Distributed Assumption-Based Truth Maintenance System for Scalable Reasoning

대용량 추론을 위한 분산환경에서의 가정기반진리관리시스템

저자
(Authors)

Batselem Jagvaral, Young-Tack Park

출처
(Source)

정보과학회논문지 43(10), 2016.10, 1115-1123 (9 pages)
Journal of KIISE 43(10), 2016.10, 1115-1123 (9 pages)

발행처
(Publisher)

한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL

http://www.dbpia.co.kr/Article/NODE07021312

APA Style

Batselem Jagvaral, Young-Tack Park (2016). Distributed Assumption-Based Truth Maintenance System for Scalable Reasoning. 정보과학회논문지, 43(10), 1115-1123.

ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 43, No. 10, pp. 1115-1123, 2016. 10
http://dx.doi.org/10.5626/JOK.2016.43.10.1115

대용량 추론을 위한 분산환경에서의 가정기반진리관리시스템

(Distributed Assumption-Based Truth Maintenance System for Scalable Reasoning)

바 트 셀 렘 [†]　　박 영 택 [††]
(Batselem Jagvaral)　　(Young-Tack Park)

요 약 가정기반진리관리 시스템(ATMS)은 추론 시스템의 추론 과정을 저장하고 비단조추론을 지원할 수 있는 도구이다 또한 의존기반 backtracking을 지원하므로 매우 넓은 공간 탐색 문제를 해결 할 수 있는 강력한 도구이다. 모든 추론 과정을 기록하고, 특정한 컨텍스트에서 지능형시스템의 Belief를 매우 빠르게 확인하고 비단조 추론 문제에 대한 해결책을 효율적으로 제공할 수 있게 한다. 그러나 최근 데이터의 양이 방대해지면서 기존의 단일 머신을 사용하는 경우 문제 해결 프로그램의 대용량의 추론과정을 저장하는 것이 불가능하게 되었다. 대용량 데이터에 대한 문제 해결 과정을 기록하는 것은 많은 연산과 메모리 오버헤드를 야기한다. 이러한 단점을 극복하기 위해 본 논문에서는 Apache Spark 환경에서 functional 및 객체지향 방식 기반의 점진적 컨텍스트 추론을 유지할 수 있는 방법을 제안한다.. 이는 가정(Assumption)과 유도과정을 분산 환경에 저장하며, 실체화된 대용량 데이터셋의 변화를 효율적으로 수정 가능하게 한다. 또한 ATMS의 Label, Environment를 분산 처리하여 대규모의 추론 과정을 효과적으로 관리할 수 있는 방안을 제시하고 있다. 제안하는 시스템의 성능을 측정하기 위해 5개의 노드로 구성된 클러스터에서 LUBM 데이터셋에 대한 OWL/RDFS 추론을 수행하고, 데이터의 추가, 설명, 제거에 대한 실험을 수행하였다. LUBM2000에 대하여 추론을 수행한 결과 80GB데이터가 추론되었고, ATMS에 적용하여 추가, 설명, 제거에 대하여 수초 내에 처리하는 성능을 보였다.

키워드: 분산 가정기반진리관리시스템, 추론엔진, 컴퓨터 클러스터, 온톨로지 추론, Spark

Abstract Assumption-based truth maintenance system (ATMS) is a tool that maintains the reasoning process of inference engine. It also supports non-monotonic reasoning based on dependency-directed backtracking. Bookkeeping all the reasoning processes allows it to quickly check and retract beliefs and efficiently provide solutions for problems with large search space. However, the amount of data has been exponentially grown recently, making it impossible to use a single machine for solving large-scale problems. The maintaining process for solving such problems can lead to high computation cost due to large memory overhead. To overcome this drawback, this paper presents an approach towards incrementally maintaining the reasoning process of inference engine on cluster using Spark. It maintains data dependencies such as assumption, label, environment and justification on a cluster

· 본 연구는 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No. R0101-16-0054, WiseKB: 빅데이터 이해 기반 자가학습형 지식베이스 및 추론 기술 개발)

† 학생회원 : 숭실대학교 컴퓨터학부
selmee006@gmail.com
†† 종신회원 : 숭실대학교 컴퓨터학부 교수(Soongsil Univ.)
park@ssu.ac.kr
(Corresponding author임)

논문접수 : 2016년 6월 1일
(Received 1 June 2016)
논문수정 : 2016년 7월 15일
(Revised 15 July 2016)
심사완료 : 2016년 7월 18일
(Accepted 18 July 2016)

정보과학회논문지 제43권 제10호(2016. 10)

of machines in parallel and efficiently updates changes in a large amount of inferred datasets. We deployed the proposed ATMS on a cluster with 5 machines, conducted OWL/RDFS reasoning over University benchmark data (LUBM) and evaluated our system in terms of its performance and functionalities such as assertion, explanation and retraction. In our experiments, the proposed system performed the operations in a reasonably short period of time for over 80GB inferred LUBM2000 dataset.

