http://dx.doi.org/10.5392/JKCA.2015.15.01.009

대규모 RDF 데이터의 특성을 고려한 효율적인 색인 기법

An Efficient Indexing Scheme Considering the Characteristics of Large Scale RDF Data

김기연, 윤종현, 김천중, 임종태, 복경수, 유재수
충북대학교 정보통신공학부

Kiyeon kim(kky@chungbuk.ac.kr), Jonghyeon Yoon(jhyoon13@chungbuk.ac.kr),
Cheonjung Kim(cjkim@chungbuk.ac.kr), Jongtae Lim(jtlim@chungbuk.ac.kr),
Kyoungsoo Bok(ksbok@chungbuk.ac.kr), Jaesoo Yoo(yjs@chungbuk.ac.kr)

요약

본 논문에서는 RDF 데이터 특성을 고려하여 대규모 데이터에 대한 질의 처리를 향상시키기 위한 새로운 색인 기법을 제안한다. 제안하는 기법은 RDF 트리플 중 주어와 술어의 값이 중복적으로 사용되는 특징을 이용하여 주어와 목적어를 S-O 색인으로 구성한다. 또한, 트리플 중 상대적으로 가장 적은 수의 값을 갖고 있는 술어는 별도의 P 색인으로 구성하여 총 색인의 크기를 최소화한다. 술어를 포함한 질의 요청시 크기가 작은 P 색인을 우선 검색하고 술어를 포함하지 않은 질의 요청에 대해서는 S-O 색인을 우선 검색한다. 성능평가를 통해 제안하는 기법이 기존 기법에 비해 질의처리 속도 관점에서 성능이 우수함을 보인다.

■ 중심어 : | 데이터베이스 | 시맨틱 웹 | RDF | 색인 |

Abstract

In this paper, we propose a new RDF index scheme considering the characteristics of large scale RDF data to improve the query processing performance. The proposed index scheme creates a S-O index for subjects and objects since the subjects and objects of RDF triples are used redundantly. In order to reduce the total size of the index, it constructs a P index for the relatively small number of predicates in RDF triples separately. If a query contains the predicate, we first searches the P index since its size is relatively smaller compared to the S-O index. Otherwise, we first searches the S-O index. It is shown through performance evaluation that the proposed scheme outperforms the existing scheme in terms of the query processing time.

■ keyword : | Database | Semantic Web | RDF | Index |

I. 서 론

'00년도 중반부터 개방형 서비스 구조를 기반으로 사용자의 참여를 통해 가치를 창출해내는 웹 2.0(Web 2.0)이 전 세계적으로 활성화되었다. 기존 웹은 사용자들이 데이터와 서비스를 수동적으로 받는 일방적인 정

*본 연구는 미래창조과학부 및 정보통신산업진흥원의 대학IT연구센터육성 지원사업/IT융합고급인력과정지원사업(NIPA-2014-H0301-14-1022), 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발 사업(14-824-09-001, 실시간 대규모 영상 데이터 이해·예측을 위한 고성능 비주얼 디스커버리 플랫폼 개발), 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2013R1A2A2A01015710)
접수일자 : 2014년 11월 11일 심사완료일 : 2014년 12월 05일
수정일자 : 2014년 12월 05일
교신저자 : 유재수, e-mail : yjs@chungbuk.ac.kr

보 제공의 개념이라면 웹 2.0은 참여와 개방을 바탕으로 사용자들이 자유롭게 정보와 네트워크를 활용하는 개념이다. 최근 웹 2.0의 발전과 함께 링크드 데이터의 성장, 정부의 개방형 데이터 서비스, 소셜 웹 서비스의 등장으로 인해 웹의 구조적 데이터는 폭발적으로 성장해 왔다[1][2].

현재의 인터넷과 같은 분산 환경에서 웹 자원에 대한 정보와 자원 사이의 관계나 의미 정보를 컴퓨터가 처리할 수 있는 시맨틱 웹(semantic web) 기반 서비스가 증가하고 있다[3]. 시맨틱 웹 기반 서비스에 대한 요구와 연구가 활발히 진행됨에 따라 웹상의 자원을 표현하기 위해 RDF 데이터 구조가 연구되었다[4-13]. RDF는 데이터 간의 관계를 주어(S : Subject), 술어(P: Predicate), 목적어(O : Object)의 트리플 구조로 표현한다. RDF는 웹을 통해 표현된 값들 사이의 관계를 파악하고 지능적으로 데이터를 처리할 수 있게 한다[14-16].

RDF에 대한 활용이 증가됨에 따라 RDF를 통한 다양한 검색 기법들에 대한 연구들이 진행되고 있다. 이와 함께 RDF 데이터를 빠르게 검색하기 위한 색인 기법에 대한 연구도 활발히 진행되고 있다. 색인은 RDF 데이터에 대한 질의 요청에 대해 전체 데이터를 검색하는 대신 원하는 데이터만을 빠르게 검색하여 질의 처리 성능을 향상시킬 수 있다. 이러한 색인의 대표적인 기법으로는 MAP[9], HexaStore[10], RDF-3X[11], Tridex[12], TripleT[13]와 같은 기법이 제안되었다. 일반적인 RDF 색인 기법들은 RDF에서 사용되는 S, P, O을 결합하여 B+-트리 구조의 색인을 구축한다. MAP 기법은 S, P, O의 결합으로 만들 수 있는 모든 경우를 고려하여 총 6개의 색인을 구축한다. 한 개의 RDF 데이터를 6개의 색인을 만들기 위해 총 6번 중복 저장하여 색인의 크기가 지나치게 증가한다는 단점을 가지고 있다. HexaStore 기법은 6개의 색인을 구축하고 SP, PS와 같이 대칭적인 구조를 갖고 있는 색인은 나머지 O의 값을 공유하여 저장하는 색인 구조를 사용한다. MAP 기법에서 나타나는 단점을 보완하였지만 여전히 6개의 색인을 사용하기 때문에 색인 크기의 증가로 인한 질의 처리 속도의 저하를 해결하지 못하였다. RDF-3X 기법은 MAP 기법과 동일하게 S, P, O을 결합하여 만들 수 있는 모든 경우를 고려한 6개의 색인을 사용한다. RDF-3X 기법에서 다른 점은 앞선 두 가지 기법들과 다르게 빠른 질의 처리를 수행하기 위해 (SP, SO, PO, PS, OS, OP)와 (S, P, O)으로 구분 된 9개의 보조 색인을 사용한다는 점이다. TripleT와 Tridex 기법은 각각 B+-트리 구조로 이루어진 색인을 1, 2개 구축한다. S, P, O을 조합하여 나올 수 있는 모든 경우의 색인을 만드는 다른 기법들과는 다르게 이 두 가지 기법은 적하 버켓(payload bucket)이라는 공간을 활용하여 색인의 크기를 줄이려는 노력을 하였다. 기존 기법들은 검색의 효율성만을 고려하기 때문에 데이터를 중복적으로 저장하거나 중복 저장을 제거하기 위해 한 개의 색인을 이용하여 불필요하게 색인의 높이가 증가되어 검색 효율이 저하되는 문제점들이 존재한다. 또한 여러 개로 나누어 사용하는 색인은 질의처리를 수행할 때도 단점을 나타낸다. 사용자의 요청에 따른 질의처리를 수행할 때 질의 유형에 따라 색인들의 조인연산을 한다. 이 때 색인의 크기가 크면 클수록 조인 연산의 비용과 시간이 기하급수적으로 늘어나기 때문에 방대한 양의 RDF 트리플을 처리하기에 적합하지 않다.

