Journal of KIIT. Vol. 19, No. 1, pp. 11-21, Jan. 31, 2021. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2021.19.1.11

하이브리드 인덱스 시스템을 위한 핫-콜드 세그먼트 분류 알고리즘

윤슬기*, 전석주**, 이석룡***, 김상철****, 이용주*****

Hot-Cold Segment Identification Algorithm for Hybrid Index System

Seulgi Yoon*, Seok-Ju Chun**, Seok-Lyong Lee***, Sang-Cheol Kim****, and Yongju Lee*****

이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. 2016R1D1A1B02008553). 이 논문은 2019년 대한민국 교육부와 한국연구재단의 지원을 받아 수행된 연구임(NRF-2019S1A5A2A03049067).

요 약

그동안 링크드 데이터는 시맨틱 웹을 구현하기 위한 최적의 방법으로 인식되어 왔지만 아직 해결해야 할 이슈들이 남아있다. 특히, 링크드 데이터 환경은 HDD보다 SSD를 사용하는 것이 효율적일 수 있지만, 기존 링크드 데이터 검색 및 저장 기술에 관한 연구들은 대부분 HDD를 기반으로 진행되어 왔다. 이를 해결하기 위해 본 논문에서는 HDD와 SSD를 함께 사용하는 하이브리드 인덱스 시스템을 새로 제안한다. 제안된 시스템은 분산되어 있는 최신 데이터 셋들을 인덱싱하여 검색 속도를 향상시키고, 의사결정 트리를 사용하여 핫 세그먼트는 SSD에, 콜드 세그먼트는 HDD에 저장함으로써 SSD의 사용량을 절약한다. 제안된 시스템은 SSD를 최소한으로 사용하면서 성능은 SSD의 70% 이상을 유지하여 비용과 성능의 최적 균형을 맞추었다.

Abstract

Until now, Linked Data have been recognized as the best way to implement the Semantic Web, but there are still many issues to be resolved. In particular, in large amount of data environment such as Linked Data, it may be more efficient to use SSD than HDD, most of the existing researches on Linked Data retrieval and storage technologies, however, have been mostly based on HDD. To solve this problem, we propose a new hybrid index system that uses both HDD and SSD. The proposed system improves the query performance by indexing the latest distributed data sets and reduces SSD's capacity by using the decision tree to store hot segment on SSD and cold segment on HDD. The proposed system achieved the optimal balance of cost and performance by using SSD to a minimum and maintaining more than 70% of SSD performance.

Keywords

linked data, RDF, SPARQL, HDD, SSD, decision tree, hot segment, cold segment

* 경북대학교 IT대학 컴퓨터학부
- ORCID: https://orcid.org/0000-0003-4516-1089
** 서울교육대학교 컴퓨터교육과 교수
- ORCID: https://orcid.org/0000-0003-1299-1203
*** 한국외국어대학교 산업경영공학과 교수
- ORCID: https://orcid.org/0000-0002-8630-5395
**** 한세대학교 사회복지학과 교수
- ORCID: https://orcid.org/0000-0001-6849-1393
***** 경북대학교 IT대학 컴퓨터학부 교수(교신저자)
- ORCID: https://orcid.org/0000-0002-1705-4967

· Received: Oct. 19, 2020, Revised: Nov. 13, 2020, Accepted: Nov. 16, 2020
· Corresponding Author: Yongju Lee
School of Computer Science and Engineering, Kyungpook National University, 80, Daehak-ro, Buk-gu, Daegu 41566, Korea,
Tel.: +82-53-950-7285, Email: yongju@knu.ac.kr