Keywords: distributed ATMS, inference engine, cluster computing, ontology reasoning, spark

1. Introduction

Reasoning systems often generate computational models of different contexts and require bookkeeping process to keep these models consistent with new information and changes in the knowledge base (KB). Many reasoning systems in different domains such as medical diagnosis and electronic circuit diagnosis, employ assumption based truth maintenance system [1,2] (ATMS) to solve complex problems and keep track of inferences (beliefs) or assumptions. ATMS can provide non-monotonic maintenance mechanism in which assumptions and beliefs are retracted and modified incrementally [5,10]. It can also answer queries about whether a given hypothesis is true and explain how it has been derived. However, to reduce the search space for a given problem, ATMS tends to generate multiple assumptions for the contexts (solutions) of the goal such that the same assumption can hold in different contexts simultaneously. As a result, the number of assumptions can grow exponentially in the ATMS network. ATMS techniques therefore have limited capabilities for large-scale knowledge based systems.

In a large-scale system such as a semantic web where a massive amount of knowledge data is constructed, observed contexts need to be altered and retracted continuously to maintain the consistency in the KB. Such reasoning systems [11,12] require ATMS like truth maintenance system, to search solutions efficiently and maintain the coherence of the KB but recently the amount of the semantically annotated data has been growing at an unprecedented rate which makes it difficult to apply ATMS to semantic web applications.

Meanwhile, cluster computing frameworks, Spark and MapReduce have become dominant in processing large data sets and made it attractive to utilize distributed processing for both reasoning and truth maintenance. A number of studies have been conducted on ontology reasoning for semantic web based on MapReduce framework [12] but these studies only focus on reasoning a large amount of ontology data and scalable incremental reasoning methods and do not consider the explanation model and non-monotonic reason maintenance.

Due to these issues, this paper presents an approach towards deploying ATMS on distributed in-memory framework. We propose a distributed ATMS system called D-ATMS and discuss how to efficiently maintain a large amount of inferences and provide explanations. In addition, the proposed D-ATMS is a domain-independent truth maintaining system for large-scale reasoning systems that can scale linearly. We distribute assumptions and inferences across multiple machines where each machine is partially responsible for the maintenance process. To the extent of our knowledge, this is the first study to show how to parallelize ATMS on a large cluster.

To evaluate the performance of our system, we consider OWL Horst and RDFS fragments that extensively enrich the knowledge base. We conduct several experiments to measure the capability of the system over a large amount of data and assess the methods we developed for distributing ATMS. The experimental results show that our system is capable of efficiently maintaining a large amount of materialized ontology data up to 100 million triples on a cluster equipped with 5 machines and can perform fast updates over such large data in a few seconds.

Our major contributions are as follows: (1) We introduce an immutable data layout for parallelizing a massive amount of data dependencies of ATMS using indexed resilient distributed datasets on Spark framework that are efficiently updatable. (2) Moreover, we provide an efficient algorithm for updating labels over large materialized data. (3) Finally, we give a performance evaluation to demonstrate the scalability of our system and effectiveness of our label updating method.

2. Related Work

We adopt the notion of ATMS [1] as modified for distributed systems by [3], using propagation algorithm of [8]. Our work differs from [2] in our advanced dependency indexing method which deals with the large amount of inferences as described by [3,4].

[2] extends ATMS to trace inferential dependencies between axiom formulas and compute the minimal sets of axioms responsible for a contradiction. This paper extends the ATMS to deal with disjunctions and explains their approach very well but it is based on the traditional ATMS system and capable of dealing with a small amount of inferences, not designed for large-scale applications.

[5] proposes a massively parallel ATMS system and show how the ATMS can be implemented on parallel hardware. It performs orders of magnitude faster than the state-of-art approaches. It introduces a connection machine algorithm for developing ATMS on a cluster of computers. In this study, each processor represents a subspace of assumption space and nodes are represented as a union of these subspaces with on bit per processor. As the number of bits grows for assumptions, it requires more processing power. Therefore, it needs exponentially many processes to construct an ATMS network. Instead, our approach assigns a partition of assumptions to a single processor and indexed them corresponding to hash codes. It encodes assumptions using binary numbers to reduce memory consumption.

ATMS is not extensively being studied because of its drawbacks such as creation of enormous assumptions. By addressing this problem, our paper claims that the ATMS can be extended to handle a large dataset and applied to the current big data processing. We consider that the ATMS is useful for systems where many solutions need to be found from the large search space. In our study, the solution space is explored in parallel, and the ATMS maintains several contexts corresponding to different assumptions while the inference engine checks for assumption consistency. Moreover, other related [1,2,7,8] works were incapable to handle large datasets; therefore, we could not compare our system to them in our experiment. We believe that this is the first study to conduct to maintain such a large amount of inferences.