본 논문에서는 대규모 RDF 데이터를 효율적으로 처리하기 위한 새로운 색인 기법을 제안한다. 제안하는 기법은 RDF 트리플 데이터의 크기가 증가할수록 효율이 저하되는 기존 기법의 문제점을 해결하기 위해 색인의 크기를 감소시킨다. 제안하는 기법은 RDF 트리플 중 S와 O의 값이 중복적으로 사용되는 특징을 이용하여 S와 O를 하나의 B+-트리 색인으로 구축하고 P를 이용한 검색을 지원하기 위해 P를 이용한 별도의 B+-트리 색인을 생성한다. S와 O의 중복되는 값과 색인의 크기를 감소시키는 방법을 통해 질의 처리 속도를 개선할 수 있다. 질의 처리는 우선 트리플 데이터를 사전을 통해 ID로 변경하여 구성된 색인을 통해 검색을 하게 된다. 두 개의 색인 중 하나를 택하여 질의에서 제공된 값을 찾게 된 후 각 색인의 단말노드에서 연결된 값을 통해 질의에서 원하는 결과 값을 반환해 줄 수 있다. 제안하는 기법은 질의를 처리 시 두 개의 색인을 사용할 필요가 있는 경우 크기가 작은 P 색인을 우선 검색한 후 S와 O로 구성된 색인을 검색함으로서 전체적인 질

의처리 속도를 향상시킬 수 있다.

본 논문은 다음과 같이 구성된다. 2절에서는 RDF 데이터를 처리하기 위해 연구되었던 기존의 기법들에 대해 설명한다. 3절에서는 대용량 RDF 데이터를 효율적으로 처리하기 위해 제안한 새로운 색인 기법을 기술한다. 4절에서는 제안하는 색인 기법과 기존 색인 기법의 성능평가를 통해 제안하는 기법의 우수성을 보인다. 마지막 5절에서는 본 논문의 결과와 제안하는 기법의 향후 연구 방향에 대하여 소개한다.

II. 관련 연구

최근 RDF 데이터를 빠르게 검색하고 효율적으로 관리하기 위한 색인 기법에 대한 연구가 활발히 진행되고 있다. MAP 기법은 RDF 데이터의 검색 효율을 높이기 위하여 B+-트리로 이루어진 <SPO, SOP, PSO, POS, OSP, OPS> 총 6개의 색인을 사용한다. RDF 트리플 S, P, O의 각 역할을 정렬하여 만들어 낼 수 있는 모든 경우를 고려하여 만들어진 색인을 통해 어떠한 질의가 요청되는 경우에도 요구되는 트리플 값을 확인하고 색인을 선택하여 결과 값을 반환해 줄 수 있다. 이 색인 구조 기법의 장점은 모든 질의를 처리 할 수 있는 색인을 구성하였다는 것이지만, 6개의 색인을 구성할 때 중복되어 저장되는 데이터가 발생된다는 단점을 갖고 있다[9]. 데이터의 크기가 최대 6배로 증가되는 점은 빅데이터 시대로 불리는 만큼 크기가 큰 데이터를 사용하여 색인을 구성하게 될 때 결과적으로 질의처리 속도가 저하되는 단점을 갖게 된다.

HexaStore 기법은 MAP 기법과 동일하게 B+-트리로 이루어진 6개의 색인으로 구성된다. 하지만 HexaStore 기법은 색인의 크기를 감소시켜 저장공간을 효율적으로 사용하기 위하여 <SO, OS, SP, PS, OP, PO>로 이루어진 6개의 색인을 사용한다. HexaStore 기법은 색인을 구성할 때 SP, PS와 같이 대칭적인 구조를 갖고 있는 색인은 S, P, O 중 색인을 생성하는데 포함되지 않은 하나의 값을 공유하여 저장한다. 이러한 방법을 통해 MAP 기법과 같이 한 개의 트리플 데이터가 최대 6번 중복 저장 될 수 있다는 단점을 보완하는 노력을 하였다[10]. 하지만 이 기법 또한 여전히 데이터가 중복되어 저장되기 때문에 저장 공간이 많이 소요되며 RDF 데이터 크기가 증가 할수록 색인의 높이가 증가되어 검색 성능이 많이 저하되는 단점을 갖고 있다.

RDF-3X 기법은 MAP 기법과 동일한 S, P, O의 결합으로 이루어진 6가지 색인을 기본적으로 사용한다. 또한 어떠한 목적어의 저자가 몇 명인지를 묻는 질의 같은 경우는 트리플의 모든 속성 값을 필요로 하지 않기 때문에 <SO, OS, SP, PS, OP, PO>와 같이 두 가지 속성으로 이루어진 보조의 색인을 별도로 구성하여 사용한다. 이 기법은 각각의 RDF 값에 ID를 부여하는 방식으로 색인의 크기를 줄이기 위한 노력과 질의 유형에 따라 색인을 선택적으로 사용하는 방법을 통해 질의처리의 속도를 개선하였다[11]. 하지만 앞서 소개된 기법들과 마찬가지로 RDF 트리플 속성들의 순서만 바뀐 색인을 구성하면서 RDF 데이터 값의 중복으로 인해 색인의 크기가 증가하는 단점은 해결하지 못하였다. 또한 두 가지 기법 모두 질의의 형태에 따라 다수 색인을 검색하여 많은 조인 연산을 수행하기 때문에 복잡한 질의처리에 비효율적이다.

Tridex 기법은 앞서 설명된 세 가지 기법과 동일하게 B+-트리를 사용하며 각각 S, P, O를 기반으로 3개의 색인을 사용한다. Tridex 기법의 특징은 적하 버켓이라는 공간을 사용하여 색인을 구성하는데 사용되지 않은 두 개의 키값을 저장한다. 예를 들어 트리플 데이터 중 S를 키값으로 사용하여 구성된 S색인의 단말 노드에 저장된 키값에는 트리플 데이터에서 S와 연결되어 있는 O와 P의 값이 쌍으로 적하 버켓에 저장되어 단말 노드와 연결이 되어 있다. 이러한 저장방법을 이용하여 S의 값을 이용하게 되는 질의를 처리할 때 단말 노드까지 이동하고 검색된 S키값과 연결된 적하 버켓에 저장되어 있는 O와 P의 값을 반환해 줄 수 있다[12]. Tridex의 기법은 적하 버켓이라는 공간을 사용하여 색인의 크기를 감소시켰으나 버켓 공간에는 여전히 많은 중복이 발생하여 버켓의 크기가 증가되는 문제점이 있다.

TripleT 기법은 데이터 중복을 최소화하기 위하여 한 개의 색인만을 사용한다. B+-트리로 이루어진 SPO 색

표 1. RDF 색인 연구 기법의 특징

	색인 구성 방법	색인의 종류	버켓 사용 유무	색인의 특징
MAP	B+-트리	SPO, SOP, PSO, POS, OSP, OPS	미사용	RDF 트리플 데이터 하나당 6번 중복 저장
HexaStore	B+-트리	SP, PS, SO, OS, PO, OP	미사용	대칭 구조를 갖는 색인이 나머지 값을 공유
RDF-3X	B+-트리	SPO, SOP, PSO, POS, OSP, OPS	미사용	사전 테이블과 압축 기법을 통해 색인의 크기를 감소
Tridex	B+-트리	S, P, O	사용	적하 버켓을 사용하여 트리플을 연결
TripleT	B+-트리	SPO	사용	하나의 색인에 모든 값을 키값으로 사용

인에는 트리플 데이터에서 S, P, O의 역할에 관계없이 중복되는 값들은 저장하지 않으며 모든 값들을 색인의 키값으로 사용한다. TripleT 색인은 적하 버켓을 사용하여 B+-트리 하나의 키값에 해당하는 3개의 버켓 공간을 활용한다. 버켓 공간에는 각 키값과 관련된 나머지 값들을 저장한다. 예를 들어 'id1'이라는 주어 값이 저장되어 있는 노드에는 주어 버켓(subject bucket), 술어 버켓(predicate bucket), 목적어 버켓(object bucket) 세 개의 버켓이 연결되어 있고, 주어 버켓에는 트리플 데이터에서 'id1'과 연결되어 있는 모든 값을 <술어, 목적어> 쌍으로 저장한다. TripleT 기법은 한 개의 색인을 사용하기 때문에 저장 공간이나 관리 측면에서 효율적이다[13]. 하지만 모든 키값이 하나의 색인에 저장되기 때문에 색인의 크기가 매우 증가하여 데이터가 증가할수록 색인의 높이가 증가하고 결과적으로 검색 시간이 증가되는 단점이 있다.

