TripleDiff: 트리플 저장소에서 RDF 문서에 대한 점진적 갱신 알고리즘

(TripleDiff: an Incremental Update Algorithm on RDF Documents in Triple Stores)

이 태 휘 † 김 기 성 † 유 상 원 † 김 형 주 ††
(Taewhi Lee) (Kisung Kim) (Sangwon Yoo) (Hyoung-Joo Kim)

요 약 시멘틱 웹(semantic web)과 함께 등장한 RDF는 웹 상의 메타데이타 및 데이타를 나타내는 표준으로 자리매김 하고 있다. 이에 따라 RDF에 대한 저장 및 질의 처리에 대한 연구가 많이 이루어졌으며, 대표적인 시스템으로 Sesame, Jena 등이 있다. 그러나 아직 갱신 방법에 대한 연구는 부족하다. RDF 데이타가 지속적으로 갱신이 이루어지는 경우에는 저장된 RDF를 갱신해야 하는 상황이 발생한다. 현존하는 RDF 저장소에서 데이타를 갱신하기 위해서는 기존의 데이타를 모두 삭제한 후 새로운 데이타를 처음부터 다시 저장해야 하는데, 이러한 상황에서는 매우 비효율적이다. 또한 한 RDF 저장소에 여러 RDF가 저장되어 있는 경우에는 갱신 문제가 더욱 복잡해진다. 이에 본 논문에서는 RDF 데이타를 점진적으로 갱신하는 기법을 제안하고자 한다. 제안한 기법은 텍스트 비교 알고리즘을 통해 얻은 결과를 보완하여 기존 RDF 데이타에서 변화된 트리플 문장만을 추출하여 갱신한다. 실제 RDF 데이타를 이용한 실험을 통해 제안한 방법을 사용하여 갱신을 효율적으로 할 수 있음을 보였다.

키워드 : RDF, 트리플 저장소, 점진적 갱신

Abstract The Resource Description Framework(RDF), which emerged with the semantic web, is settling down as a standard for representing information about the resources in the World Wide Web. Hence, a lot of research on storing and query processing RDF documents has been done and several RDF storage systems, such as Sesame and Jena, have been developed. But the research on updating RDF documents is still insufficient. When a RDF document is changed, data in the RDF triple store also needs to be updated. However, current RDF triple stores don't support incremental update. So updating can be performed only by deleting the old version and then storing the new document. This updating method is very inefficient because RDF documents are steadily updated. Furthermore, it makes worse when several RDF documents are stored in the same database. In this paper, we propose an incremental update algorithm on RDF documents in triple stores. We use a text matching technique for two versions of a RDF document and compensate for the text matching result to find the right target triples to be updated. We show that our approach efficiently update RDF documents through experiments with real-life RDF datasets.

Key words : RDF, Triple store, Incremental update

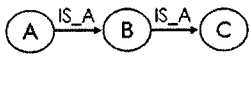
1. 서 론

RDF(Resource Description Framework)[1]는 웹 상의 리소스들에 대한 메타데이타 및 데이타를 표현하기 위한 언어로써 W3C(World Wide Web Consortium)에서 제정한 언어이다. RDF는 시멘틱 웹(semantic web)에 대한 연구가 진행됨에 따라 점점 그 사용이 높아지고 있다. 그 예로 생물학 분야의 경우를 들 수 있다. 용어를 통일하고 데이타의 접근과 공유를 쉽게 하기 위해 Gene Ontology[2], UniProt[3] 등 많은 RDF 문서가

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성지원 사업(IITA-2005-C1090-0502-0016)과 BK21의 연구결과로 수행되었음
† 학생회원 : 서울대학교 전기.컴퓨터공학부
twlee@idb.snu.ac.kr
kskim@idb.snu.ac.kr
swyoo@idb.snu.ac.kr
†† 종신회원 : 서울대학교 전기.컴퓨터공학부 교수
hjk@snu.ac.kr
논문접수 : 2006년 2월 9일
심사완료 : 2006년 6월 25일

사용되고 있다.

대표적인 RDF 저장소로는 Sesame[4]와 Jena[5]가 있다. 이 저장소들은 RDF 데이타를 효율적으로 처리하기 위해 그림 1과 같이 RDF 데이타를 문장(statement)들로 나누어 저장한다. RDF 문서는 주어(subject), 술어(predicate), 목적어(object)의 트리플(triple) 구조를 갖는 문장들로 이루어지며, 이를 DAG(Directed Acyclic Graph) 모델로 나타낼 수 있다. 트리플 저장소(triple store)란 RDF의 문장들을 주어, 술어, 목적어로 나누어 저장하는 저장소를 말한다.



Subject	Predicate	Object
A	IS_A	B
B	IS_A	C
...	...	...

그림 1 RDF 문장을 나타낸 그래프와 이를 트리플 저장소에 저장한 모습

RDF 문서들은 주로 웹을 통해 배포되는데, 시간이 흐름에 따라 변하는 정보를 담고 있는 문서들은 더 정확한 정보, 새로운 정보를 사용자에게 제공하기 위하여 그 내용이 지속적으로 갱신된다. 그 예로 Gene Ontology 데이타는 하루 단위로 매일 갱신되고 있으며, UniProt 데이타의 경우는 2주 단위로 갱신되고 있다. 그림 2를 통해 Gene Ontology 데이타가 2005년 11월에 지속적으로 갱신되었음을 알 수 있다. 이렇게 계속해서 갱신되는 RDF 문서를 저장소에 저장하여 사용할 경우, RDF 문서가 갱신될 때마다 이를 저장소에 반영해 주어야 한다. 그런데 실제로 RDF가 갱신될 때, 어떤 부분이 바뀌었는지에 대한 정보 없이 갱신된 새 버전의 RDF 문서만 제공되는 것이 보통이다. 게다가 현존하는 여러 RDF 저장소들 역시 점진적인 갱신은 고려하지 않고 있기 때문에, 이러한 경우에는 새 버전의 RDF 문서 전체를 다시 저장하여 사용하는 수밖에 없다. 그러나 이러한 방법은 다음과 같은 이유에서 매우 비효율적이다.