3. Distributed Assumption-based TMS

3.1 Background

To set the scene for this paper, we begin with a brief overview of context inference and maintenance. Reasoning engine deduces a conclusion P based on a specific set of assumptions, which could be an inference from default reasoning or an incorrect assertion. It then might discover new evidence from a different rule, such as sentence $\neg P$ which says P is an incorrect assertion. In order to prevent from the creation of this contradiction in the KB, the incorrect sentence P needs to be retracted.

However, retracting P causes a problem if P implied another sentence Q such that $P \Rightarrow Q$ after it was inferred. To solve this contradiction problem, one can retract all the sentences derived from P but it complicates the situation if other implication such as $K \Rightarrow Q$ occurred. The implication $K \Rightarrow Q$ convinces that Q is reliable and should not be retracted. To cope with this complication, the assumption-based TMS keeps the track of reasoning process and its dependencies in the KB. In particular, it records all the assumptions, which cause the derivation of P, to the KB. When a contradiction such as $\neg P$ occurs, P is removed from the KB, the system retracts the assumptions that cause P to be true and removes those assumptions from the assumptions of other successor inferences. This procedure keeps the KB consistent and reduces the computation cost when the system deals with a large amount of inferences. Fig. 1 shows an example of maintaining course information in the KB to illustrate the process of retracting such an invalid sentence from the KB. First, reasoning engine asserts a fact such that a student enrolled in a course and then, it learns that the student

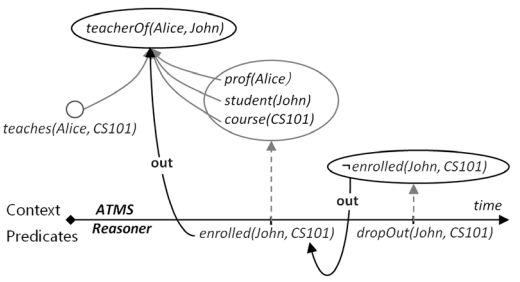

Fig. 1 Maintenance of context predicates

recently dropped out of the course. In this case, it needs to retract contradictions without removing any valid inference.

3.2 Formalisation

Reasoning engine sends such inferences, materialized rules, to the ATMS via a specific interface (Section3). Inference is mapped into a justification such that $\alpha \rightarrow \gamma$ which is a grounded atom and has the form (1), where n is the consequent node and $n_{\phi_1}, \cdots, n_{\phi_k}$ are the antecedent nodes.

$$\gamma: n_{\phi_1} \wedge \ldots \wedge n_{\phi_k} \Longrightarrow n_{\phi} \quad (1)$$

Each node n_{ϕ} consists of a set of environments $\{E_1, ..., E_k\}$ in the horn form. Environment E is defined as a set of assumptions, that is a conjunction of assumptions such that $E = A_1 \wedge \ldots \wedge A_l$. Assumptions are only generated when necessary. n_{ϕ} node holds in E if it can directly be derived from E. If n_{ϕ} has no antecedent, it is called a premise node.

$$\gamma: n_{\phi_1} \wedge \ldots \wedge n_{\phi_k} \Longrightarrow \perp \quad (2)$$

E is nogood if it is subsumed by environments in the label of contradiction node $n_{\perp}$ such that $E \subseteq E' \mid E \in \perp$. Nogoods are denoted by ∇. Contradiction nodes $n_{\perp}$ are the nodes that the inference engine indicates it cannot hold (2). Nogood is minimal if there is not a single node that subsumes the others. An environment is contradiction or nogood if the distinguished node $\perp$ holds in it $E \vdash \perp$. ATMS ensures that no node is considered to follow from a set of assumptions if a contradiction node also follows from that set of assumptions.

Furthermore, the ATMS network is incrementally updated and receives additional justifications, and various queries about the environments in which nodes hold. To facilitate answering these queries the ATMS maintains a set of environments $\{E_1, ..., E_k\}$ for each node n.

3.3 System Overview

The purpose of distributed D-ATMS is to efficiently and incrementally maintain a large amount of reasoning results while preventing the KB from inconsistency[12].

The D-ATMS receives a stream of additional justifications incrementally and given a set of justifications $\mathcal{J}$, it maps a finite set of assumptions A into P partitions of machines on the cluster and implements the label updating for each justification of $\mathcal{J}$ in parallel. Then, it computes the incremental changes caused by either adding a justification or retracting an assumption. The proposed D-ATMS system is built on a series of indexed RDDs. Indexed immutable RDD enables efficient modification and deletions in the D-ATMS. It utilizes hashed binary tree to provide fast lookups and modifications in the ATMS.

