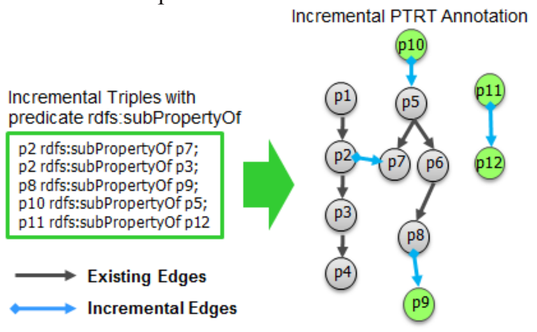

Incremental PTRT Annotation
Incremental Triples with predicate rdfs:subPropertyOf
p2 rdfs:subPropertyOf p7;
p2 rdfs:subPropertyOf p3;
p8 rdfs:subPropertyOf p9;
p10 rdfs:subPropertyOf p5;
p11 rdfs:subPropertyOf p12
Existing Edges
Incremental Edges
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10
p11
p12

Fig. 6. Update of PTRT.

Query Strategy

In this section, 6 operators and 8 searching strategies are defined to support the query based on TRT and UAT.

Operator 1: Given subject-object pair <*s, o*> to search a predicate list. First, search <*s, o*> in PUAT to obtain its predicate list, then for each *p* in the predicate list, output the nodes in the forward path of *p* in PTRT.

Operator 2: Given the subject *s* and the predicate rdf:type, to search an object list. First, search *s* in CUAT to obtain its object list, then for each *c* in the object list, output the nodes in the forward path of *c* in CTRT.

Operator 3: Given predicate *p* to search subject-object pairs. In PTRT, for each node *q* in the reverse path of node *p* including *p* itself, output the subject-object pairs that have predicate *q* in PUAT.

Operator 4: Given predicate rdf:type and object *c* to search subjects. In CTRT, for each node *d* in the reverse path of node *c* including *c* itself, output the subjects that have object *d* in CUAT.

Operator 5: Given subject *s* and predicate rdfs:subPropertyOf, to search an object list. If *s* exists in PTRT, output the nodes in the forward path of *s* in PTRT.

Operator 6: Given subject *s* and predicate rdfs:subClassOf, to search an object list. If *s* exists in CTRT, output the nodes in the forward path of *s* in CTRT.

Here we take SPARQL query language as an example, to introduce the searching strategy for 8 basic query statements. More complicated queries can be decomposed into basic query types.

1) <*?x ?y ?z*>: For each subject-object pair in PUAT, run Operator 1, and for each subject in CUAT, run Operator 2.

2) <*?x p ?z*>: search for all the subject-object pairs that have the specified predicate *p*. If *p* is in PTRT, run Operator 3; if *p* is rdf:type, run Operator 2; if *p* is rdfs:subPropertyOf, run Operator 5; if *p* is rdfs:subClassOf, then run Operator 6.

3) <*s ?y ?z*>: search all the predicate-object pairs with specified subject *s*. First, obtain all the subject-object pairs in PUAT with subject *s*, and then for each returned subject-object pair <*s, o*>, search <*s ?p o*> according to the 7th query type. If *s* exists in CUAT, obtain the object list corresponding to *s* in CUAT, and for each object in the object list, run Operator 2 and record the results as set O, and output <*s rdf:type O*>.

4) <*?x ?y o*>: search all the subject-predicate pairs with specified object *o*. First, obtain all the subject-object pairs in PUAT with *o*, and then for each returned subject-object pair <*s, o*>, search <*s ?p o*> according to the 7th query type. If *o* exists in CTRT, run Operator 4 to obtain a subject list S, and output <*S rdf:type o*>.

5) <*?x p o*>: search <*?x p ?z*> as the 2nd query type, and then filter the results by object *o*.

6) <*s p ?z*>: search <*?x p ?z*> as the 2nd query type, and then filter the results by subject *s*.

7) <*s ?y o*>: search all the predicates for the given subject *s* and object *o*. If <*s, o*> exists in PUAT, then run Operator 1. If *o* exists in CTRT, then run Operator 4 to check whether *s* is in the result. If so, then the predicate rdf:type should also be included in the results.

8) <*s p o*>: search <*?x p ?z*> according to the 2nd query type, and then filter the results by subject *s* and object *o*. If no results are left, return empty to the user.

Inconsistency Detection and Recovery

An additional benefit of our proposed method is to detect and recover the inconsistency of the ontologies. Based on the tree structure of TRT, it is convenient to discover inconsistencies by following the strategies as below.

1) In PTRT, if a closed loop including more than 2 nodes is detected according to the directions of the links between nodes, a PTRT inconsistency occurs. To resolve it, the oldest triple in this circle is deleted by default. Each triple has a timestamp, and generally newly-arrived triples are more confident than old ones. However, users can also decide which one to be deleted manually.