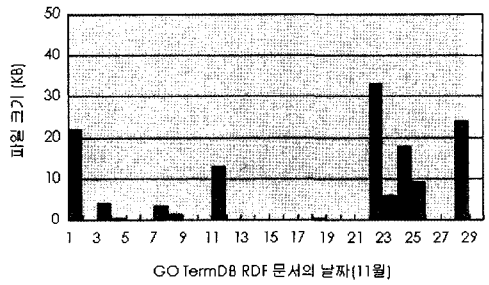


그림 2 2005년 11월의 Gene Ontology TermDB RDF 문서의 변화량 그래프

우선, 전체 RDF 문서에 비해 갱신된 부분의 크기가 작은 경우 이렇게 전체 RDF 문서를 새로 저장하는 것은 이미 올바르게 저장되어 있는 많은 데이타를 반복해서 저장하는 것이기 때문이다. 실제로 Gene Ontology TermDB RDF 문서의 경우를 예로 살펴보면, 이 문서는 하루 단위로 갱신되는데 2005년 11월 중의 이 RDF 문서 하나의 용량은 약 23MB인데 반해, 그 변화한 양은 최대 33KB로 매우 작음을 알 수 있다. UniProt[3] taxonomy RDF 문서의 2005년 11월 23일과 12월 7일 버전의 경우에는 각 문서의 용량은 약 96MB인데 반해 그 변화량은 약 567KB였다.

또한, 전방향 추론(forward chaining)[6]을 사용하는 경우 저장뿐만 아니라 추론 과정까지 반복된다. 현재 대부분의 RDF 저장소는 질의 시간을 빠르게 하기 위해 전방향 추론을 사용한다. 즉, 추론할 수 있는 모든 문장을 미리 추론하여 저장해 둔다. 이러한 추론 과정은 데이타의 크기가 큰 경우 상당히 많은 시간을 필요로 한다. 데이타를 새로 저장하게 되면 이러한 추론 과정도 반복해야 하는 부담이 있다.

마지막으로, 여러 RDF 문서가 하나의 데이타베이스에 저장되어 있는 경우를 생각해보자. 이러한 경우 어떤 문장이 어떤 문서로부터 온 것인지 알 수 없다. 따라서 만약 한 문서가 갱신된 경우 그 문서로부터 온 문장들을 구별할 수 없기 때문에 그 문서만을 지우고 새로 저장할 수가 없다. 즉, 점진적인 갱신 기법이 없으면 저장한 여러 RDF 문서 중 한 문서만이 갱신되었을 때에도 전체 RDF 문서 모두를 새로 저장해야 하므로 그 부담이 더 커지게 된다. 갱신시에 삭제하고 추가해야 할 트리플이 어떤 것인지 알 수 있으면 이러한 비효율을 피할 수 있다.

이러한 이유로 인해 RDF 데이타의 점진적인 갱신 기법이 필요하다. 점진적 갱신을 위해 본 논문에서는 텍스트 비교 알고리즘을 사용한다. 가장 널리 사용되고 있는 방법은 GNU diff[7]로, 텍스트 파일의 바뀐 부분을 줄 단위로 효과적으로 찾아준다. RDF 데이타는 텍스트 문서 형태로 배포되기 때문에 diff를 이용하여 기존 문서와 새 문서를 비교하여 갱신된 부분을 알 수 있다. 하지만 diff 결과를 바로 RDF 데이타의 갱신에 사용하기는 어렵다. 텍스트 비교가 RDF의 반구조(semi-structured)적인 특징을 이해하지 못하여 비교 결과에서 RDF 문서의 구조가 어긋나는 경우가 발생하기 때문이다. 따라서 diff 결과에서 갱신의 대상이 되는 트리플 문장을 알아내는 방법이 필요하다. 이를 위해 본 논문에서는 RDF 데이타에 대해 몇 가지 가정을 한 뒤, diff 결과를 보정하여 갱신 대상이 되는 트리플들을 알아내어 이를 갱신한다. 세운 가정은 다음과 같다.

- RDF 문서를 표현하는 방법이 동일하다.
- 갱신할 때 RDF 문서에 있던 엘리먼트들의 순서는 변하지 않는다.
- 공 노드(blank node)를 사용할 때 rdf:nodeID는 사용하지 않는다.

첫번째 가정과 두번째 가정은 효율성을 위한 가정으로, 이 가정들이 없더라도 동작은 제대로 이루어진다. 세번째 가정의 rdf:nodeID는 고려하지 않았다. 그러나 rdf:nodeID를 사용하지 않고 동일한 의미의 RDF 문서를 작성할 수 있다. 실제 배포되는 데이타를 보면 이러한 가정이 타당함을 있음을 알 수 있다.

실험은 실제 데이타인 Gene Ontology TermDB RDF 문서와 UniProt taxonomy RDF 문서를 가지고 수행하였다. 실험을 통해 저장 시간의 단축과 갱신의 정확성을 보였고, 갱신된 문장의 수에 따라 수행시간이 어떻게 변하는지 확인하였다.

논문의 전체 구성은 다음과 같다. 2장에서는 본 논문과 관련된 배경 지식 및 선행 연구들에 대해 살펴본다. 3장에서는 RDF의 점진적 갱신을 위해 일반 텍스트 비교를 사용할 경우의 문제점을 지적하고, 4장에서 이를 보완해서 이용하는 TripleDiff라는 점진적 갱신 기법을 제안한다. 5장에서는 갱신된 문서를 새로 저장하는 것과 TripleDiff 기법의 성능을 비교하고 TripleDiff 기법이 갱신을 올바로 수행하는지, 데이타의 차이에 따라 어떻게 달라지는지 검증하며, 6장에서 결론과 향후 연구 방향에 대해 제시한다.

2. 관련 연구

온톨로지의 변화를 찾고 이를 버전별로 이용할 수 있도록 하는 연구가 [7]에서 이루어졌다. 이 연구에서는 온톨로지의 버전이 다르더라도 최대한 호환되어 사용이 가능하도록 그래프 매칭을 통해 두 버전이 어떻게 다른지 사용자에게 보여주고 변화한 개념들간의 관계를 사용자가 지정할 수 있도록 하였다.

[8,9]에서는 RDF 그래프의 매칭 기법을 제안하였다. 그래프를 공 노드가 트리플의 어느 부분에 있느냐에 따라 트리플을 여러 클래스로 분류하고 같은 클래스에 속한 트리플간에만 비교하도록 하여 그래프를 매칭할 때 그 비교 공간을 줄였다. 또한 전자 서명(digital signature)을 목적으로 한 후속 연구에서 공 노드의 식별자를 무시한 채 트리플들을 정렬한 후 이를 토대로 비교하는 기법을 제시하였는데, 이 연구의 단점은 문서가 극히 일부만 변하는 경우에도 많은 양의 텍스트 비교 결과를 초래할 수 있다는 점이다.