Fig. 2 shows the proposed system architecture that composed of three main components. In practice, inference engines such as OWL and RDFS make inferences and send the justifications of the inferences to

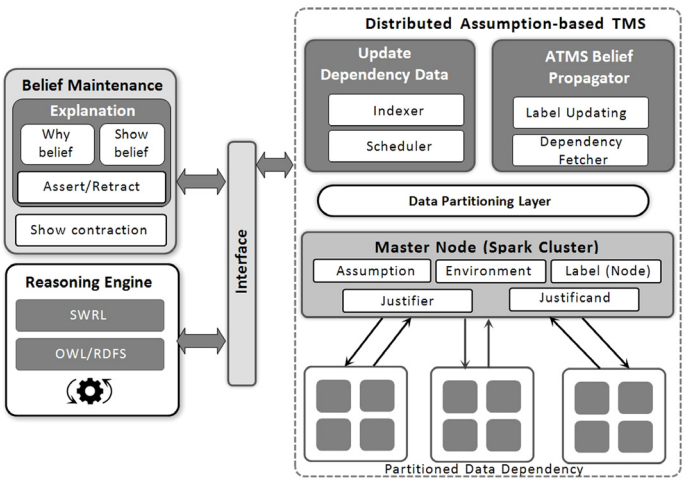

Fig. 2 System architecture design

the ATMS. Belief maintenance is an interactive interface designed for users to manipulate these inferences in ATMS, which is explained in Section 3.6. The D-ATMS component receives updates in real time from external components and incrementally re-computes labels. For example, a single update can also affect other successor nodes justified from a specific node, which require re-labeling. DATMS incorporates update dependency graph nodes which stores the inferences on cluster as partitioned data. D-ATMS scheduler propagates the changes through the network using the nodes and justifies beliefs etc. Belief maintenance subcomponent is used for users to analyze the D-ATMS network and retract/assert statements (nodes) when it is necessary. Dependency fetcher retrieves relevant dependency data described at section 2.4 to query node and also simplifies accessing data stored in cluster.

3.4 Data Abstraction

As shown in Fig. 3, believed nodes reside in different machines and connected with each other via justification. To effectively maintain inferences, we have developed a distributed data structure that supports non-monotonic label updating in the ATMS as shown in Fig. 5. The data structure is based on indexed *RDD* sets that store all dependency nodes. These dependency data sets persistently preserve the computation of unchanged labels and can be recomputed when the change occurs. To minimize data movement, we construct two *RDD* sets for environment and nodes respectively. In addition to these datasets, we create two more data sets for justifications [1,9] such as justifier and justificand (Fig. 4). It facilitates updating labels of the network and helps to focus on computing a specific set of labels rather than recomputing all the labels in the network. The proposed system reads a set of justifications in the

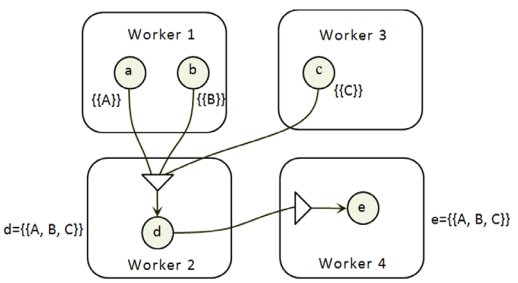

Fig. 3 An example of data parallelism

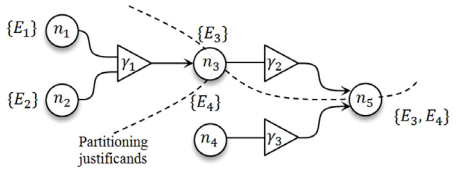

Fig. 4 An example of ATMS network

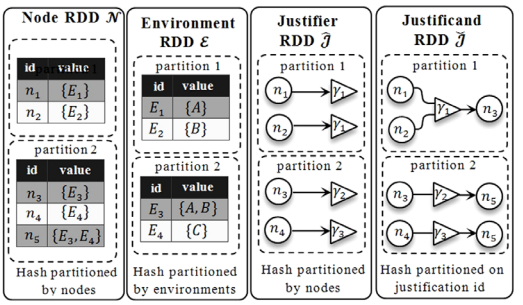

Fig. 5 Distributed representation of D-ATMS network

raw format from files and constructs those *RDD* data sets to build the ATMS network. After building it, the system can iteratively receive a constant stream of justifications and update the network. Suppose we define an ATMS network as Fig. 4 and split each data set into 2 partitions on the cluster. Here the network dependencies determine which nodes are linked together so that during the update, it pinpoints nodes and fetches only the desired nodes from the data store on the cluster since they are hash partitioned by id.