[표 1]은 앞서 설명된 기존 색인 기법 관련 연구들의 특징을 요약 비교 분석하여 정리한 표이다. 표는 색인의 구성방법과 색인의 종류, 버켓 사용 유무, 색인의 특징을 정리한다. 분석된 표를 통해 알 수 있듯이 주어, 술어, 목적어로 조합할 수 있는 모든 색인을 구성하는 방식과는 달리 버켓을 사용하여 색인을 제안하는 방식은 색인의 종류가 적은 것을 알 수 있다.

Ⅲ. 제안하는 RDF 데이터 색인 기법

1. 개요

본 논문에서는 RDF 트리플 데이터에서 사용자의 질의를 정확하고 빠르게 처리하기 위한 색인 기법을 제안한다. 주어, 술어, 목적어로 구성된 RDF 트리플 데이터는 술어를 통해 주어와 목적어의 관계가 표현된다. 이러한 RDF 트리플 데이터로 표현되는 웹 데이터의 양이 시맨틱 웹의 발전과 함께 빠른 속도로 증가함에 따라, RDF를 효율적으로 저장하고, 사용자가 요청하는 질의 처리를 빠르게 검색할 수 있는 색인 구조의 필요성이 높아지고 있다. RDF 트리플 데이터의 관계를 이용한 기존 색인 기법들이 많이 연구되었지만, 방대한 빅데이터 처리시 색인의 크기도 크게 증가하여 질의 처리 성능이 현저하게 떨어지는 문제점이 있다. 제안하는 색인 기법은 빅데이터 처리시 기존 색인 기법이 갖고 있는 문제점을 해결하기 위해 RDF의 특성을 고려하여 두 개의 B+-트리 색인을 구축하여 질의를 효율적으로 처리한다.

RDF 트리플 데이터의 특징은 주어와 목적어를 나타내는 S와 O의 값의 중복이 매우 빈번하게 나타난다는 점이다. 예를 들어, facebook, 트위터 등의 SNS에서 서로 연결 되어 있는 사람들을 관리하는 RDF 데이터가 있다고 가정 할 때 'id1'이라는 키 값은 주어로 사용되어 술어를 통해 'id2'(목적어)와 연결 될 수 있지만, 또한 'id3'를 사용하는 주어의 목적어로 사용 될 수 있다. RDF 트리플에서 주어와 목적어의 관계를 서술해 주는 술어의 양은 자체적으로 중복이 심하고 데이터의 양이 적지만, 주어와 목적어는 값이 매우 방대하고 서로 중복되는 값을 상당수 갖고 있다. 이러한 RDF 트리플의 특성을 이용하여 색인의 크기를 감소시키기 위하여 주어와 목적어를 하나의 색인(S-O 색인)으로 구성한다. 또한, 트리플 중 상대적으로 가장 적은 수의 값을 갖고

표 2. 트리플 테이블 집합

Dataset	#Triples	#S	#O	#(S∩O)	#P
LUBM 10M	13,879,970	2,181,772	1,623,318	501,365	18
LUBM 50M	69,099,760	10,857,180	8,072,359	2,490,221	18
LUBM 100M	138,318,414	21,735,127	16,156,825	4,986,781	18
LUBM 500M	691,085,836	108,598,613	80,715,573	24,897,405	18
LUBM 1B	1,335,081,176	217,206,844	161,413,041	49,799,142	18
UniProt	2,954,208,208	543,722,436	387,880,076	312,418,311	112

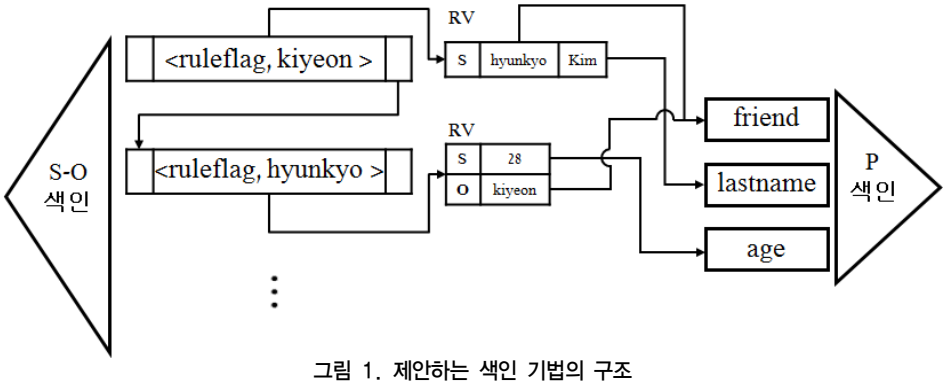


그림 1. 제안하는 색인 기법의 구조

있는 술어는 별도의 색인(P 색인)으로 구성한다.

[표 2]은 앞서 설명된 RDF 트리플 데이터의 특성을 나타내주고 있는 실제 RDF 데이터 집합을 보여준다 [17][18]. 표에서 보는 것과 같이 주어와 목적어를 표현하는 S와 O 데이터의 수가 가장 많고 이 둘의 관계를 서술해 주는 P의 값은 상당히 적은 양의 값이 있는 특성을 확인할 수 있다. 또한 S와 O 데이터 값의 교집합을 나타내는 (S∩O)값의 수를 통하여 본 논문에서 제안하는 기법이 고려한 RDF 트리플 데이터의 S-O간 중복이 많이 발생함을 확인할 수 있다.

제안하는 기법은 B+-트리로 구축된 두 개의 색인 <S-O 색인, P 색인>으로 구성된다. [그림 1]은 RDF 트리플 데이터의 특성을 고려한 제안하는 기법의 색인 구조를 나타낸다. S-O 색인은 트리플에서 S와 O가 각각 매핑된 값을 키 값으로 이용하여 B+-트리로 구성된다. 이때, 색인의 크기를 감소시키기 위해 S와 O가 중복적으로 사용되는 값은 한번 저장하며 각각의 값들은 ID 값으로 저장이 된다. 또한, 실제 저장되는 값은 RDF 트리플 데이터의 실제 값이 아닌 정수 값 ID가 저장되기 때문에 색인의 크기를 감소시킬 수 있다. S-O색인의 단말노드에는 RV 버켓의 공간을 두어 색인의 단말노드에 저장된 ID값의 실제 데이터가 트리플 테이블에서 연결되어 있는 나머지 값을 저장하고 P색인의 단말노드와 연결한다. P 색인 또한 사전 테이블을 통해 매핑된 ID 값을 키값으로 사용한 B+트리의 형태로 구성된다. S-O 색인에 비해 P 색인은 키값이 매우 적기 때문에 색인의 크기가 크지 않다. 이러한 이유로 S-O 색인과 P 색인을 선택적으로 사용할 수 있는 질의처리 시 제안하는 기법에서는 크기가 작아 검색속도가 빠른 P 색인을 우선적으로 사용하여 질의처리의 효율을 극대화 시킨다.

2. 사전 테이블

색인을 구성하기 전에 먼저 색인의 크기를 감소시키기 위해 각 S, P, O의 값을 고유의 ID 값으로 매핑하는 사전 테이블을 작성한다. ID 값은 저장되는 값에 순차

적으로 번호를 매핑해 주는 방식으로 한다. [그림 2]는 RDF 일부를 고유의 ID 값으로 매핑한 결과를 나타낸다. 제안하는 색인 기법에서는 실제 값(문자열 형태)이 아닌 할당된 ID 값을 정수 값으로 저장함으로서 색인의 크기를 감소시킬 수 있다. RDF 데이터의 특성상 S와 O의 값은 서로 중복되어 사용되기 때문에 S와 O를 하나의 색인으로 구성하여 전체적인 색인의 크기를 감소시킨다. 'hyunkyo'와 'jonghyun' 값은 주어와 목적어 두 가지 값으로 모두 사용 되었지만, S-O 색인을 구성하기 위한 매핑 단계에서 중복되는 값은 한번만 ID 값을 할당해 주게 된다.