그리고 [10]은 RDF 문서가 갱신되었을 때 어떻게 갱신되었는지 그 갱신 정보를 표현하는 데 사용되는 delta 온톨로지를 제안하였다. 이 온톨로지를 이용하여 어떤 트리플이 추가, 삭제, 변경되었는지를 배포하자는 것이다. 그러나 delta 온톨로지는 제안에 그치고 있으며 현재 실제로 사용되지는 않는다.

텍스트 비교에 관해서는 GNU diff[11]가 대표적이다. GNU diff는 두 파일 간의 차이를 줄 단위로 출력해 주며, 매우 효율적이고 빠르다는 장점을 지니고 있다. 하지만 텍스트 비교는 반구조(semi-structured) 데이타의 구조를 이해하지 못하기 때문에[12] 이를 그대로 RDF 문서들의 차이를 알아내는 데 이용할 수는 없다. 그리고 아직까지 대용량 문서에 대해서는 동작이 불완전한데, 이는 향후에 개선할 목록에 포함되어 있기 때문에[13] 이는 추후에 해결될 것으로 보인다.

RDF 저장소 시스템으로는 대표적인 것으로 Sesame [4]와 Jena[5]가 있는데, 이 두 저장소 모두 하부 데이타베이스로 RDBMS 등을 사용할 수 있으며, 전방향 추론을 사용하는 영속적 모델(persistent model)을 제공한다. 그러나 두 시스템의 최근 안정 버전인 Sesame 1.2.4와 Jena 2.4 모두 모델을 수정하는 기능만 제공할 뿐, 두 RDF 문서의 차이점을 찾아내어 점진적으로 갱신하는 기능은 제공하지 못한다. 따라서 1장에서 설명한 환경들에서 매우 비효율적이다.

3. 문제 설명

본 장에서는 RDF의 점진적 갱신을 위해 일반 텍스트 비교를 사용할 경우의 문제점들을 살펴본다.

3.1 문제 정의

RDF 문서를 저장소에 저장하여 사용하고 있는데 이 RDF 문서가 갱신이 되었을 경우를 생각해보자. 본 논문의 목적은 이러한 경우에 갱신된 문서를 새로 저장하지 않고 기존에 저장해서 사용하고 있는 RDF 문서와 새 버전의 RDF 문서를 비교하여 그 변화된 부분을 얻어낸 뒤, 이를 이용하여 갱신을 효과적으로 수행하겠다는 것이다. 이 목적은 서론에서 살펴본 것과 같이 RDF 문서의 크기에 비해 그 변화량이 매우 작다는 사실에 기인한 것이다. 전체 시나리오는 그림 3과 같다.

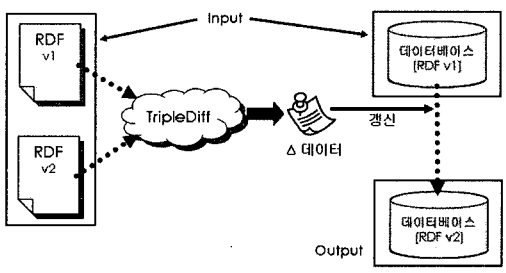


그림 3 TripleDiff의 전체 시나리오

두 RDF 문서의 변화 부분을 알아내기 위해서는 효율성을 위해 텍스트 비교를 이용한다. 그러나 텍스트 비교 결과를 갱신에 그대로 사용할 수는 없기 때문에 이를 보완하는 형태를 취하는데, 이를 위해서는 다음과 같은 몇 가지 가정이 필요하다.

- RDF 문서를 표현하는 방법이 동일하다.
- 갱신할 때 RDF 문서에 있던 엘리먼트들의 순서는 변하지 않는다.
- 공 노드를 사용할 때 rdf:nodeID는 사용하지 않는다.

우선, 첫째 가정은 문서의 띄어쓰기와 같은 작성 방법이 동일하다는 것이다. 한 RDF 문서는 보통 특정 기관과 같이 일정한 배포자가 주관하여 배포하기 때문에 표현 규칙이 동일하게 유지되며, 따라서 이 가정은 실제 세계에서도 일반적으로 성립한다. 이 가정이 지켜지지 않았을 경우에는 적절한 문서 변환 과정을 통해 지켜지도록 할 수 있다. 둘째 가정은 RDF 문서에서는 원래 순서의 변화가 영향을 미치지 않아 기존에 있던 엘리먼트들의 순서가 바뀌어도 그 내용은 동일한데, 갱신 과정에서 엘리먼트들의 순서가 변하지 않는다고 가정한 것이다. 부가적으로, 동일한 트리플을 여러 형태의 RDF 구문으로 나타낼 수 있는데 이것 역시 갱신 과정에서 변하지 않는다고 가정한다. 이 가정이 지켜지지 않았을 경우에도 갱신은 가능하나, 텍스트 비교 결과가 커져 갱신 시간이 오래 걸리게 된다. 마지막 가정은 RDF에서 공 노드를 사용할 때, 특정 공 노드를 가리킬 수 있도록 문서 내에서만 사용될 수 있는 식별자를 기술하여 사용할 수 있는데 이를 배제한 것이다. rdf:nodeID를 사용하지 않고도 동일한 내용을 기술할 수 있기 때문에 이 가정도 지켜질 수 있다.

실제 사용되고 있는 RDF 데이타와 이들의 변화 패턴을 살펴보면 이 가정이 타당하다는 것을 알 수 있다. 갱신이 엘리먼트 단위로 삭제되거나 추가되며, 전체적인 구조의 변화는 거의 일어나지 않기 때문이다.

3.2 RDF 갱신 예제

그림 4, 5는 본 논문 전체에서 사용할 예제이다.

그림 4의 RDF 문서가 데이타베이스에 저장되어 있고, 새로운 버전의 RDF 문서가 그림 5과 같을 때, 갱신 사항을 데이타베이스에 반영하기 위해서는 그림 6과 같은 연산들을 수행해야 한다.

따라서, 관건은 두 RDF 문서를 텍스트 비교하였을 때 그림 6의 연산의 대상이 되는 트리플들을 두 RDF 문서의 텍스트 비교를 통해서 알아낼 수 있는가 하는 것이다. 그림 7은 두 RDF 문서의 차이를 GNU diff를 통해 얻어낸 결과다.