In general, the bulk of the data processing is performed in ATMS. This balanced architecture can deliver linear scalability to more than a thousand processing streams executing in parallel.

Suppose we have γ justification justifies $\gamma : b \wedge c \rightarrow f$. This justification is mapped into (α, γ) pair in justifier *RDD* set which states that a node justifies γ justification. Let the justifier *RDD* be j and j denote justificand *RDD* set. j indicates which antecedent nodes justify the consequence via what justifier. In particular, keys of pairs are hashed to index positions and values are stored in the corresponding entry in the value array. Hash collisions are resolved using probing indices. For example, pair element $\langle n_5, \{E_2, E_2\}\rangle \in \check{J}$ has two environments whose values are mapped into the environment *RDD* set denoted as ε. To immutably update values in *RDD* sets, it adopts a high-

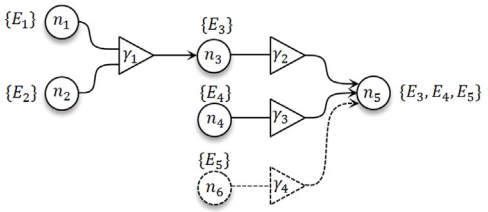

Fig. 6 Label updating

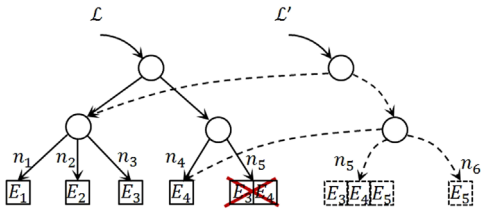

Fig. 7 Radix tree based label updating. Dashed lines indicate new label L′ that creates a pointer of the tree object without making a full copy of labels for a single modification

branching-factor tree (radix tree) which has 32 branching factors; thus, each node in the tree has a multiple cache line size. Each leaf nodes has an array with 32 elements (Fig. 7).

These datasets can support an efficient label updating such that ATMS can update and retract nodes it wants without fully copying immutable datasets and propagate changes.

Fig. 6,7 demonstrates the updating method for distributed datasets. As the reasoner pass γ_4 justification, ATMS updates the label of n_4. To update n_4, it retrieves justifications that are responsible for deriving n_4 and recomputes its label. Such partitioned tree structure makes it possible to update specific labels of nodes without massive communications. It maintains a hash index within each partition and uses immutable data structure to enable efficient modifications and deletions. Furthermore, it allows streaming updates for the ATMS network.

Overall updating takes $O(\log n)$ time to modify a single node. In addition to indexed datasets, to store a large amount of dependencies, we represent assumptions by binary code. By this way we can efficiently retrieve dependency elements and compute labels via bitwise operations (Section 3.3). For example each bit represents a unique assumption in a single environment so that an environment will be a sequence of bits.

3.5 Computing Labels

In this section, we explain how we adopt the standard label computation algorithm to distributed maintenance system. Reasoners typically assert contexts that might be retracted later. Reasoner requires this bookkeeping process to maintain the consistency of semantic contexts in real time. When a new event occurs, a context predicate is inferred along with many semantic contexts. When the event terminates or the asserted contexts change, stored previously inferred contexts may no longer be valid. But recomputing a large set of inferred data may take a long time. To solve this problem, we need dependency-aware updating [13] via ATMS. The principle behind our algorithm is to incrementally maintain a large amount of materialized semantic contexts and assert/retract contexts efficiently in real time. It propagates solely incremental changes of node labels.

The system assumes that all other node labels are correct before the introduction of the new justification and therefore only propagates the incremental changes caused by a new justification. The algorithm operates by making labels locally correct and propagating label changes until labels become globally correct in backward [1]. Let $\{E_{i\in I}\}$ be the environments of L_A indexed by I and be $\{E_{i\in J}\}$ be the environments of L_B indexed by J respectively. Then the updated label L' can be defined as follows:

$$L' = \mathcal{MAX}(\{\cup_i E_i \,|E_i \in L_{ij}\}) \quad (1)$$

where $\mathcal{Max}$ computes the subsumption that returns the minimal sufficient environments. $\mathcal{Max}$ removes nogoods and environments subsumed by others in L'. Given two labels L_2 and L_1, we state that L_2 is subsumed by that L_1 as written as $L_2 \subseteq L_1$.

$$L_2 - L_1 = \{E \in L' | s.t. E' \subseteq E\} \quad (2)$$

For every environment $E \in L'$, there exists environment $E' \in L_2$:

$$L' = \mathcal{MAX}(L_1 \cup L_2) \quad (3)$$

We can rewrite environments using bit-vectors, E = 1100 and E' = 1100. For example, E' is subsumed by E iff $E \,\&\, E'$ is equal to E.