S	O
kiyeon	hyunkyo
hyunkyo	jonghyun
kiyeon	Kim
sooyong	Yoon
hyunkyo	28
jonghyun	28

ID	Value
0	kiyeon
1	hyunkyo
2	jonghyun
3	Kim
4	sooyong
5	Yoon
6	28

그림 2. S, O 값의 ID 매핑

색인 구성을 위해 [그림 2]와 같이 중복되는 값은 제외하고 주어와 목적어의 값에 따라 순차적으로 ID 값을 할당한다. [그림 3]은 P 색인을 만들기 위해 P의 값을 ID 값으로 매핑하는 사전 테이블이다. P 사전 테이블 또한 S와 O와 같이 실제 값을 각각 순차적으로 할당되는 ID 값으로 저장하기 때문에 만들어지는 색인의 크기를 줄일 수 있다. P 사전은 P 값을 매핑하는 용도로 사용하기 때문에 RDF 트리플 데이터에서 중복적으로 나타나는 P 값은 제외하고 ID 값을 할당한다. 예를 들어 [그림 3]에 나타나는 RDF 데이터 중 P의 값 name이 처음 '0'의 ID 값을 할당 받게 된 이후에 나오는 4번 째 name은 ID 값을 별도로 할당하지 않는다.

P
name
age
gender
name
friend
friend
hasAuthor
publisher
publisher

ID	Value
0	name
1	age
2	gender
3	friend
4	hasAuthor
5	publisher

그림 3. P 값의 ID 매핑

3. S-O 색인

제안하는 기법의 S-O 색인은 사용자로부터 <s ?p o>, <?s ?p o>, <s ?p ?o> 유형의 질의를 받았을 때 질의처리를 수행하기 위해 사용된다. 기존 색인 기법의 구조와 다른 점은 RDF 트리플 데이터를 이용해 색인을 구성할 때, 주어와 목적어의 값이 대다수 중복되어 사용된다는 특징을 이용하였다는 점이다. S-O 색인은 ruleflag 값을 이용하여 저장 되어 있는 각 키값이 트리플에서 어떠한 역할로 이용 되었었는지 파악 할 수 있기 때문에 S, O의 중복 값을 제외하여 저장하기 때문에 색인의 크기를 감소시키고 결과적으로 질의처리를 할 때 속도 향상의 효과를 낼 수 있다. [그림 4]는 [그림 2]에서 나타난 S-O 사전을 통해 매핑된 결과를 사용하여 구성된 제안하는 기법의 S-O 색인 구조를 나타낸다.

[그림 5]는 제안하는 S-O 색인의 단말노드 구조를 나타낸다. 색인의 단말노드는 <ruleflag, ID, RVptr>로 저장되어 각각이 RV 버켓과 연결된 형태로 사용된다. ruleflag는 저장되는 값이 실제 트리플에서 어떠한 역할로 사용되었는지를 나타낸다. ruleflag는 해당 키값이 S의 역할로 사용되었을 때는 '01', O의 역할로 사용되었을 때는 '10', 그리고 S와 O의 역할에 모두 사용되었을 때는 '11'으로 값을 표기한다. 예를 들어 [그림 3]에서 값'kiyeon'은 주어만으로 사용 되었기 때문에 '01'로 표기를 하고 값, '28'은 목적어만으로 사용되었기 때문에 '10'이고, 'hyunkyo','jonghyun' 은 주어와 목적어의 두 가지 역할을 모두 하고 있으므로 '11'으로 값을 표기한다.

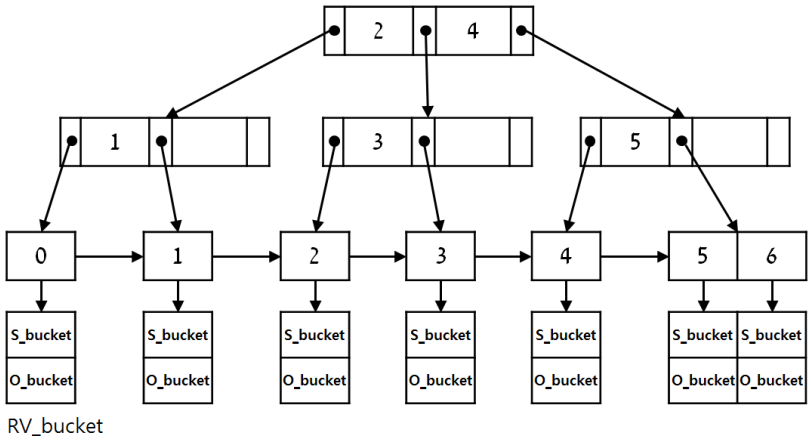


그림 4. S-O 색인

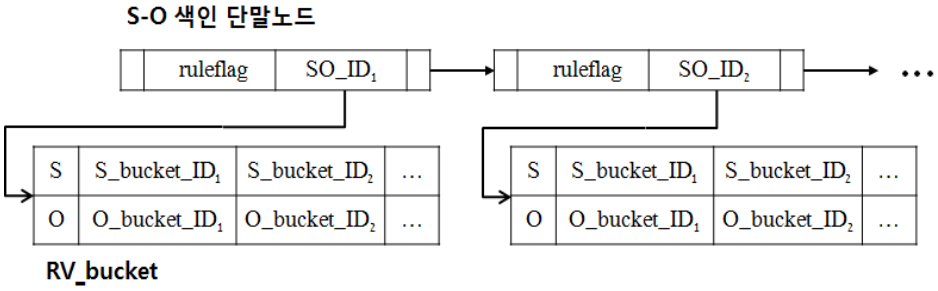


그림 5. S-O 색인 단말 노드의 구조

ID는 S, O가 사전에 중복을 피해 매핑 된 값을 나타내며, 실제 데이터의 형태가 아닌 순차적으로 부여된 ID 값을 저장하는 방법을 통해 전체 색인의 크기를 감소시킨다. RVptr 값은 RV 버켓과 연결하기 위한 주소를 저장한다. RV(Related Value) 버켓은 노드에 저장하는 값과 트리플에서 관련되었던 값들의 ID를 저장한다. 특히 노드의 키값이 트리플에서 S와 O의 역할로 모두 사용되었을 경우는 RV 버켓에 각각의 역할과 관련된 값들의 ID값을 저장한다. 예를 들어 'kiyeon'의 ID를 담고 있는 단말노드는 실제 RDF 트리플 데이터에서 'hyunkyo'와 'Kim'이라는 목적어 값과 연결되어 있기 때문에 RV 버켓에는 <hyunkyo, jonghyun>이 저장이 된다. 또한 값 'hyunkyo' 는 RDF 트리플 데이터에서 주어와 목적어의 두 가지 역할로 모두 사용되었기 때문에 RV 버켓에는 값이 주어로 사용 되었을 때 연결된 목적어를 저장하는 버켓과 목적어로 사용되었을 때 연결된 주어를 저장하는 버켓 두 개의 공간이 필요하다. 이때 각각 주어 버켓(subject bucket)에는 <28>, 목적어 버켓(object bucket)에는 <kiyeon>이 저장된다. 이렇게 RV 버켓에 저장되는 값들은 P 색인에 저장하는 P값의 주소와 포인터로 연결하게 된다. 이러한 단말 노드 구조를 통해, 트리플 중 S 또는 O의 값을 요구하는 질의를 처리 할 때 노드에 표기된 ruleflag 값을 검색에 이용하여 S-O 색인 전체를 검색하지 않고 질의를 빠르게 처리 할 수 있다. 또한 노드의 RV 버켓에 저장된 값을 이용하여 트리플에서 연관되어 있는 S-O의 값을 직관적으로 사용할 수 있다.