Ⅰ. 서 론

링크드 데이터(Linked data)는 시맨틱 웹 환경을 만들기 위한 방법의 하나로 웹상에 존재하는 데이터를 개별 URI(Uniform Resource Identification)로 식별하고, 공개, 공유 연결하는 방법이다[1]. 링크드 데이터의 핵심 개념은 일반 웹 아키텍처를 하나의 글로벌한 규모의 데이터베이스처럼 사용할 수 있다는 것이다. 링크드 데이터를 게시하고 여러 리소스를 연결하기 위해 URI, RDF(Resource Description Framework), HTTP(Hypertext Transfer Protocol) 기술이 적용된다. 여기서 RDF는 주어(Subject), 술어(Predicate), 목적어(Object)의 트리플(Triple) 형태로 구성된다. 그동안 링크드 데이터는 시맨틱 웹(Semantic web)을 구현하기 위한 최적의 방법으로 인식되어 왔지만 해결해야 할 이슈들이 아직까지 남아있다. 예를 들면, RDF 트리플은 스키마를 사용하지 않고 그래프로 모델링되기 때문에 기존의 XML 솔루션 등에 직접 적용할 수 없다. 따라서 링크드 데이터의 검색 및 저장을 위한 새로운 솔루션이 필요하다. 링크드 데이터 검색 및 저장 기술에 관한 기존 연구는 다음과 같이 분류할 수 있다.

첫째, 모든 RDF 트리플을 로컬 저장소에 복사 및 관리하는 로컬 접근 방식(Local approach) 이다[2][3]. 로컬 접근 방식은 RDF 데이터를 미리 수집하여 사전 처리 후 결합된 데이터를 로컬 저장소에 저장한다. 이 방식은 네트워크 트래픽이 발생하지 않기 때문에 응답 속도는 빠르지만, 최신 데이터를 반영하지 못한다. 둘째, 재귀적인 URI 탐색 엔진을 사용하여 분산된 저장소에 접근하는 실시간 탐색 방식(Live exploration approach)이다[4][5]. 실시간 탐색 방식은 분산된 SPARQL(SPARQL Protocol and RDF Query Language) 엔드 포인트에 대해 검색을 수행한다. 이 방식은 데이터를 동기화할 필요가 없으며 최신 데이터를 반영할 수 있지만, 게시자가 신뢰할 수 있는 SPARQL 엔드 포인트를 제공한다고 보장할 수 없다. 셋째, 효율적인 검색 처리를 위해 링크드 데이터 셋에 대해 인덱싱하는 인덱스 방식(Index approach)이다[6][7]. 인덱스를 사용함으로써 분산되어 있는 데이터 셋들을 효율적으로 탐색할 수 있고, 필요한 부분만 빠르게 필터링할 수 있지만, 인덱스의 구축 및 유지 보수에 큰 비용이 필요하다. 본 논문은 각 방식의 단점을 보완하기 위해 인덱스 방식과 실시간 탐색 방식을 융합한 하나의 새로운 하이브리드 인덱스 시스템을 개발함으로써 두 접근법의 단점을 제거하고 장점을 얻는다.

링크드 데이터를 효율적으로 검색 및 저장하는 기존 연구는 대부분 HDD(Hard Disk Drive)를 기반으로 진행되어 왔다. 링크드 데이터는 대용량 데이터를 얼마나 효율적으로 저장하고, 원하는 정보를 빨리 찾아낼 수 있느냐가 하나의 중요한 목표가 될 수 있다. 이를 위해선 HDD보다 SSD(Solid State Drive)를 사용하는 것이 보다 효율적일 수 있다. 하지만 SSD의 가격 하락이 계속 진행되고 있지만, 여전히 SSD의 가격은 2020년 기준 1GByte당 240원으로 1GByte당 94원인 HDD에 비해 2.6배 비싸다[8]. 그렇기 때문에 전체 저장 장치를 SSD로 대체하는 것은 비효율적이다. 본 논문에서 SSD 성능의 일정 수준을 유지하면서 비용은 최소한으로 사용하기 위해 HDD와 SSD를 함께 사용하는 하이브리드 저장 시스템을 여과(Filtering)와 정제(Refinement) 단계가 분리된 인덱스 시스템에 활용한다. 머신러닝(Machine learning)[9] 기술 중 의사결정 트리(Decision tree)를 사용하여 접근이 적은 정적인 속성의 데이터는 콜드 세그먼트로 분류하여 HDD에 저장하고, 빈번히 접근되는 동적인 속성의 데이터는 핫 세그먼트로 분류하여 SSD에 저장함으로써 비싼 SSD의 저장 효율을 높인다. 결과적으로 SSD를 최소한으로 사용하면서 성능은 일정 수준을 유지하여 비용과 성능의 균형을 맞출 수 있다.