3.3 일반 텍스트 비교를 사용했을 경우의 문제점

3.2절의 예에서 알 수 있듯이 diff의 결과(그림 7)만으로는 그림 6의 갱신 연산을 알아내기가 어렵다. 이 절에서는 왜 diff 결과만으로 갱신해야 할 RDF 문장을 알

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <rdf:RDF xmlns:go="http://www.geneontology.org/go#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
3    <go:term rdf:about="http://www.geneontology.org/go#GO:0000001">
4      <go:name>mitochondrion inheritance</go:name>
5    <go:definition>The distribution of mitochondria, including the mitochondrial genome, ...</go:definition>
6      <go:is_a rdf:resource="http://www.geneontology.org/go#go:0048308">
7      <go:is_a rdf:resource="http://www.geneontology.org/go#go:0048311">
8    </go:term>
9    <go:term rdf:about="http://www.geneontology.org/go#GO:0000002">
10     <go:name>mitochondrial genome maintenance</go:name>
11     <go:definition>The maintenance of the structure and integrity of the mitochondrial genome.</go:definition>
12     <go:is_a rdf:resource="http://www.geneontology.org/go#GO:0007005" />
13     <go:dbxref rdf:parseType="Resource">
14      <go:database_symbol>Pfam</go:database_symbol>
15      <go:reference>PF06420 Mgm101p</go:reference>
16     </go:dbxref>
17     <go:dbxref rdf:parseType="Resource">
18      <go:database_symbol>InterPro</go:database_symbol>
19      <go:reference>IPR009446</go:reference>
20     </go:dbxref>
21     <go:dbxref rdf:parseType="Resource">
22      <go:database_symbol>InterPro</go:database_symbol>
23      <go:reference>IPR009447</go:reference>
24     </go:dbxref>
25   </go:term>
26 </rdf:RDF>
```

그림 4 이전 버전의 RDF 문서

```
1  <?xml version='1.0' encoding='UTF-8'?>
2   <rdf:RDF xmlns:go="http://www.geneontology.org/go#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#">
3       <go:term rdf:about="http://www.geneontology.org/go#GO:0000001">
4           <go:name>mitochondrion inheritance</go:name>
5               <go:definition>The distribution of mitochondria, including the mitochondrial genome,
...</go:definition>
6           <go:is_a rdf:resource="http://www.geneontology.org/go#go:0048311">
7           <go:is_a rdf:resource="http://www.geneontology.org/go#go:0048312">
8       </go:term>
9       <go:term rdf:about="http://www.geneontology.org/go#GO:0000002">
10          <go:name>mitochondrial genome maintenance</go:name>
11          <go:definition>This line was changed.</go:definition>
12          <go:is_a rdf:resource="http://www.geneontology.org/go#GO:0007755" />
13          <go:dbxref rdf:parseType="Resource">
14              <go:database_symbol>Pfam</go:database_symbol>
15              <go:reference>PF06420 Mgm101p</go:reference>
16          </go:dbxref>
                                        <4줄삭제>
17          <go:dbxref rdf:parseType="Resource">
18              <go:database_symbol>InterPro</go:database_symbol>
19              <go:reference>IPR009447</go:reference>
20          </go:dbxref>
21      </go:term>
22 </rdf:RDF>
```

그림 5 새 버전의 RDF 문서

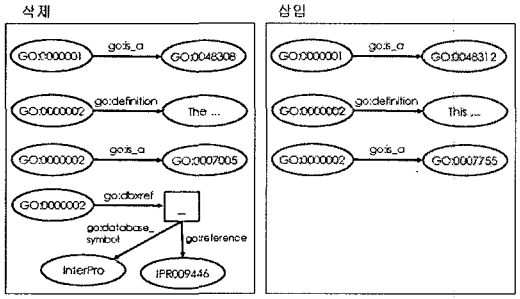


그림 6 예제에서 갱신을 반영하기 위해 수행해야 하는 연산

수 없는지 그 이유를 살펴본다.

3.3.1 엘리먼트 함유가 어긋난 경우

가장 먼저 생각할 수 있는 텍스트 비교의 문제점은 엘리먼트의 함유(nesting)가 잘못된 비교 결과를 만들어 낼 수 있다는 점이다. 실제 RDF 문서의 갱신은 엘리먼트 단위로 이루어지는데, 텍스트 비교의 경우 단순히 줄 단위로 두 문서를 비교하기 때문에 이와 같은 현상이 발생한다. 예에서 GNU diff 결과가

```
19,22d18
<         <go:reference>IPR009446</go:reference>
<     </go:dbxref>
<     <go:dbxref rdf:parseType="Resource">
<         <go:database_symbol>InterPro</go:
          database_symbol>
```

인 부분을 살펴보자. 이 결과에 해당하는 이전 버전의 RDF 문서의 부분이 19~22번째 줄로 되어 있다. 그런데, 두 문서를 살펴보면 원래 의도했던 갱신은 17~20번째 줄의 삭제임을 알 수 있다. 그리고 결과를 살펴보면 엘리먼트의 함유가 잘못되어 있음을 알 수 있다. 엘리먼트가 열리지 않은 채 </go:dbxref>라는 닫는 부분이 먼저 나와 있다. 이렇게 텍스트 비교 결과에서 엘리먼트의 함유가 잘못된 경우는 결과가 원래 의도했던 갱신을 제대로 찾지 못한 것이며, 이 결과에 해당하는 트리플을 찾아 갱신을 하면 그림 8에서 보는 것과 같이 실제 갱신과 의미가 다른 갱신이 수행된다.

3.3.2 트리플의 일부를 알 수 없는 경우

두 RDF 문서를 텍스트 비교하였을 경우 엘리먼트의 함유가 제대로 되었다고 할지라도 트리플을 구성하는 모든 요소들을 알 수 없는 경우가 발생한다. 예에서 GNU diff 결과가

```
7a7
>       <go:is_a rdf:resource="http://www.geneontology.
        org/go#GO:0048312" />
```

인 부분을 살펴보자. 이 결과가 시사하는 것은 트리플 (http:// geneontology.org/go#GO:0000001 go:is_a http:// www.geneontology.org/go#GO:0048312)의 삽입이지만, 결과만으로는 트리플의 주어를 알 수 없다. 이와 같이 트리플을 구성하는 요소들 중 일부를 알 수 없는 경우가 발생하게 된다.

3.3.3 트리플에 공 노드가 포함되어 있는 경우

트리플의 정보를 모두 알더라도 갱신의 대상이 되는 트리플에 공 노드가 포함되어 있는 경우 이를 처리하는 것이 문제가 된다. 그림 9와 같이 RDF에는 공 노드를