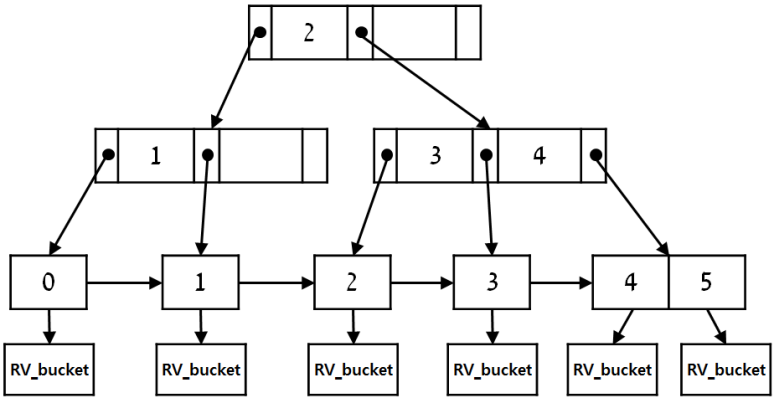


그림 6. P 색인

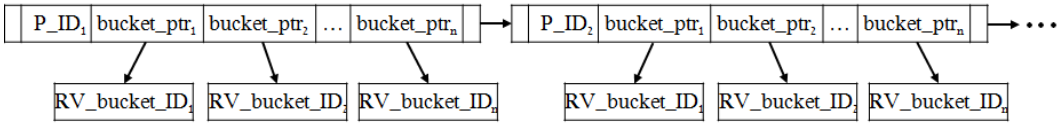


그림 7. P 색인의 단말 노드의 구조

4. P 색인

제안하는 기법의 S-O 색인은 사용자로부터 <?s p o>, <s p ?o>, <?s p ?o> 유형의 질의를 받았을 때 질의처리를 수행하기 위해 사용된다. [그림 6]은 앞서 제안된 [그림 3]의 P 색인 테이블을 통해 ID 값으로 변환된 P 값을 사용하여 구성한 P 색인의 구조를 나타낸다.

B+-트리로 구성된 P 색인의 특징은 S-O 색인에 비해 그 크기가 매우 작다는 점이다. RDF 데이터에서 주어와 목적어가 갖고 있는 값의 개수 보다 훨씬 적은 양의 값을 갖고 있기 때문에 색인을 구성할 때 그 크기가 매우 작게 된다. 따라서 사용자의 요청에 의해 질의처리를 할 때 S-O 색인과 P 색인을 모두 사용할 수 있는 질의라면, P 색인을 우선 검색하고 P 색인을 통해 나온 1차 결과를 이용하여 처리하는 것이 질의처리의 속도를 향상시키는 방법이다.

[그림 7]은 P 색인의 단말노드 구조를 나타낸다. P 색인의 단말노드는 P 사전을 통해 매핑된 술어의 ID 값과 그 ID와 연결된 S-O 색인의 RV 버켓의 주소를 나타내는 포인터를 저장하는 공간으로 되어 있다. S-O 색인의 단말노드와 다른 점은 P 색인의 단말 노드는 별도의 RV 버켓 공간을 갖고 있는 것이 아니라 각 단말 노드에 P값이 키값으로 저장되며 연결되어 있는 S, O의 값을 포함한 S-O 색인의 RV 버켓의 주소를 저장한다.

5. 질의 처리

제안하는 기법에서는 <?s p o>, <s p ?o>, <s ?p o>, <?s ?p o>, <s ?p ?o>, <?s p ?o>의 여섯 가지 질의 패턴을 고려한다. <?s ?p ?o>의 형태는 색인 전체를 검색해야 하므로 별도로 처리 과정을 설명하지 않는다. 제안하는 기법의 질의처리 과정은 크게 3단계로 나뉜다. 우선 사용자로부터 입력되는 질의의 유형을 파악한다. 질의의 유형을 파악한 후에 색인의 효율성을 검사하여 제안하는 기법에서 구성하는 S-O 색인과 P 색인 중 질의처리에 유리한 색인을 결정하게 된다. 그 후 결정된 각 색인을 통해 질의를 처리하여 사용자에게 원하는 값을 반환해 주는 방식이다.

제안하는 색인 구조를 사용하여 질의를 처리할 때는 질의의 유형을 파악하여 S-O 색인을 사용할지 P 색인을 사용할 것인가를 결정한다. 또한 S-O 색인과 P 색인 둘 중 어느 것을 사용하더라도 질의에 대한 답을 얻을 수 있는 경우는 색인의 크기가 작기 때문에 검색 시간이 적은 P 색인을 우선적으로 사용하여 질의를 처리한다. 색인을 결정할 때는 질의 트리플 중 술어의 값을 모르는 질의의 경우 <s, ?p, o>, <?s, ?p, o>, <s, ?p, ?o>는 S-O 색인을 사용하고, 주어 또는 목적어의 값을 모르는 질의의 유형 <?s, p, o>, <s, p, ?o>, <?s, p, ?o>는 P 색인을 사용한다. [그림 8]은 제안하는 기법을 사용할 때 색인을 결정하는 질의처리 알고리즘을 나타낸다.

```
01:  funtion Query_type {
02:    query; /* 사용자의 질의 유형 저장 */
03:    answer; /* 질의 처리 결과 */
04:
05:    switch (query) {
06:      case ( s ?p o ) :
07:      case ( ?s ?p o ) :
08:      case ( s ?p ?o ) :
09:        answer = SO_index(query);
10:        /* SO 색인 사용 */
11:        break;
12:      case ( ?s p o ) :
13:      case ( s p ?o ) :
14:      case ( ?s p ?o ) :
15:        answer = P_index(query);
16:        /* P 색인 사용 */
17:        break;
18:    }
19:
20:    return answer; /* 질의 처리 결과 반환 */
21:  }
```

그림 8. 색인을 결정하는 알고리즘

질의가 입력되면 질의 유형을 파악하여 질의처리 속도에 유리한 색인을 결정한다. 이때, 주어나 목적어를 사용하는 질의처리는 동일한 색인을 사용하는 것이기 때문에 S-O 색인을 사용할 것인가 P 색인을 사용할 것인가를 결정하는 것과 같다. S-O 색인을 사용하여 질의처리를 수행 할 때는 우선 사용하는 값이 주어인지 목적어인지 파악하게 된다. 사용하는 값이 주어라면 목적어의 값과 술어의 값을 반환해주게 되고 사용하는 값이 목적어라면 주어와 술어의 값을 반환해주게 된다. P 색인을 통한 질의처리 과정은 우선 술어 의외에 알고 있는 값이 있는지 검사하게 된다. 주어를 알고 있다면 색인의 말단노드에서 연결된 주어의 값을 파악하고 목적어의 값을 반환해주게 된다. 또한 목적어를 알고 있다면 말단노드에 연결된 목적어의 값을 사용하여 주어의 값을 반환해준다. 하지만 술어의 값만을 알고 주어와 목적어를 알고 있지 않다면, 색인의 말단노드에 연결된 RV_bucket에 저장된 ID 값과 그와 연결된 S-O 색인의 말단노드에 저장된 ID 값을 주어, 목적어 쌍으로 반환해주게 된다.