본 논문은 다음과 같이 구성된다. 2장에서 관련 연구를 살펴보고, 3장에서 하이브리드 저장 장치 기반 2단계 인덱스 시스템을 제안한다. 4장에서 실험 및 분석을 수행하고 5장에서 결론을 내린다.

Ⅱ. 관련 연구

2.1 링크드 데이터 검색 및 저장 기술

그동안 링크드 데이터 검색 및 저장 기술에 관한 많은 연구가 진행되어 왔다. 기존 연구는 로컬 접근 방식, 실시간 탐색 방식, 인덱스 방식으로 분류할

수 있으며 표 1은 링크드 데이터 검색 및 저장 기술에 대한 기존 연구를 요약하여 나타낸 표이다.

표 1. 링크드 데이터 저장 및 검색 기술
Table 1. Storage and retrieval technology for linked data

	Description	Related researches
Local approach	All RDF data are stored in the local repository	Hexastore[2], RDF-3X[3]
Live exploration approach	Access to RDF datasets via SPARQL endpoints	DARQ[4], SemWIQ[5]
Index approach	Use pre-populated indexes.	MDH[6], QTree[7]

Weiss 등[2]은 수직 분할 아이디어의 강점을 이용하였다. RDF 트리플의 3가지 요소로 가능한 순서에 대해 6가지 방법으로 인덱싱한다. RDF 트리플의 각 요소는 두 개의 벡터와 연관된다. 각 벡터는 각 벡터 요소에 부착된 세 번째 유형인 자원 목록과 함께 다른 유형의 요소들을 수집한다. 따라서 6가지 튜플(Tuple) 인덱싱 체계를 나타내는 Hexastore를 제안했다. Neumann 등[3]은 RDF 트리플을 저장 및 인덱싱하는 솔루션으로 물리적인 변경이 필요하지 않고, 빠른 머지 조인을 활용한 간단한 검색 프로세서이며 최적의 조인 순서를 선택하기 위한 검색 최적화 프로그램인 RISC(Reduced Instruction Set Computer) 스타일의 아키텍처를 추구한 RDF-3X를 제안하였다.

Quilitz 등[4]은 SPARQL 엔드 포인트를 사용하면 네트워크 트래픽으로 인해 많은 오버헤드가 발생하여 검색 성능이 저하되는 문제를 해결하기 위해 DARQ를 제안하였다. DARQ는 실제 RDF 데이터가 웹에 분산되어 있음에도 불구하고 하나의 단일 RDF 그래프를 검색하는 것처럼 보이게 한다. Langegger 등[5]은 SPARQL을 사용하여 데이터를 검색하고 웹에서 접근할 수 있다면 링크드 데이터로 설명할 수 있도록 하였다.

Harth 등[6]은 링크드 데이터 원칙을 준수하는 소스의 그래프 구조를 요약한 근사 인덱스 구조를 제안하였다. Umbrich 등[7]은 검색을 평가하는 중에 관련된 소스를 결정하기 위한 데이터 요약으로 다차원 인덱스를 사용하는 방법을 제안하였다.