```
6d5
<                <go:is_a rdf:resource="http://www.geneontology.org/go#GO:0048308" />
7a7
>                <go:is_a rdf:resource="http://www.geneontology.org/go#GO:0048312" />
11,12c11,12
<                  <go:definition>The maintenance of the structure and integrity of the mitochondrial
genome.</go:definition>
<                <go:is_a rdf:resource="http://www.geneontology.org/go#GO:0007005" />
---
>                <go:definition>This line was changed.</go:definition>
>                <go:is_a rdf:resource="http://www.geneontology.org/go#GO:0007755" />
19,22d18
<                        <go:reference>IPR009446</go:reference>
<                </go:dbxref>
<                <go:dbxref rdf:parseType="Resource">
<                        <go:database_symbol>InterPro</go:database_symbol>
```

그림 7 두 RDF 문서를 텍스트 비교한 결과

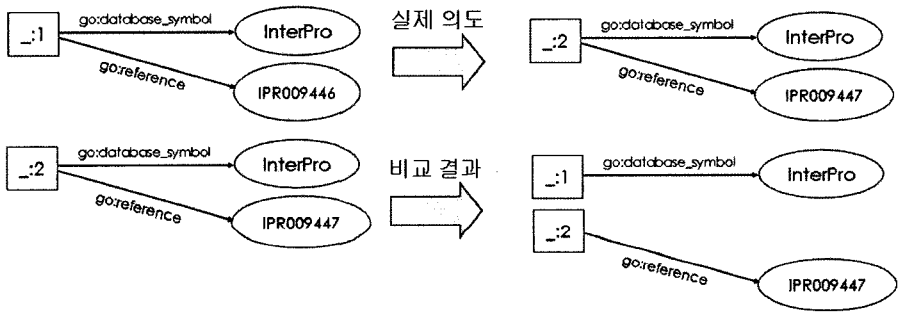


그림 8 엘리먼트의 함유가 잘못된 텍스트 비교 결과에 따른 의도와 맞지 않는 갱신

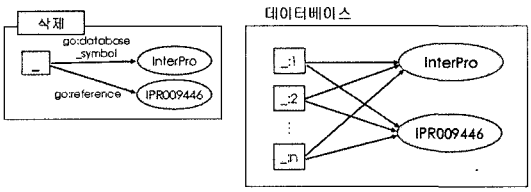


그림 9 트리플에 공 노드가 포함되어 있는 경우

포함하면서 같은 구조를 갖는 트리플이 여러 개가 있을 수 있다. 이러한 경우 트리플을 알아냈다 하더라도 데이타베이스에 갱신을 할 때 정확한 대상 트리플이 어떤 것인지 알 수 없는 경우가 발생한다.

4. TripleDiff: RDF 문서의 점진적 갱신 알고리즘

이 장에서는 3장에서 살펴본 일반 텍스트 비교를 사용했을 경우의 문제점들을 보완하여RDF 문서를 점진적으로 갱신하는 TripleDiff 알고리즘을 제안하고, 각 문제점들을 어떻게 해결하였는지 설명한다.

4.1 어긋난 엘리먼트 함유의 교정

3.3.1절에서 텍스트 비교가 엘리먼트의 함유가 잘못된 결과를 생성할 수 있다는 사실을 살펴보았다. 따라서 잘못된 함유를 바로잡아주는 과정이 필요하다. 함유가 잘못되었는지는 비교 결과에서 엘리먼트의 열고 닫음이 제대로 되었는지를 보면 알 수 있다. 그렇기 때문에, 텍스트 비교 결과의 내용을 파싱할 때 엘리먼트가 열고 닫힐 때마다 스택을 이용해서 열고 닫음이 올바른지 검사하여 다음과 같이 함유를 바로잡아주면 된다.

- 여는 태그가 없이 닫는 태그가 먼저 나온 경우, 해당 문서를 읽어 닫는 태그가 나온 줄부터 윗줄로 거슬러 올라가며 그 태그의 여는 태그를 찾아 그 줄부터 연산이 시작된 것으로 조정한다.
- 여는 태그가 나왔으나 닫히지 않고 비교 결과가 끝난 경우, 해당 문서를 읽어 비교 결과의 마지막 줄부터 그 태그의 닫는 태그를 찾아 그 줄까지 연산이 해당되는 것으로 조정한다.
- 공통적으로 삭제 또는 삽입의 범위를 조정한 경우 그 조정된 범위 만큼 반대 연산을 추가시켜 준다. 즉, 더 삭제하게 된 부분을 다시 삽입시켜 주거나 더 삽입하게 된 부분을 미리 삭제하는 연산을 추가시켜 주는 것이다. 반대 연산의 추가 범위는 연산의 조정 범위와 같다. 예를 들어 위로 두 줄을 조정하였으면 반대 연산도 위로 두 줄을 추가하고, 아래로 두 줄을 조정하였으면 반대 연산도 아래로 두 줄을 추가한다.

여기에서 해당 문서란 연산 종류가 delete나 change

의 delete 부분일 경우 이전 버전의 문서를, 연산 종류가 insert나 change의 insert 부분일 경우 새 버전의 문서를 가리킨다. 예에서 GNU diff 결과가