In algorithm 1, we show pseudo code for computing labels that parses justification and passes it to the justifier which calls the function PROPAGATE-UPDATE to update labels in backward. The first

step of maintenance is to create *RDD*s for assumptions and justifiers. Labels are represented by pair indexed *RDD*s which contain the name of node and a set of environments. It propagates the changes to the nodes reachable by justifiers. When a new justification J is supplied, ATMS calls propagate function. It operates recursively until no change is made. The arguments to propagate function are a justification consisting of antecedents and consequences, and a set of environments added to the label of the node.

In Algorithm 2, we run backtracking which performs label updating for each $J \in \mathcal{J}$. Removing ∇ requires the system to access all nodes in the network. Therefore, to speed up the nogood removal process, we perform $\mathcal{Max}$ in each partition simultaneously. Fortunately, Spark framework transformation allows us to transform partitions in parallel on the cluster (line 4 of algorithm 2). Nogood environments are serialized and broadcasted to each cluster and then invoked to the partitions by GET-LOCAL-DATA function. Broadcasting can effectively reduce the communication cost when we have a small data set to be matched to a large data set. Here we assume that ∇ is small enough to be copied and cached on each machine.

The label of a node contains all the solutions to the problem solving task. As problem solving proceeds, more nogoods are discovered and these eliminate many of the environments in nodes. The ATMS then determines which combinations of choices are consistent and which conclusions they lead to. On the basis of these results, the problem solver explores additional consequences of those conclusions, possibly introducing new choices. This cycle repeats until a set of choices is found to satisfy the goal or all combinations are proven contradictory.

3.6 Belief Maintenance and Explanation

When dealing with a large number of inferences, incremental reasoning tasks and reasoning explanation become complex. Belief maintenance subcomponent facilitates this complexity by providing an interface for users to interact with the ATMS and explore beliefs and explanation models of the beliefs [6]. For example, when it detects nogood, we can revise it and update changes.

The size of the node's label and the number of set

Table 1 Algorithms for distributed ATMS label updating

Algorithm 1 UPDATE-LABEL(J)
1 $\langle C, A\rangle \leftarrow \mathcal{J}$.get($J$), $L \leftarrow \mathcal{N}$.get(C), $\tilde{L}_T \leftarrow \tilde{L}$, $\hat{L} \leftarrow \emptyset$
2 **for each** $n \in body$ **do**
3 $\quad \hat{L} \leftarrow \hat{L} \cup \mathcal{N}$.get($n$)
4 **if** $\tilde{L} \neq \emptyset$ **then**
5 $\quad \tilde{L} \leftarrow \mathcal{MAX}(\{E_i \cup E_j | E_i \in L; E_j \in \hat{L}\})$
6 $\quad$ **if** $\tilde{L} \neq \tilde{L}_T$ **then** PROPAGATE-UPDATE($N, \tilde{L}$)
7 **else** PROPAGATE-UPDATE ($N, \hat{L}$)

Algorithm 2 PROPAGATE-UPDATE(n, L)
1 **if** $n = \perp$ **then**
2 $\quad L_{bc} \leftarrow$ BROADCAST(L)
3 $\quad \nabla \leftarrow \nabla \cup L$
4 $\quad$ **for each partition** $\mathcal{L}_p \in \mathcal{N}$.labels in parallel **do**
5 $\quad\quad L_{loc} \leftarrow$ GET-LOCAL-DATA(L_{bc})
6 $\quad\quad$ **for each** $\tilde{L} \in \mathcal{L}_p$ **do**
7 $\quad\quad\quad$ **if** $E \subseteq \tilde{E}$ for some $E \in L_{loc}$ and $\tilde{E} \in \tilde{L}$ **then**
8 $\quad\quad\quad\quad \tilde{L} \leftarrow \tilde{L} - \tilde{E}$
9 **else**
10 $\quad \mathcal{J} \leftarrow \hat{\mathcal{J}}$.get($n$)
11 $\quad\quad$ **for each** $J \in \mathcal{J}$ **do**
12 $\quad\quad\quad$ UPDATE-LABEL (J)

Algorithm 3 JUSTIFY(j, C, A)
1 $\hat{\mathcal{J}} \leftarrow \hat{\mathcal{J}}$.put($j, \langle C, A\rangle$)
2 **for each** $n \in A$ **do**
3 $\quad \hat{\mathcal{J}} \leftarrow \hat{\mathcal{J}}$.put($n, \hat{\mathcal{J}}$.get($n$) $\cup \{j\}$)
4 UPDATE-LABEL (J)

of assumptions where this node holds in, can grow exponentially in the number of assumption nodes.