2.2 국내외 저장 장치 연구 현황

Shi 등[10]은 SSD를 사용하여 하이브리드 저장 장치 시스템을 구축하고 적절한 데이터를 SSD로 이동시키는 알고리즘을 제안하였다. 데이터 할당 문제를 MCKP(Multiple Choice Knapsack Problem)로 공식화하고 최적의 솔루션을 찾기 위해 MDP(Multiple stage Dynamic Programming)를 제안하였고 기존 알고리즘보다 최대 6배까지 향상될 수 있음을 보였다. 최지현 등[11]은 요청된 파일에 대한 메타 데이터 검색 속도를 향상시키기 위해 SSD 기반 블룸 필터(Bloom filter)를 사용한 새로운 디렉터리 파싱 기술을 제안했다. Fevgas 등[12]은 SSD와 3DXpoint 메모리를 결합하여 새롭고 빠른 검색 결과를 제공하는 하이브리드 검색 프레임워크를 제안했다. 이 방법은 SSD의 높은 대역폭, 최신 SSD의 내부 병렬 처리, 그리고 NVMe 프로토콜의 효율성을 활용하여 I/O 작업을 그룹화한다.

2.3 핫-콜드 데이터 식별 연구

최근 핫-콜드 데이터 식별 연구로는 MBF (Multiple Bloom Filter)[13], HotDataTrap[14] 등을 포함하고 있다. MBF 방식은 m비트의 배열로 표현되는 블룸 필터 n개와 k개의 서로 다른 해시 함수를 사용하여 데이터를 식별한다. 검색할 데이터가 집합에 속하는지 확인하기 위해 데이터의 주소값을 서로 다른 k개의 해시 함수를 사용하여 얻은 값과 대응되는 위치의 비트 값을 확인한다. 모든 비트의 값이 1이면 해당 데이터가 집합에 속한다는 뜻으로 핫 데이터임을 의미한다. 그림 1은 MBF 방식의 진행 과정이다.

HotDataTrap 방식은 기본키, 하위키, 빈도수, 최근성 항목으로 구성된 카운터 테이블(Counter table)을 유지하며 핫 데이터를 식별한다. 기본키는 입력된 데이터 주소의 상위 비트 일부를 해시한 값이 저장되고, 하위키는 보다 더 정확히 데이터를 식별하기 위해 기본키에서 사용한 상위 비트를 제외한 나머지 하위 비트를 해시한 값이 저장된다. 빈도수는 해당 데이터에 접근된 횟수가 저장되고 최근성은 해당 데이터에 최근에 접근되었는지의 여부가 저장된다.

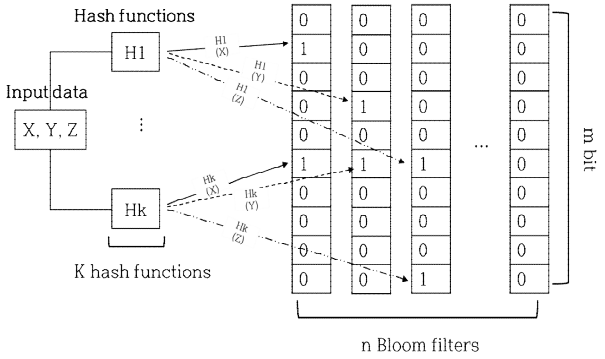


그림 1. MBF 구조
Fig. 1. MBF structure

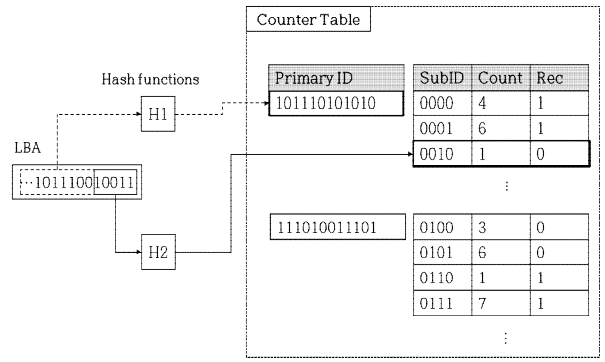


그림 2. HotDataTrap 구조
Fig. 2. HotDataTrap structure

검색할 데이터가 입력되면 기본키로 테이블의 영역을 찾고 하위키를 사용하여 정확한 위치를 찾는다. 미리 정의된 핫 데이터의 조건과 테이블에 저장된 데이터의 정보를 비교하여 조건을 만족하면 해당 데이터를 핫 데이터로 식별한다. 그림 2는 HotDataTrap 방식의 진행 과정이다.