```
19,22d18
<      <go:reference>IPR009446</go:reference>
<   </go:dbxref>
<   <go:dbxref rdf:parseType="Resource">
<      <go:database_symbol>InterPro</go:database_ symbol>
```

인 부분의 경우, 이러한 교정 단계를 거쳐 17,24d16과 17a17,20으로 바뀌게 되며, 이를 합하여 17,24c17,20으로 바꿀 수 있다. 그림 6처럼 교정 결과가 최적이 되지는 않지만, 올바르게 갱신할 수 있는 트리플 집합이 만들어진다.

4.2 줄 번호 스택의 사용

3.3.2절에서 살펴본 것과 같이 엘리먼트의 함유가 제대로 된 경우에도 트리플 전체를 알 수 없는 문제가 발생하게 되는데, 각 엘리먼트의 줄 번호를 저장하는 스택을 사용함으로써 이를 해결할 수 있다.

먼저 4.1절의 방법으로 어긋난 엘리먼트 함유를 교정한 후 텍스트 비교 결과에서 미리 명령 리스트와 그에 해당하는 줄 번호들을 얻는다. 그 후, 파서는 이전 버전과 새 버전의 RDF 문서를 차례로 파싱하게 된다. 파서는 파싱 중에 엘리먼트가 여는 태그가 나오면 이 엘리먼트의 이름을 스택에 저장하고 닫는 태그가 나오면 스택에서 이 엘리먼트를 제거하는 작업을 하면서, RDF 문장이 형성되면 이 트리플을 처리한다.

우리의 해결 방법은 파서가 스택에 엘리먼트의 이름을 저장하고 제거할 때, 이와 함께 엘리먼트가 존재했던 줄의 번호를 스택에 저장하도록 하는 것이다. 그리고 RDF 문장의 엘리먼트 중에서 줄 번호가 서로 일치하는 것이 있는 경우에만 명령을 수행하도록 한다. 이렇게 함으로써 텍스트 비교를 통해 전체 트리플을 알 수는 없지만, 대신 RDF 문서에서 파싱된 트리플이 비교 결과에 해당되는 것인지를 알 수 있게 된다. 여기서, 이전 버전의 문서를 파싱할 때는 delete와 change 명령만 고려하며, 새 버전의 문서를 파싱할 때는 add와 change 명령만 고려한다(change 명령은 delete + add 명령으로 생각할 수 있다.).

예에서 GNU diff 결과가

```
7a7
>      <go:is_a rdf:resource="http://www.geneontology.
       org/go#GO:0048312" />
```

인 부분을 살펴보자. Diff 결과를 파싱할 때 새 버전의 문서의 7번째 줄이 add 연산에 해당된다는 것을 미리 operationList에 저장해 놓게 된다. 이후 파서가 새 버전의 문서인 그림 6을 파싱할 때, 7번째 줄에 도달하여 그림 10과 같은 RDF 문장을 형성하게 된다. 이 때, 줄 번호 스택을 사용함에 따라 각 엘리먼트가 몇 번째 줄에서 온 것인지를 알 수 있다. 그리고 이 줄 번호들 중에서 7이라는 번호가 add 연산에 해당하는 줄과 일치하므로 이를 데이타베이스에 삽입해 주면 된다. delete 연산의 경우에도 마찬가지 방법으로 해결할 수 있다.

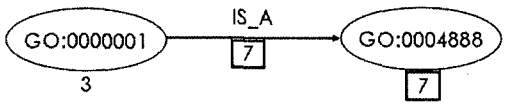


그림 10 줄 번호 스택의 사용 예

4.3 공 노드가 포함된 트리플의 처리

갱신의 대상이 되는 트리플에 공 노드가 포함되어 있는 경우에 대해서는 [10]에 공 노드를 간접적으로 이름이 있는 노드로부터의 경로를 통해 식별할 수 있다고 언급되어 있다.

공 노드는 다음과 같은 성질을 지니고 있다.

- 공 노드는 주어나 목적어에 나타날 수 있다.
- 이름이 있는 노드로부터 공 노드까지의 경로는 유일하거나, 그렇지 않을 경우는 공 노드에는 이름이 없기 때문에 서로 구분할 필요가 없다.

따라서 트리플의 주어와 목적어를 검사하여 공 노드가 포함되어 있는 경우, 파서가 파싱을 진행할 때 열린 후 아직 닫히지 않은 엘리먼트들의 이름을 담고 있는 elementStack을 이용하여 이름이 있는 가장 가까운 노드로부터의 경로를 알아낸 뒤, 데이타베이스에 이 확장한 경로를 사용하여 질의를 함으로써 연산의 대상이 되는 트리플을 찾아낼 수 있다.

4.4 TripleDiff 알고리즘

그림 11은 제안하는 RDF 문서의 점진적 갱신 기법인 TripleDiff 알고리즘을 보여준다.

TripleDiff 알고리즘은 크게 텍스트 비교 결과를 파싱하는 단계와 파서가 RDF 문서를 파싱하는 단계로 나눌 수 있다. 텍스트 비교 결과를 파싱하는 단계에서는 결과를 읽어 갱신해야 할 명령과 해당 줄 번호의 리스트를 얻는데, 4.1절에서 설명한 대로 중간 과정에서 필요한 조정 단계를 거친다. 그리고 RDF 문서의 파싱 단계에서는 4.2절에서 설명한 줄 번호 스택을 사용하여 엘리먼트의 줄 번호를 기억해 두고 이를 이용하여 갱신의 대상이 되는 트리플을 구별한다. 그리고 트리플에 공 노드가 있을 경우 4.3절에 설명한 대로 경로를 확장한다. RDF 문서를 파싱할 때는 기존 버전의 RDF 문서와 새 버전의 RDF 문서를 모두 파싱하여 처리하게 된다. 이는 기존 버전의 문서에서는 트리플의 삭제와 관련된 정보가, 새 버전의 문서에서는 트리플의 삽입과 관련된 정보가 필요하기 때문이다.

Algorithm TripleDiff