Suppose n_1 holds label $\{\{A, B\}, \{C\}\}$ and n_2 holds $\{\{A\}, \{C\}\}$. Let ∇ be $\{C, D\}$. If n_1 and n_2 justify γ; γ:$n_1 \wedge n_1 \rightarrow n_3$. The label of n_3 is defined by $\{\{A, B\}, \{A, C\}\}$. This operation is denoted as $\mathcal{Max}$ function in the algorithm.

It removes environments in n_3 subsumed by environments in ∇ nogood set $E \in E' \mid E' \in \nabla$. Here we retract nogood environments from a single label but we expect to have millions of labels in a single network. For example, if there are n assumptions, then there are potentially 2^n contexts (environments).

We broadcast the nogood set to distributed machines to pinpoint environments subsumed by nogoods in parallel. Broadcasting will lead to efficient computation by reducing data movements between machines. However, when the data to be broadcasted becomes large such that it does not fit into the memory, broadcasting can cause a memory overhead problem. In our case, we expect the nogood set to be small enough such that it can fit into the memory and be effortlessly copied into all available nodes. Moreover, we can retract nodes whose assumptions are invalid

in parallel. To utilize those functionalities, we have developed several user interface APIs that interacts with the distributed ATMS such as SHOW-NOGOOD, RETRACT-ASSUMPTION and ASSERT-ASSUMPTION.

4. Evaluation

We run our proposed system on Hadoop cluster with 5 nodes, each equipped with 64Gbyte memory RAM and 20 core CPUs. We evaluated it using ontology benchmark dataset LUBM. The experiments are performed over LUBM ontology datasets with a various number of Universities ranging from 200 (8Gb) to 2000 (80Gb). OWL/RDFS reasoning is applied over each datasets and the results are passed to D-ATMS to construct an ATMS network. We expect the amount of justifications to be much larger, since the conflicting rules can justify the same statement. As a result, the amount of justifications ranges from 28 million to 280 million respectively. Experiments are conducted in terms of three different functionalities; explanation, assertion, and retraction. We believe that these are the most important features that maintenance systems need to possess. For example, from the inferences of LUBM datasets, it explains, why graduate student Alice is a person in the following contexts:

```
>> explain(Person(Student0))
1. domain(telephone, Person) ∧
hasTelephone(Student0, "xxx-xxx-xxxx")
---> Person(Student0)
2. domain(hasEmail, Person) ∧
hasEmail(Student0, student0@depart19.edu)
---> Person(Student0)
3. range(memberOf, Person) ∧
inverseOf(memberOf, member) ∧
member(Student0,department19)
---> Person(Student)
```

Fig. 7 shows the evaluation of explaining inferences with two distinct contexts. OWL reasoning produces *rdf:type* triples from a number of rules. We found that some of the inferences hold about 30 contexts but here we only consider inferences with 3 and 6 contexts. Explanation model only involves getting data from indexed RDD sets. Therefore, runtimes are expected to be extremely low and tend to linearly grow with the number of justifications. In addition, for each context, we need to decode *bitvector* assumptions to human readable texts. In practice, encoding reduces memory occupation massively and enables D-ATMS to maintain large amounts of assumptions in a single environment. It also improves updating labels by reducing computation cost but when it comes to explanation, it shows poor performance. In experiment, we had to decode every assumption respectively.

Therefore, as the number of contexts increases, runtimes slow down. Explanation takes almost $log(n)$ time to perform. Fig. 8 shows the assertion time for the ATMS. We assert a justification as an inference to a node with 3 contexts and 6 contexts and measure the runtimes. Assertion involves minimization checking and label updating. Therefore, it runs slower than explanation. However, for such large datasets, it updates labels incrementally in a reasonably short amount of time as shown in Fig. 8.

We show runtimes for retracting inferences in Fig. 9. Retracting time is a bit faster compared to assertion because it does not require updating propagation.