III. 하이브리드 기반 2단계 인덱스 시스템

3.1 여과와 정제가 분리된 인덱스 구조

본 논문에서는 기존 링크드 데이터 검색 및 저장 기술의 단점을 보완한 여과와 정제 단계가 분리된 인덱스 구조를 제안한다. RDF 트리플은 3차원 데이터이며 그래프로 모델링되기 때문에 기존 데이터베이스에서 사용되는 인덱스 구조를 그대로 적용할 수 없다. 이를 해결하기 위해 R*-tree[15]를 3차원 데이터 환경에 적합하게 다시 프로그래밍 하였다. 불규칙한 3차원 공간의 점들을 효율적으로 관리할 수 있는 MBB(Minimum Bounding Box) 기반 인덱스 구조인 R*-tree와 버킷 내의 점들을 구조화할 수 있도록 Kd-tree(K dimensional-tree)[16]를 결합시켰다. 디스크 환경에 최적화된 인덱스 구조인 R*-tree를 플래시 메모리에 적합하게 개선한 Kd-tree와 결합하여 사용함으로써 SSD 환경에 최적화 시킨다.

여과와 정제 단계가 분리된 인덱스 구조를 구축하기 위해 RDF 트리플을 3차원 공간의 점으로 변환해야 한다. 점으로 변환하기 위해 RDF 트리플의 구성 요소인 주어, 술어, 목적어에 개별적으로 해시 함수를 적용하여 3개의 숫자를 얻는다.

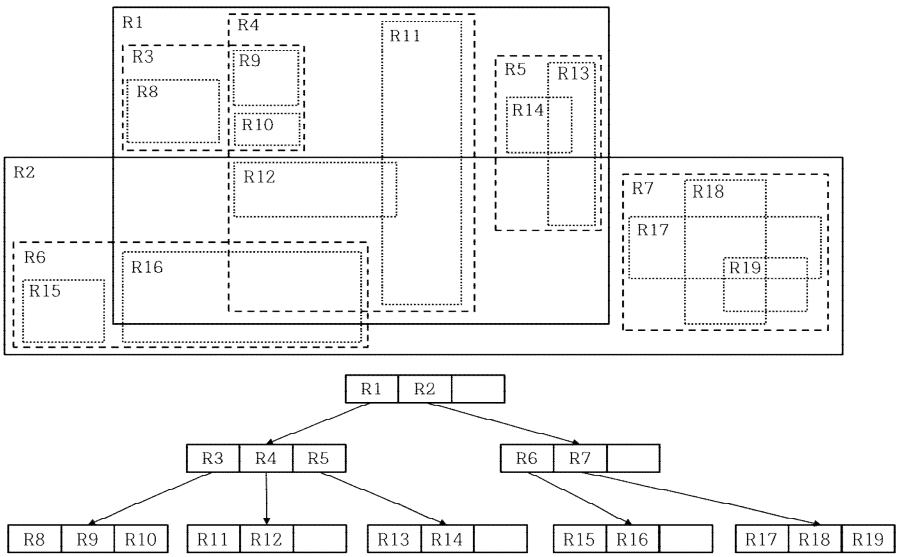


그림 3. 여과 단계 인덱스 구조
Fig. 3. Index structure for filter phase

다음 단계에서 3차원 공간은 버킷으로 분할되며 각 버킷에 3개의 숫자로 변환된 RDF 트리플 정보가 포함된다. R*-tree는 내부 노드와 리프 노트로 구성되어 있으며 내부 노드는 자식 노드의 버킷 영역과 주소 지시자를 포함하고 있다. 리프 노드는 버킷 식별자와 MBB 그리고 Kd-tree의 주소 지시자를 포함하고 있다. R*-tree 하부에 있는 Kd-tree는 R*-tree의 버킷을 구성하는 RDF 트리플의 식별자와 URI를 포함하고 있다. 그림 3은 여과 단계를 처리하기 위한 R*-tree를 보여주고 있고, 그림 4는 정제 단계를 위한 Kd-tree를 직관적으로 표현하고 있다.