S-O 색인을 사용하여 질의처리를 하기 위해서는 우선 질의에서 얻는 주어 또는 목적어의 값을 S-O 사전을 통해 ID를 확인하여야 한다. 예를 들어 <kiyeon ?p hyunkyo> 질의를 처리하기 위해서는 'kieyon' 또는 'hyunkyo'를 사용하여 S-O 색인을 검색하게 된다.'kieyon' 값을 통해 질의처리를 할 때는 S-O 사전을 통해 'kiyeon'에 대한 ID 값이 '3'이라는 것을 확인 할 수 있다. 이 ID 값을 S-O 색인에서 검색하여 단말노드까지 도달하게 된다. 도달한 단말노드에서 RV bucket에 있는 목적어 값을 확인하여 그 값이 'hyunkyo'가 있는 것을 알 수 있다. 질의에서 주어진 목적어 값과도 일치하는 이 값을 확인하고 이와 연결된 P 색인의 ID값을 확인한다. 확인된 ID값을 이용하여 P 사전을 통해 실제 value 값으로 변환하고 질의에 대한 최종적인 응답으로 'friend '값을 반환해주게 된다. [그림 9]는 S-O 색인을 사용할 때의 질의처리 알고리즘을 나타낸다.

```
     function SO_index (query) {
01:    subject_ID = SO_dictionary(subject);
02:    /* ID로 변환 */
03:    object_ID = SO_dictionary(object);
04:    /* ID로 변환 */
05:
06:    if(subject_ID) {
07:    /* subject 값을 이용한 검색 */
08:      for every value of RV_bucket {
09:        O ← RV.object_ID[i];
10:        P ← RV.object_ID[i].predicate_ID;
11:      }  /* subject와 연결된 모든 RV값과
12:                    predicate값 저장 */
13:    value ← SO_dictionary(O),
14:              P_dictionary(P);
15:    }
16:
17:    if(object_ID) {
18:    /* object 값을 이용한 검색 */
19:      for every value of RV_bucket {
```

```
20:
21:         S ← RV.subject_ID[i];
22:         P ← RV.subject_ID[i].predicate_ID;
23:       }  /* object와 연결된 모든 RV값과
24:                    predicate 값 저장 */
25:       value ← SO_dictionary(O),
26:                 P_dictionary(P);
27:     }
28:     return value; /* 질의 결과 반환 */
29:   }
```

그림 9. S-O 색인을 사용한 질의처리 알고리즘

P 색인을 사용하여 질의처리를 하기 위해서는 우선 질의에서 얻는 P값을 P 사전을 통해 ID를 확인한다. [그림 10]은 P 색인을 사용할 때의 질의처리 알고리즘을 나타낸다. 알고리즘을 처리하는 방법은 예를 들어 <?s firend hyunkyo> 질의를 처리하기 위해서 'friend' 값을 사용하여 P 색인을 검색하게 된다. 이 때 또한 S-O 색인을 사용한 처리와 비슷하게 P 사전을 통해 'friend'에 대한 ID 값이 '0'이라는 것을 알아낼 수 있다. 이 ID 값을 P 색인에서 검색하여 단말노드까지 검색하게 된다. 도달한 단말노드에 연결된 S-O 색인의 RV bucket에 있는 목적어 값을 확인하여 값이 'hyunkyo'일 때 이 RV bucket와 연결된 S-O 색인의 단말노드에 표기된 ID를 확인한다면 질의에서 원하는 주어의 값을 반환해 줄 수 있게 된다. 마찬가지의 순서로 확인된 ID를 S-O 사전을 통해 value 값으로 변환해주게 되고 최종적으로 질의에 대한 응답으로 'kiyeon'을 반환할 수 있다.

여섯 가지 질의 패턴 중 <?s p o>와 <s p ?o>, <?s ?p o>와 <s ?p ?o>의 형태는 S-O 색인을 사용하여 처리하는 방법이 주어와 목적어의 차이점이 있지만 동일한 방법의 순서대로 처리하기 때문에 같은 질의 형태로 간주하여 네 가지 질의 패턴의 처리 과정을 설명한다.

<?s p o> 질의가 요청 되는 경우에는 목적어를 이용하여 S-O 색인을 검색하는 방법보다는 우선 크기가 작은 P 색인을 검색한다. P 색인을 통해 해당되는 술어 값의 검색을 완료하면 이 값과 연결되어 있는 목적어의 노드를 검색한다. 주어 값으로는 목적어의 노드가 포함하고 있는 모든 RV 버켓에 저장된 값을 반환한다.

<s ?p o> 질의가 요청 되는 경우에는 ruleflag 값을 이용해 주어 또는 목적어를 검색하게 된다. 주어의 값을 검색하여 질의를 처리 하는 경우 '01' 또는 '11'으로 표기 되어 있는 노드를 검색하여 RV 버켓 값에 질의에 제시된 목적어가 있는지 검색하게 된다. 검색된 노드가 목적어를 포함하고 있지 않다면 후보에서 제외된다. 남아있는 노드들과 연결되어 있는 술어 값을 모두 반환하면 질의에서 제공한 주어와 목적어의 값과 연관되어 있는 모든 술어 값을 검색할 수 있다.

```
01:
02: function P_index (query) {
03:   predicate_ID = P_dictionary(predicate);
04:   /* ID로 변환 */
05:
06:   if( when we know the subject ) {
07:   /* 주어를 알고 있을 때 */
08:     subject_ID = SO_dictionary(subject);
09:     /*ID로 변환*/
10:     value = leafnode_ID;
11:     /* 단말노드에 연결된 RV_bucket를 갖고
12:        있는 S-O 색인의 leaf_node ID값 저장 */
13:
14:     return SO_dictionary(value);
15:     /* 주어 값 반환 */
16:    }
17:
18:   else if( when we know the object ) {
19:   /* 목적어를 알고 있을 때 */
20:     object_ID = object_dictionary(O);
21:     /* ID로 변환 */
22:     value = leafnode_ID;
23:     /* 단말노드에 연결된 RV_bucket를 갖고
24:      있는 S-O 색인의 leaf_node ID값 저장 */
25:     return SO_dictionary(value);
26:     /* 목적어 값 반환 */
27:    }
28:
29:   else if(when we don't know both subject
30:                                and object ){
31:   /* 주어와 목적어를 모두 모를 때 */
32:     for every value of RV_bucket is linked
33:                                  P leafnode {
34:     /* 서술어와 연결된 모든
35:                          RV_bucket 값 검색 */
36:
37:     value ← RV_bucket_ID, leafnode_ID;
38:     /* RV_bucket에 저장된 ID와
39:        RV_bucket를 갖고 있는 S-O 색인의
40:                    leaf_node ID값 저장 */
41:
42:     return SO_dictionary(value);
43:     /* 주어, 목적어 값 반환 */
44:    }
45: }
```

그림 10. P 색인을 사용한 질의처리 알고리즘

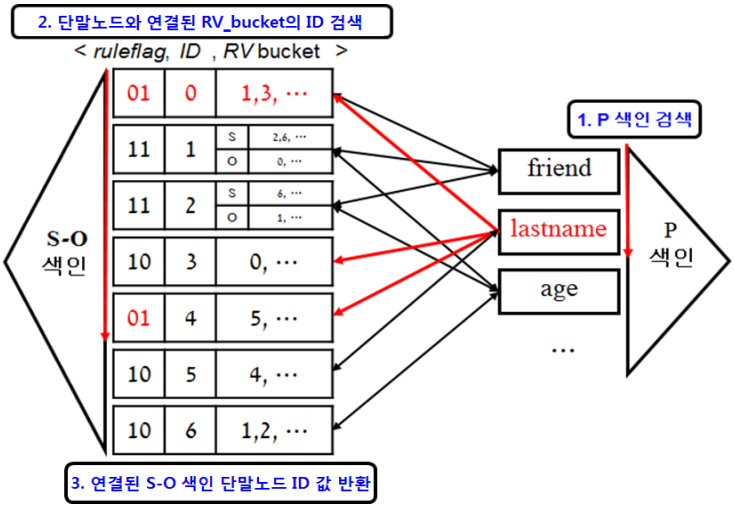


그림 11. 〈?s p o〉 질의 처리 과정

<?s ?p o> 형태의 질의가 요청되는 경우에는 우선 S-O 색인에서 ruleflag 값이 '10' 또는 '11'으로 표기 되어 있는 노드를 검색한다. '10'으로 표기된 노드는 유지하고 있는 RV의 공간 안에 있는 모든 값과 연결된 P의 쌍을 반환한다. '11'으로 표기된 노드는 RV에 저장 되어 있는 값 중 노드에 저장된 값이 목적어로 사용된 값들과 연결된 술어의 쌍을 결과로 반환한다.