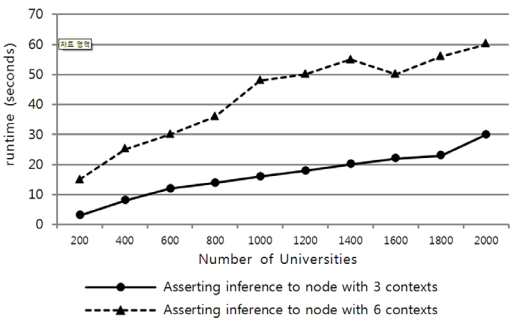

Fig. 8 Explanation for LUBM data sets

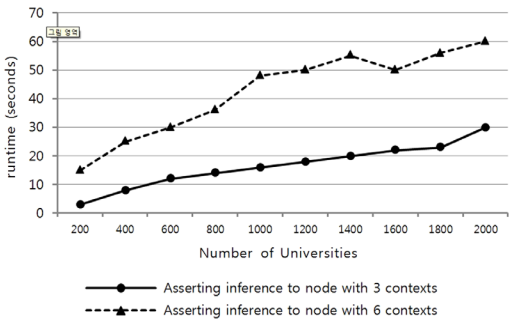

Fig. 9 Assertion to the ATMS

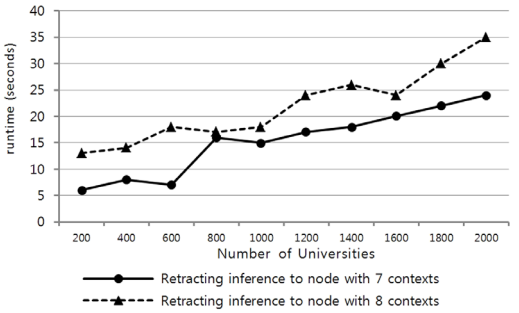

Fig. 10 Retraction from the ATMS

As shown in Fig. 10, retracting can be performed by broadcasting ∇ and transforming $\mathcal{N}$ in parallel. Then, matched environments will be retracted from $\mathcal{N}$ (Algorithm 1). In retraction, $E \in E' \mid E' \in \nabla$ environments are eliminated from the network.

Instead of retracting a justification, we add *false* justification from the assumptions within the node we want to retract (Fig. 9). Finally, overall experiments show encouraging results, it was able to explain large data sets in a few seconds and perform assertion and retraction of inferences efficiently.

5. Conclusion

In traditional ATMS systems, it is prohibitive to keep large amounts of assumptions in memory because of memory insufficiency. Thus, we here propose an approach for distributing ATMS on the cluster to hold a large amount of inferences and enable efficient updates through parallel transformations for non-monotonic reasoning. To minimize data movement between parallel machines, we propose an indexed data structure for storing assumptions and their dependencies which removes I/O bottlenecks and boosts maintaining processes. It is based on a fundamental principle of radix tree that provides a fast inference explanation mechanism for solving problems. The proposed ATMS can be efficiently used in a various type of AI problem solving. To illustrate the efficiency of our system, we evaluate it with LUBM data sets and it was able to explain and provide the solution to a problem over millions of inferences in a few seconds.

References

[1] J. D. Kleer, "A General Labeling Algorithm for Assumption-based Truth Maintenance," *Proc. of the 7th National Conference on Artificial Intelligence*, pp. 188-192, 1988.

[2] N. Hai, N. Alechina, B. Logan, "Axiom Pinpointing Using an Assumption-Based Truth Maintenance System," *Proc. of the 25th International Workshop on Description Logics (DL 2012)*, pp. 290-300, 2012.

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," *Proc. of the 9th USENIX Conference on Networked Systems Design and Implementation*, pp. 2-2, 2012.

[4] Johan de Kleer, "Massively Parallel ATMS," *Proc. of AAAI-88*, pp. 199-204, 1988.

[5] J.-M. Kim, and Y.-T. Park, "An Approach to Detect all Axioms Responsible for Unsatisfiable Concepts in Ontology," *The Journal of KIISE, Software and Applications*, pp. 464-472, 2012.

[6] J.C. Madre, O. Coudert, "A Logically Complete Reasoning Maintenance System Based on a Logical Constraint Solver," *Proc. of IJCAI'9*, Vol. 1, pp. 294-299, 1991.

[7] J. Jones, M. Millington, M. Virvou, "An Assumption-based Truth Maintenance System in Active Aid for UNIX Users," *The Journal of Artificial Intelligence Review*, Vol. 14, pp. 229-252, 2000.

[8] T. Cronin, "Using An Assumption-Based Truth Maintenance System to Switch Context during Data Fusion Processing," Technical Report, 1991.

[9] F. F. Monai, T. Chehire, "Possibilistic Assumption based Truth Maintenance System, Validation in a Data Fusion Application," *Proc. of the 8th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 83-91, 1992.

[10] T.-H. Ngair, G. Provan, "Focusing ATMS Problem-Solving: Formal Approach," Technical Report, 1992.

[11] Ren, Y., Pan, J.Z, "Optimizing Ontology Stream Reasoning with Truth Maintenance System," *Proc. of the ACM Conference on Information and Knowledge Management (CIKM)*, pp. 831-836, 2011.

[12] J. Urbani, A. Margara, C. Jacobs, F. V. Harmelen, H. Bal, "DynamiTE: Parallel Materialization of Dynamic RDF Data," *The Semantic Web-ISWC 2013*, pp. 657-672, 2013.

Batselem Jagvaral
정보과학회논문지
제 43 권 제 1 호 참조

박 영 택
정보과학회논문지
제 43 권 제 1 호 참조