```
// operationList : diff 결과로부터 얻은 연산 종류(add, change, delete)와 해당 줄 번호를 저장하는 리스트
// elementStack : 파서가 RDF 문서를 파싱하는 도중에 엘리먼트 이름을 저장하는 스택
// lineNumStack : elementStack에 저장된 엘리먼트들의 줄 번호를 저장하는 스택
// diffStack : diff 결과를 파싱할 때 임시로 사용하는 스택
1:  Get diff-result from the old RDF document and the new RDF document by diff algorithm;
/* diff 결과로부터 operationList를 얻는다. 중간 과정에서 엘리먼트 함유가 어긋난 것을 바로잡는다. */
2:  diffline = diff-result.readline();
3:  WHILE diffline is not null
4:    IF diffline is operation line THEN          // a line describing operation type and line numbers
5:      operationList.add(diffOperationLineParse(diffline));
6:    WHILE (diffline = diff-result.readline()) is not operation line
7:      IF there is an element-open-tag in diffline THEN diffStack.push(open-element-name);
8:      IF there is an element-close-tag in diffline THEN
9:        IF diffStack.isEmpty() THEN expand the target range of the operation in operationList upward to the line
having an open-tag of the element
10:       ELSE diffStack.pop();
11:   END WHILE
12:   IF !diffStack.isEmpty() THEN expand the target range of the operation in operationList downward to the line
having a close-tag of the element
13:  END WHILE
/* 파서가 RDF 문서를 파싱할 때 lineNumStack에 엘리먼트의 줄 번호를 함께 저장하고 RDF 문장이
형성되었을 때 각 엘리먼트의 줄 번호가 operationList의 줄 번호와 일치하는 것이 있을 경우에만 명령
을 수행한다. */
14:  WHILE parser parses two RDF documents in order
15:    IF parser pushes an element to elementStack THEN lineNumStack.push(the-line-number-of-the-element);
16:    IF parser creates RDF statement THEN
/* 단, 여기서 고려하는 operationList의 명령은 이전 버전의 RDF 문서를 파싱할 때에는 delete와 change
만, 새 버전의 RDF 문서를 파싱할 때에는 add와 change만 고려한다. */
17:      IF the line number of operationList == one of the line numbers of S, P, O elements of the statement THEN
           /* 트리플에 공 노드가 있을 경우에는 이를 경로를 확장하여 처리한다. */
18:        IF S or O of the statement is a blank node THEN expand the path of the triple
19:        process the statement (add or remove)
20:  END WHILE
```

그림 11 TripleDiff 알고리즘

두 문서의 크기를 m, n이라고 할 때, 이 알고리즘의 시간 복잡도는 diff 알고리즘이 O(m+n)이고 그 뒤의 과정도 두 문서를 읽어서 파싱하고 비교하는 등의 일밖에 하지 않기 때문에 O(m+n)이 된다. 그러나 한 가지 주지할 점은 갱신에 소요되는 시간은 diff 결과의 크기, 즉 갱신된 문장의 수에 비례한다는 점이다. 파싱하는 데 걸리는 시간은 데이타를 저장하고 삭제하는 연산에 비해 매우 짧다. 데이타베이스에 SQL 연산을 처리하는 시간에 비하면 파싱하는 데 걸리는 시간은 무시할 만한 수준이기 때문에, 텍스트 비교 결과의 크기에 비례하는 갱신된 문장의 수가 수행시간에 중요한 영향을 미친다.

5. 성능 평가

5.1 실험 환경과 실험 데이타

본 실험은 Pentium 4-3.2GHz, 2GB RAM 사양의 컴퓨터에서 수행하였으며 OS는 Linux(Fedora Core 3)를 사용하였다. TripleDiff 알고리즘의 구현은 대표적인 RDF 저장소인 Sesame 1.2.2를 수정하여 구현하였으며, RdfRepository SAIL을 이용하여 저장하였다. Java SDK 1.5.0으로 구현하였으며, 하부 데이타베이스로는 MySQL 4.1.9를 사용하였다. 실험 데이타는 실제 데이타인 Gene Ontology TermDB 의 2005년 11월 22일과 11월 23일자 RDF 문서(약 23MB, diff 결과 33KB), UniProt

Taxonomy의 2005년 11월 23일과 12월 7일자 RDF 문서(약 96MB, diff 결과 567KB)를 이용하였다.

5.2 실험 결과

실험은 다음의 세 가지로 나누어 진행하였다.

- 실제 데이타를 대상으로 한 성능 측정
- TripleDiff 기법을 사용했을 때 갱신이 올바로 이루어지는지 확인
- 갱신된 문장의 수에 따라 수행 시간이 어떻게 변하는지 분석

수행 시간 측정에서는 리소스를 삭제하는 시간은 제외하였다. Sesame에서는 문장을 삭제할 때, 삭제되어 더 이상 사용되지 않는 리소스를 삭제하는 작업이 수행된다. 이 작업은 Sesame의 구현과 관련되어 수행되는 특화된 작업이며, 우리의 목적인 트리플의 갱신과는 무관하기 때문에 시간 측정에서 제외하였다.

먼저 실험 데이타를 대상으로 하여 새 버전의 RDF 문서를 새로 저장하는 데 걸리는 시간과 TripleDiff 기법을 사용하여 갱신하였을 때 걸리는 시간을 비교해 보았다. 결과는 그림 12와 같다.

텍스트 비교 크기, 즉 갱신된 양이 매우 작기 때문에 TripleDiff를 사용하여 데이타를 갱신하였을 때의 수행 시간이 훨씬 적음을 알 수 있다.

표 1은 TripleDiff 기법을 사용했을 때 갱신이 올바로 이루어지는지를 확인하기 위한 것이다. 갱신된 RDF 문서를 새로 저장했을 경우와 이전 버전의 RDF 문서를 TripleDiff를 사용하여 갱신한 경우의 총 문장 개수가 일치함을 확인하였으며, SQL 질의를 통하여 두 데이타베이스가 완전히 동일함을 확인하였다.

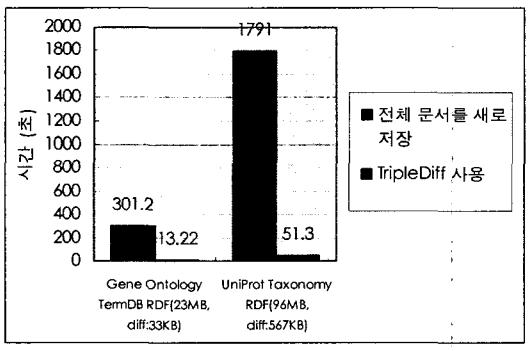


그림 12 실제 데이타를 대상으로 한 수행 시간의 비교