<?s p ?o> 형태의 질의가 요청되는 경우에는 P 색인을 검색하여 술어 값이 저장되어 있는 노드를 검색한다. 검색이 완료되면 그 노드와 연결되어 있는 모든 S-O 색인의 노드를 검색하여 ruleflag 값과 RV 버켓에 저장되어 있는 값들을 참고하여 S-O 쌍으로 결과를 반환한다.

[그림 11]는 제안하는 기법의 색인 구조를 간단히 표현하여 실제로 <?s p o> 질의를 처리하는 과정을 나타낸다. 예를 들어, 성(lastname)이 'Kim'인 사람의 이름을 묻는 질의를 처리하는 과정은 S-O 색인과 P 색인을 선택하여서 처리 할 수 있기 때문에 색인의 크기가 작은 P 색인을 우선적으로 선택하여 검색을 실행한다. P 색인을 검색하여 lastname에 해당하는 ID 값을 갖고 있는 단말노드에 접근한 후 이와 연결된 RV 버켓을 활용하여 Kim의 ID를 찾게 된 후 이와 연결된 S-O 색인의 노드를 검색한다. 이때, ruleflag와 ID 값을 활용하여 전체 색인을 검색 하지 않고 해당되는 노드를 찾아 낼 수 있으며 검색된 단말노드에 저장 되어 있는 ID 값을 확인해 ID 값을 변환한다면 'kiyeon'의 결과를 얻을 수 있다. 또한 성이 'Kim'인 사람의 친구를 검색하는 질의는 두 개의 트리플 조인 유형 <?s1 lastname Kim · ?s1 friend ?o>으로 검색할 수 있다. 이 질의는 앞서 설명한 예를 통해 나온 결과를 같은 방법으로 다시 사용하여 질의를 처리하면 <hyunkyo, jonghyun>의 결과를 얻을 수 있다.

Ⅳ. 성능 평가

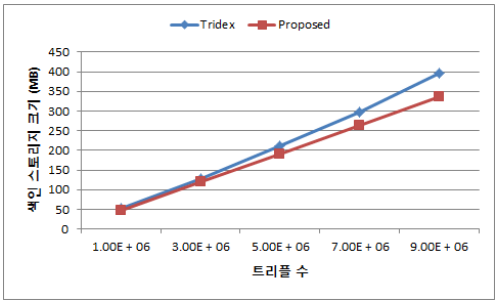
제안하는 기법의 우수성을 입증하기 위해 기존 기법과의 성능 평가를 수행하였다. 성능평가는 Intel Core i5-3570 CPU 3.40GHz, 8GM 램, 64 bit Window 7의 환경에서 수행하였다. 본 성능 평가는 1,000,000개의 RDF 트리플을 사용하여 제안하는 기법과 기존 기법을 구축한 후 생성되는 색인의 크기와 6개 질의의 실행 시간을 비교한다. 데이터 집합은 임의의 데이터로 보편적인 RDF 트리플 데이터의 특성과 제안하는 기법의 효율이

표 3. RDF 트리플 데이터 특성

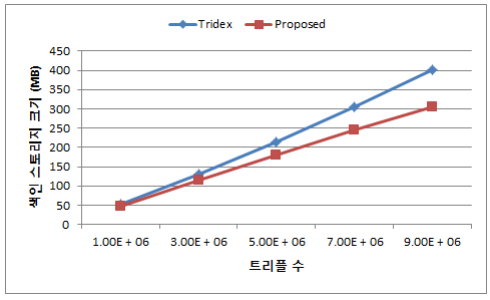
데이터집합	#Triples	#S	#O	#(S∩O)	#P
Dataset1	1,000,000	300,000	250,000	10,000	10,000
Dataset2	1,000,000	150,000	125,000	37,500	5,000

떨어질 수 있는 데이터 집합의 특성으로 데이터를 생성하여 실험한다. [표 3]는 성능평가에 사용한 RDF 트리플 데이터 집합의 특성을 나타낸다. 제안하는 기법의 우수성을 나타내기 위해 비교한 기존의 기법은 주어, 술어, 목적어를 사용해 각각 S, P, O 총 3개의 B+-트리 색인을 이용하는 Tridex 기법과 비교하였다.

[그림 12]는 트리플 수에 따른 색인의 크기를 나타낸다. 성능평가는 두 가지 데이터 집합을 사용하여 진행된 결과를 나타낸다. 결과를 확인하면 Dataset 1을 사용한 성능평가는 기존 기법과 비교하여 저장 공간 사용 효율이 약 10%, Dataset 2를 사용한 성능평가는 약 16% 향상된 결과를 확인 할 수 있다. 이러한 결과는 색인의 크기를 측정할 때 Dataset1은 주어와 목적어의 중복되는 값이 적고 술어 값의 개수가 크기 때문에 기존의 기법과 색인의 크기의 차이가 두드러지지 않지만, 보편적인 RDF 트리플 데이터의 특성을 갖고 있는 Dataset2를 사용한 성능평가에서 제안하는 기법은 기존 기법과는 다르게 중복되는 값이 많은 S, O를 한 개의 색인으로 구성하였기 때문에 트리플의 수가 증가할수록 저장 공간 크기의 증가폭이 감소하는 것을 확인하였다.



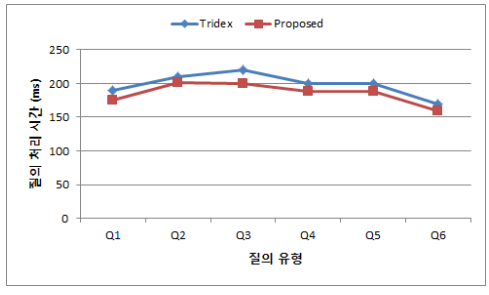
(a) Dataset 1



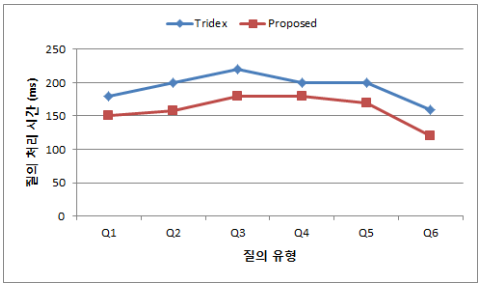
(b) Dataset 2

그림 12. 트리플 수에 따른 색인의 크기

[그림 13]은 질의유형에 따른 질의 처리 시간을 나타낸다. 성능평가 결과, 평균적으로 Dataset 1을 사용한 성능평가는 기존 기법에 비해 약 6%의 성능 향상을 나타낸 반면 Dataset 2를 사용한 성능평가 결과는 기존기법에 비해 제안하는 기법의 질의 처리 시간이 약 17% 감소하는 효과를 나타낸다. Dataset1을 사용한 평가에서는 3개의 색인을 사용하는 기존기법과 질의 처리 속도 성능이 크게 차이 나지 않지만, Dataset2를 사용한 평가는 RDF 트리플 데이터의 특성을 고려하여 S와 O의 중복을 배제하고 색인의 크기를 감소시켜 질의 처리를 하기 때문에 제안하는 색인 기법의 질의 처리 시간이 기존 기법과 비교하여 크게 향상됨을 확인 할 수 있다. 특히 크기가 작은 P 색인을 시작으로 질의를 처리하는 Q1, Q2, Q6 질의에 대하여 제안하는 기법의 처리 속도가 크게 향상됨을 확인하였다.