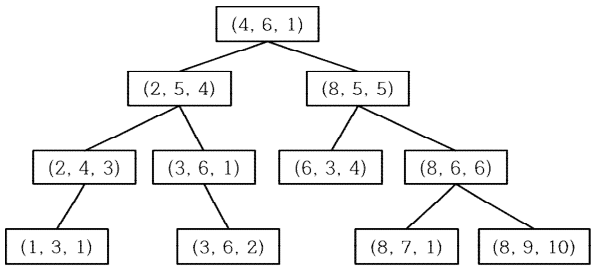


그림 4. 정제 단계 인덱스 구조
Fig. 4. Index structure for refinement phase

3.2 2단계 인덱스 조인 연산

SPARQL 질의는 8개의 트리플 패턴이 있다. 이들 중 (?s, ?p, ?o)는 s, p, o가 어떤 값이든 될 수 있기 때문에 전체 스캔이 필요하고, (s, p, o)는 s, p, o가 정해져 있기 때문에 결과가 존재하거나 존재하지 않는다. 따라서 검색 성능을 향상시키기 위해 고려해야 할 패턴은 (?s, p, o), (s, ?p, o), (s, p, ?o), (?s, ?p, o), (s, ?p, ?o), (?s, p, ?o) 6개이다. 실제 사용자는 패턴 하나를 검색하는 경우보다 6개 패턴을 조인(join)하여 검색하는 경우가 일반적이다.

조인 함수는 Nested-Loop 조인, Sort-Merge 조인, Hash 조인이 있다. Nested-Loop 조인은 선행 테이블의 처리 범위를 하나씩 접근하면서 추출된 값으로 연결할 테이블을 조인하는 방식이다. Sort-Merge 조인은 양쪽 테이블의 처리 범위를 각자 접근하여 정렬한 결과를 스캔하면서 연결의 조건으로 합병하는 방식이다. Hash 조인은 해시 값을 이용해 조인하는 방식으로 병렬처리를 사용하기 때문에 대용량 데이터를 처리하는데 적합한 방법이다. Sort-Merge 조인은 데이터 리스트를 반드시 정렬해야 하므로 시간이 오래 걸리는 단점이 있고, Hash 조인은 메모리 사용량이 많다는 단점이 있기 때문에 본 논문에서는 모든 트리플에 수정 없이 직관적으로 사용 가능하며 메모리 사용량, 성능 모두 적정 수준을 만족하는 Nested-Loop 조인을 사용하였다.

조인 검색 과정은 그림 5와 같다. 사용자가 검색을 요청하면 먼저 R*-tree가 필요한 데이터가 존재하는 버킷을 필터링한다. 그다음 필터링된 버킷을 Kd-tree로 인덱싱하여 각 트리플 패턴을 구체화하는 정제 과정을 거친다. 마지막으로 각 Kd-tree를 조인한 결과가 존재하면 해당 결과를 사용자에게 반환한다. 사용자에게 반환되는 결과 값은 URL로 해당 엔드 포인트에서 RDF 트리플을 검색할 수 있기 때문에 실시간 탐색이 가능하다.

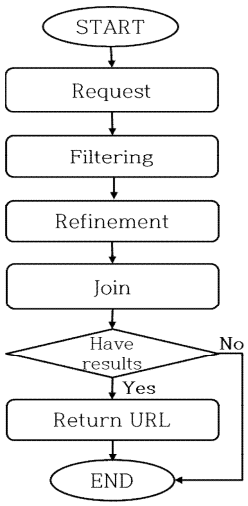


그림 5. 조인 과정 순서도
Fig. 5. Join processing flowchart

3.3 핫-콜드 세그먼트 분류 기법

SSD와 HDD에 데이터를 어떻게 재배치할 것인지 판단하기 위해 핫-콜드 세그먼트 분류 기법을 제안한다. 사용자가 데이터에 접근하면 사용자가 접근한 RDF 트리플을 식별 테이블에 그림 6과 같이 저장한다. S, P, O 속성은 메모리의 사용량을 줄이기 위해 RDF 트리플의 각 요소에 해시 함수를 적용하여 저장한다. Rec 속성은 최근성을 나타내는 속성으로 해당 데이터가 최근에 접근 되었는지의 여부가 저장되고, Count 속성은 데이터에 접근된 횟수가 저장