그림 13은 Gene Ontology TermDB RDF 파일을 대상으로 삭제 또는 추가되는 문장의 개수를 달리 하여 TripleDiff의 수행 시간을 측정한 것이다. 데이타베이스에서 문장을 삭제하는 데 걸리는 시간이 추가하는 데 걸리는 시간보다 더 길다는 점을 감안하여 두 연산을 완전히 분리시켜 실험을 해 보았다. 데이타는 삭제 연산 실험의 경우 Gene Ontology TermDB의 2005년 11월 23일자 파일(23MB, 294,650문장)의 문장을 점차적으로 삭제하여 만들었으며, 추가 연산 실험에서는 삭제 실험에서 만든 데이타를 역으로 이용하였다.

실험 결과를 보면, 문장 삭제의 경우 약 75,000개(전체 문서의 약 25%)의 문장을 삭제할 때 전체 문서를 새로 저장하는 것과 TripleDiff의 수행 시간이 같아지는 것을 볼 수 있다. 반면 문장 추가의 경우에는 10만 개(전체 문서의 약 35%)가 넘어도 전체 문서를 새로 저장하는 것보다 TripleDiff의 성능이 좋음을 알 수 있다. 실제로 사용되는 데이타의 갱신은 새로운 데이타가 추가되는 비중이 높으며, 실험의 경우처럼 삭제가 많이 일어나는 경우는 드물다. 또한 지속적으로 갱신되는 데이타의 경우 전체 문서의 25%가 변화할 만큼 큰 변화가 있는 경우는 없다고 할 수 있다.

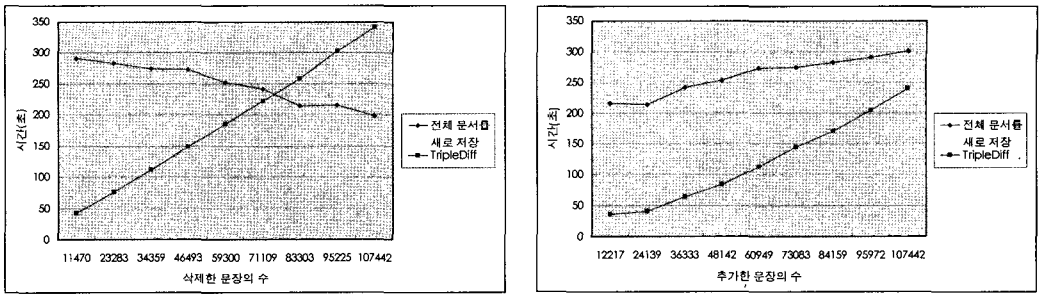
종합해 볼 때, TripleDiff는 지속적으로 갱신되는 RDF 데이타를 효율적으로 갱신할 수 있다. 그 효과는 문서의 변화한 양이 적고 삭제되는 문장의 수가 적을수록 더 커진다. 최악의 경우를 가정해 보면, 전체 문서의 25% 이상이 갱신되었을 경우에는 TripleDiff를 사용하는 것보다 문서를 새로 저장하는 것이 오히려 더 효과적이다. 그러나 실제 데이타에서 이러한 경우는 드물고, 서론에 기술한 것과 같이 여러 RDF 문서가 하나의 데이타베이스에 저장될 수도 있기 때문에 이러한 점진적 갱신 알고리즘은 꼭 필요하다.

6. 결론 및 향후 연구과제

우리는 본 연구에서 RDF 문서가 지속적으로 갱신되는 경우 이를 데이타베이스에 효과적으로 반영하는 TripleDiff라는 점진적인 갱신 알고리즘을 제안하였다. 이 기법은 텍스트 비교를 이용하여 이전 버전의 RDF 문서와 새 버전의 RDF 문서의 차이를 알아낸 후, 이에

표 1 TripleDiff를 이용한 갱신의 정확성 검사

	GO TermDB 20051122 + TripleDiff	GO TermDB 20051123
저장되어 있는 문장	294,467	294,650
삭제된 문장(TripleDiff)	333	-
추가된 문장(TripleDiff)	516	-
총 문장	294,650	294,650



(a) 삭제 (b) 추가

그림 13 갱신된 문장의 수에 따른 수행 시간의 비교

텍스트 비교의 문제점을 보완하여 갱신의 대상이 되는 트리플들을 알아내어 새 버전의 RDF 문서 전체를 새로 저장하지 않고 효율적으로 갱신할 수 있게 해준다. 그리고 실험을 통해 갱신의 크기가 약 25%를 넘지 않는 경우 새 버전의 RDF 문서를 저장하는 것보다 TripleDiff를 이용하는 것이 더 효율적임을 확인하였다.

본 연구는 향후에 연구실에서 계획하고 있는 OASIS (Omics AnalySIS for microbial organisms) 시스템에 사용될 예정이다. OASIS 시스템은 여러 생물학 데이타베이스와 분석 도구들을 통합하여 연구자에게 좀 더 정확한 예측이나 새로운 시각을 제공하기 위한 시스템이다. 따라서 이 시스템에서는 RDF로 된 대용량의 여러 생물학 데이타베이스를 저장해서 사용하고, 그 위에 여러 응용 도구들이 자리잡게 된다. 이 때, 데이타베이스에 저장되어 있는 데이타들을 항상 최신으로 유지할 필요가 있다. 또한 이 데이타들은 조금씩 계속해서 갱신된다. 본 연구는 이러한 목적에 부합되기 때문에 시스템에 효과적으로 사용될 것으로 기대하고 있다.

참 고 문 헌

[1] Graham Klyne, Jeremy J. Carroll, and Brian McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, 2004.
[2] Gene Ontology Consortium, http://www.geneontology. org
[3] UniProt RDF, http://www.isb-sib.ch/%7Eejain/rdf/
[4] Jeen Broekstra, et. al. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, In Proceedings of the International Semantic Web Conference, 2002.
[5] Kevin Wilkinson, et. al. Efficient RDF Storage and Retrieval in Jena2. In Proceedings of the first International Workshop on Semantic Web and Databases, 2003.
[6] 김기성, 유상원, 이태휘, 김형주. RDFS 함의 규칙 적용 순서를 고려한 전방향 RDFS 추론 엔진의 최적화. 정보과학회논문지:데이타베이스, 33(2), 2006.
[7] Michael Klein, et. al. Ontology Versioning and Change Detection on the Web. In Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management, 2002.
[8] Jeremy J. Carroll. Matching RDF Graphs. In Proceedings of the International Semantic Web Conference, 2002.
[9] Jeremy J. Carroll. Signing RDF Graphs. In Proceedings of the International Semantic Web Conference, 2003.
[10] Tim Berners-Lee and Dan Connolly. Delta: An Ontology for the Distribution of Differences Between RDF Graphs. http://www.w3.org/DesignIssues/Diff
[11] GNU diff, http://www.gnu.org/software/diffutils/diffutils.html
[12] Sudarshan S. Chawathe, et. al. Change Detection in Hierarchically Structured Information. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1996.
[13] David MacKenzie, Paul Eggert and Richard Stallman. Comparing and Merging Files. http://www.fnal.gov/docs/products/diffutils/diff_toc.html

이 태 휘
정보과학회논문지 : 데이타베이스
제 33 권 제 2 호 참조

김 기 성
정보과학회논문지 : 데이타베이스
제 33 권 제 2 호 참조

유 상 원
정보과학회논문지 : 데이타베이스
제 33 권 제 1 호 참조

김 형 주
정보과학회논문지 : 데이타베이스
제 33 권 제 1 호 참조