(a) Dataset 1



(b) Dataset 2

그림 13. 질의 유형에 따른 질의 처리 시간

V. 결론

본 논문에서는 대규모 RDF 처리를 위한 효율적인 색인 기법을 제안하였다. 기존 색인 연구는 트리플 데이터 하나당 총 6번의 중복 저장이 일어나는 단점을 갖고 있다. 버켓을 활용하는 기존의 연구들 또한 RDF 트리플 데이터의 특성을 고려하지 않았기 때문에 빅데이터에 적합하지 않은 색인 구조를 갖고 있다. 제안하는 기법은 RDF 트리플 중 주어와 목적어 값들의 중복이 빈번하다는 특징을 이용하여 주어와 목적어를 하나의 색인으로 구성한다. 또한 트리플 중 상대적으로 가장 적은 수의 값을 갖고 있는 P는 색인을 따로 구성하여 총 색인의 크기를 최소화하였다. 성능평가 결과, 제안하는 기법은 기존 기법에 비해 질의 처리 시간이 약 17% 감소하는 것을 확인하였다. 제안하는 기법은 특히 P 색인을 우선적으로 선택하여 검색할 수 있는 질의에 대하여 기존 기법과 비교하여 뛰어난 성능을 나타냈다. 이러한 결과는 제안하는 기법의 색인 구조가 RDF 트리플 데이터의 특성을 고려하여 색인을 구성하였기 때문에 크기가 작은 P 색인을 이용한 검색의 우수한 결과이다. 기존 기법과의 성능평가로 나타난 결과는 실험에 사용한 데이터의 양이 증가할수록 좋은 효율을 보여준다. 따라서 계속해서 증가하는 모든 RDF 트리플 빅데이터에 적합한 색인의 구조임을 증명하고 있다. 향후에는 성능평가 질의의 유형과 데이터의 종류를 더욱 다양화하여 제안하는 기법의 성능의 우수성을 다시 한 번 입증하며 제안하는 색인 기법을 실질적인 대용량 RDF 데이터 관리 시스템에 적용할 예정이다.

참 고 문 헌

[1] Anupriya Ankolekar, Markus Krotzsch, Thanh Tran, and Denny Vrandecic, “The two cultures: Mashing up Web 2.0 and the Semantic Web,” In Proceedings of the 16th WWW, pp.825-834, 2007.

[2] http://semanticweb.org

[3] Tim Berners Lee, James Hendler, and Ora Lassila, “The Sementic Web,” In Proceedings of the Scientific American, Vol.284, No.5, pp.34-43, 2001.

[4] 고훈준, 유원희, “응용프로그램의 검색을 위한 RDF 메타데이터 시스템의 설계”, 한국콘텐츠학회논문지, 제5권, 제6호, pp.1-9, 2005.

[5] Dongmin Seo, Seungwoo Lee, Pyung Kim, Hanmin Jung, Mikyoung Lee, and Won-Kyung Sung, “RDF Labeling Scheme for Quickly Determining RDF Structural Relationships,” 한국콘텐츠학회논문집, pp.259-260, 2010.

[6] Shijie Zhang, Shirong Li, and Jing Yang, “GADDI: Distance Index based Subgraph Matching in Biological Networks,” In Proceedings of the 12th international conference on extending database technology, 2009.

[7] Medha Atre, Vineet Chaoji, Mohammed J. Zaki, and James A. Hendler, “Matrix “Bit”loaded: A Scalable Lightweight Join Query Processor for RDF Data,” In Proceedings of the WWW, pp.41-50, 2010.

[8] Jiewen Huang, Daniel J. Abadi, and Kun Ren, “Scalable SPARQL Querying of Large RDF Graphs,” In Proceedings of the PVLDB, Vol.4, No.11, pp.1123-1134, 2011.

[9] Andreas Harth and Stefan Decker, “Optimized index structures for querying RDF from the web,” In Proceedings of the 3rd LA-WEB, pp.71-80, 2005.

[10] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein, "HexaStore: sextuple indexing for semantic web data management," In Proceedings of the VLDB Endowment, Vol.1, No.1, pp.1008-1019, 2008.
[11] Thomas Neumann and Gerhard Weikum, "RDF-3X:a RISC-style enginee for RDF," In Proceedings of the PVLDB Endowment, Vol.1, No.1, pp.647-659, 2008.
[12] Seungseok Kang, Junho Shim, and Sang goo Lee, "Tridex: A lightweight triple index for relational database-based semantic web data management," Expert Systems with Applications, Vol.40, No.9, pp.3421-3431, 2013.
[13] George H. L. Fletcher and Peter W. Beck, "Scalable indexing of RDF graphs for efficient join processing," In Proceedings of the 18th ACM conference on Information and knowledge management, CIKM, pp.1513-1516, 2009.
[14] http://www.w3.org/TR/rdf-sparql-query
[15] http://www.w3.org/TR/rdf-schema
[16] http://www.w3.org/TR/2011/WD-rdf-concepts-20110830
[17] http://swat.cse.lehigh/edu/projects/lubm
[18] http://dev.isp-sib.ch/projects/uniprot-rdf

저 자 소 개

김 기 연(Kiyeon Kim) 준회원

- 2013년 2월 : 충북대학교 정보통신공학과(공학사)
- 2013년 3월 ~ 현재 : 충북대학교 정보통신공학부 석사과정

<관심분야> : 빅데이터, 맵-리듀스, RDF, 데이터 베이스 시스템 등

윤 종 현(Jonghyeon Yoon) 준회원

- 2013년 2월 : 충북대학교 정보통신공학과(공학사)
- 2013년 3월 ~ 현재 : 충북대학교 정보통신공학부 석사과정

<관심분야> : P2P 네트워크, 부하분산, 빅데이터, 데이터 베이스 시스템 등

김 천 중(Cheonjung Kim) 준회원

- 2013년 2월 : 한국교통대학교 컴퓨터공학과(공학사)
- 2013년 3월 ~ 현재 : 충북대학교 정보통신공학부 석사과정

<관심분야> : 빅데이터, 맵-리듀스, RDF, 데이터 베이스 시스템 등

임 종 태(Jongtae Lim) 정회원

- 2009년 2월 : 충북대학교 정보통신공학과(공학사)
- 2011년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2011년 3월 ~ 현재 : 충북대학교 정보통신공학과 박사과정

<관심분야> : 데이터베이스 시스템, 시공간 데이터베이스, 위치기반 서비스, 모바일 P2P 네트워크, 빅데이터 등

복 경 수(Kyoungsoo Bok) 종신회원

- 1998년 2월 : 충북대학교 수학과 (이학사)
- 2000년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2005년 2월 : 충북대학교 정보통신공학과(공학박사)
- 2005년 3월 ~ 2008년 2월 : 한국과학기술원 전산학과 Postdoc
- 2008년 3월 ~ 2011년 2월 : ㈜가인정보기술 연구소
- 2011년 3월 ~ 현재 : 충북대학교 정보통신공학과 초빙부교수

<관심분야> : 데이터베이스 시스템, 위치기반서비스, 모바일 P2P 네트워크, 소셜 네트워크 서비스, 빅데이터 등

유 재 수(Jaesoo Yoo) 종신회원

- 1989년 2월 : 전북대학교 컴퓨터공학과(공학사)
- 1991년 2월 : KAIST 전산학과 (공학석사)
- 1995년 2월 : KAIST 전산학과 (공학박사)
- 1995년 3월 ~ 1996년 8월 : 목포대학교 전산통계학과 (전임강사)
- 1996년 8월 ~ 현재 : 충북대학교 정보통신공학부 및 컴퓨터정보통신연구소 교수
- 2009년 3월 ~ 2010년 2월 : 캘리포니아주립대학교 방문교수

<관심분야> : 데이터베이스시스템, 빅데이터, 센서네트워크 및 RFID, 소셜 네트워크 서비스, 분산 객체컴퓨팅, 바이오인포매틱스 등