E.g., in Fig. 7, a new triple <*p4 rdfs:subPropertyOf p1*> is added to the ontology base, then p1→p2→p3→p4→p1 forms a closed loop, which is unreasonable in practice. Therefore one triple in this loop is deleted to recover the consistency of PTRT. In CTRT, the method for dealing with inconsistencies is similar to that in PTRT.

2) In DRTT, when a new triple indicates a different domain/range for an existing predicate, we detect whether the two have sub-class relation in CTRT. If yes, the one who is sub-class is retained. If not, the new one is retained by default. However the users can also configure the system to retain the old domain/range. In Fig. 7, the new triple <*p1 rdfs:domain c6*> indicates a new domain *c6* for *p1*, so we delete the old edge from *p1* to *c2*.

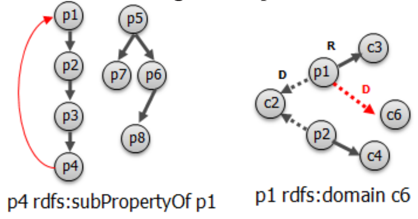


Fig. 7. Inconsistency in PTRT (left) and DRTT (right)

Notably, if the ontology has certainty degree for each triple, which indicates the probability that the triple is correct, we can delete the one with lowest certainty degree in the closed loop in PTRT/CTRT, and the domain/range edge with lower certainty degree in DRTT as well.

System Implementation

We have implemented a system on Hadoop platform and Amazon Cloud. As seen in Fig. 8, the core of the system is the IDRM units, which receive the input incremental RDF datasets, perform the reasoning by a set of MapReduce programs, detect and recover ontology inconsistencies, interact with HBase and return the query results to end-users. HBase (HBase 2014) is a distributed and scalable data store for Hadoop. We have designed 6 HBase tables to store the encoded ID, PTRT, CTRT, DRTT, PUAT and CUAT. We use Hadoop 1.2.1 and HBase 0.94.12, which are built on 1-16 m1.medium EC2 instances, each with 2 EC2 Compute Units, 3.75GB memory and 410GB storage.

BTC (BTC 2012) is a public dataset crawled from the Web. It contains 5 large datasets as seen in Table 1. To show the performance of our method, we compare IDRM with WebPIE (Urbani et al. 2012) on BTC dataset. We use WebPIE to generate the RDF closure and then search the related triples as the output of the query.

Each method is executed three times on each dataset to calculate the number of the output triples and the average processing time for reasoning (see Table 2). For IDRM, the output triples are the ones in TRT/UAT. For WebPIE, the output triples are the ones in RDF closure. We can see that the reasoning time for IDRM is less than WebPIE (70% of WebPIE in total time) and the output triples for IDRM is much fewer than WebPIE (61.9% of WebPIE).

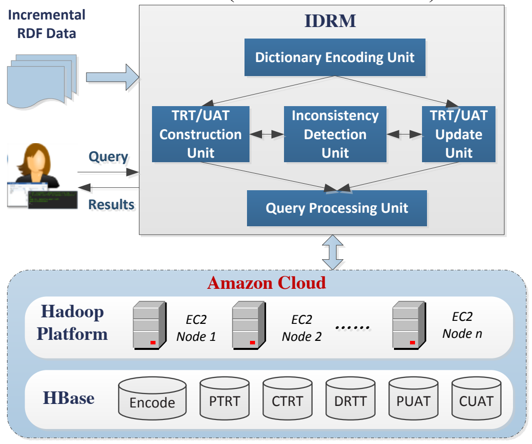


Fig. 8. System architecture.

Table 1. Basic Information of BTC Dataset

Dataset	Space (GB)	# of triples	Sub-Class	Sub-Property	Domain & Range
Datahub	215.8	910078982	26146	15068	36338
DBpedia	48.2	198090024	275	0	1136
Freebase	38.9	101241556	0	0	1
Rest	17.8	22328242	30373	746	2905
Timbl	4.7	204806751	291095	24431	55086
Overall	325.4	1436545555	347889	40245	95466

Table 2. Result for the Reasoning (16 Nodes)

Dataset	No. of Triples in TRT/UAT	Time of IDRM(min)	No. of Triples in RDF closure	Time of WebPIE(min)
Datahub	713574291	26.3	1079343655	43.1
DBpedia	133242743	13.4	198091689	17.6
Freebase	94134030	7.2	101241556	9.1
Rest	17073633	6.5	26287842	8.3
Timbl	114130464	15.2	326688386	19.8
Overall	1072155161	68.6	1731653128	97.9

To validate the scalable performance, we take the Datahub dataset as an example and report the reasoning time when the number of EC2 nodes increases from 1 to 16 (see Fig. 9). When more nodes are deployed, the execution time continues dropping. The cost of each medium instance is $0.078/hour, and the total cost varies along with the increase of nodes.

To further compare the performance when the input data are incremental, the whole dataset is divided into 4 parts and input to the system gradually. As seen in Table 3, the reasoning time of IDRM drastically reduced in comparison with WebPIE as input size grows. Then we input 50 queries to Datahub dataset, and the average response time is 53ms for IDRM and 57ms for WebPIE.

Application Scenario

Then we collaborate with a Chinese hospital to validate our method in real-world healthcare data. For confidentiality reasons, we cannot reveal the name of the hospital. Our target is to assist the retrieval of Electronic Medical Record (EMR) in the hospital's information system. An EMR is a digital version of a patient's medical records including all the medical history, medication and allergies, and personal statistics like age and weight.

We first build a medical ontology based on 1.5 million EMRs, using an ontology learning tool. 0.59 billion triples are learned and added to the ontology. New EMRs are generated and new RDF triples are added to the ontology base periodically. Then our IDRM method is performed on a Hadoop cluster with 16 computing nodes for parallel computating. The queries from doctors and nurses with respect to patients, illnesses and drugs are then executed to assist their diagnosis and treatment.

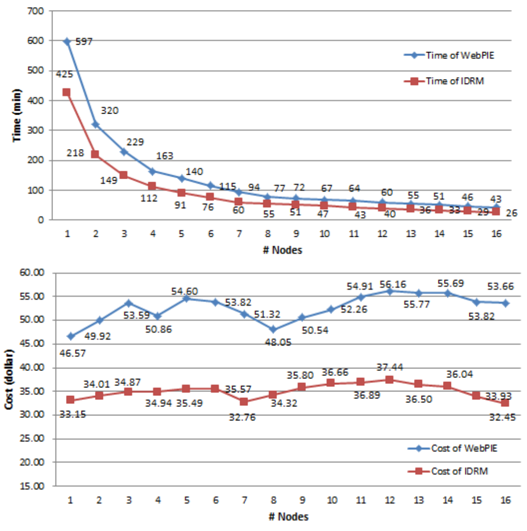


Fig. 9. Processing time and cost on different nodes.

Table 3. Update Time with Incremental Input (16 nodes)

Input Size (billions of triples)	0.1	0.5	1	1.44
Time of IDRM (min)	7.4	21.3	25.7	24.2
Time of WebPIE(min)	8.1	30.2	61.8	96.5

Conclusions and Future Work

This work proposes an incremental and distributed reasoning method to deal with large-scale ontologies. The construction of TRT/UAT significantly reduces the re-computation time as well as the storage. Users can execute online queries and detect ontology inconsistency effectively. A system is implemented using Hadoop on Amazon Cloud. The application in healthcare validates the feasibility of our method. In future, we will validate IDRM on more datasets and extend it to other ontology languages.

References

BTC 2012. Billion Triples Challenge 2012 Dataset. from http://km.aifb.kit.edu/projects/btc-2012/.

Dean, J. and Ghemawat, S. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51(1): 107-113.

Fokoue, A., et al. 2012. Querying Linked Ontological Data Through Distributed Summarization. Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence: 31-37.

Guo, J., et al. 2012. Semantic Inference on Heterogeneous E-Marketplace Activities. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 42(2): 316-330.

Guttmann, C., et al. 2013. On the challenges of balancing privacy and utility of open health data. Joint Proceedings of the Workshop on AI Problems and Approaches for Intelligent Environments and Workshop on Semantic Cities: 43-47.

Hadoop 2014. Hadoop. from http://hadoop.apache.org/.

HBase 2014. HBase. from http://hbase.apache.org/.

Lecue, F. 2012. Diagnosing Changes in An Ontology Stream: A DL Reasoning Approach. Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence: 80-86.

McCreadie, et al. 2009. On Single-Pass Indexing with MapReduce. SIGIR Forum: 742-743.

RDF-Schema 2014. RDF Schema. from http://en.wikipedia.org/wiki/RDFS.

RDF-Semantics 2004. RDF Semantics. from http://www.w3.org/TR/rdf-mt/.

SPARQL 2013. SPARQL 1.1 Overview. from http://www.w3.org/TR/sparql11-overview/.

Urbani, J., et al. 2012. WebPIE: A Web-scale Parallel Inference Engine using MapReduce. Web Semantics: Science, Services and Agents on the World Wide Web 10: 59 - 75.

W3C 2010. Linking Open Data on the Semantic Web. from http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics.

Weaver, J. and Hendler, J. 2009. Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. The Semantic Web - ISWC 2009, Lecture Notes in Computer Science 5823: 682-697.

Zhang, Y., et al. 2013. A Semantic Approach to Retrieving, Linking, and Integrating Heterogeneous Geospatial Data. Joint Proceedings of the Workshop on AI Problems and Approaches for Intelligent Environments and Workshop on Semantic Cities: 31-37.