된다. 마지막으로 Type 속성은 해당 데이터의 핫-콜드 여부가 1과 0값으로 저장된다. 수집된 사용자 접근 데이터의 핫-콜드 여부는 Rec 속성 값이 1이고, Count 속성 값이 미리 정의된 임계값보다 크면 해당 데이터를 핫 데이터로 식별한다. 핫-콜드 데이터를 식별하기 위한 조건 중 Rec 속성은 굉장히 중요하다. 아무리 많이 접근된 데이터라도 최근에 접근되지 않았다면 앞으로도 접근되지 않을 가능성이 크기 때문이다.

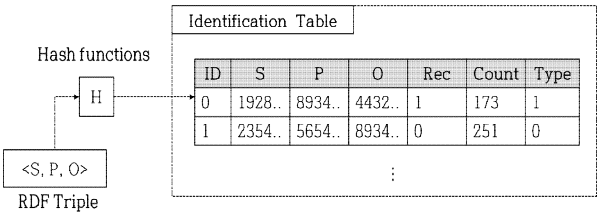


그림 6. 식별 테이블
Fig. 6. Identification table

위 과정은 의사결정 트리의 트레이닝 데이터로 사용하기 위한 관측값을 수집하는 과정이다. 의사결정 트리[17]는 관측값으로 주어진 수많은 데이터를 어떤 속성을 기준으로 식별할지 선택할 때 정보획득 접근법을 사용한다. 정보획득 접근법은 속성의 정보량과 엔트로피를 사용한다. 정보량은 사건이 가지는 정보의 양으로 드물게 발생하는 사건일수록 정보량의 값은 크다. 엔트로피는 정보량의 기댓값으로 발생한 모든 사건들의 정보량의 평균을 구한 것이다. 엔트로피가 크다는 것은 각 사건들이 일어날 확률이 거의 비슷한 경우이다. 즉, 엔트로피가 작을수록 정보 획득량이 크다. 의사결정 트리는 정보 획득량이 큰 속성 순서로 데이터를 식별한다. 다음은 식별 테이블 속성의 정보 획득량을 구하는 과정이다.

먼저 Type 속성에 대한 엔트로피를 구한다. Nhot은 number of hot data, Nall은 number of all data, Ncold는 number of cold data의 약자로 각 데이터의 개수를 의미한다.

$$E(Type) = \frac{Nhot}{Nall}\log_2\frac{1}{\frac{Nhot}{Nall}} + \frac{Ncold}{Nall}\log_2\frac{1}{\frac{Ncold}{Nall}} \quad (1)$$

다음은 Rec 속성에 대한 Type 속성의 엔트로피를 구한다. T는 Type, R은 Rec의 약자이다. Pone은 probability of one, Pzero는 probability of zero의 약자로 속성 값의 확률을 의미한다.

$$E(T|R) = Pone * E(T|one) + Pzero * E(T|zero) \quad (2)$$

Rec의 정보 획득량은 식 (1) - 식 (2)로 구할 수 있다. 이와 같은 과정을 통해 식별 테이블의 모든 속성의 정보 획득량을 구하여 식별할 속성의 순서를 결정한다. 속성의 순서가 결정되면 목표 변수인 테스트 데이터로 입력받은 사용자가 접근하지 않은 데이터의 핫-콜드 여부를 식별하고, 식별 결과를 신뢰할 수 있는지 정확도를 구한다. 의사결정 트리가 식별한 데이터는 RDF 파일로 작성되어 적절한 저장장치에 재배치된다. 핫 데이터로 구성된 R*-tree의 버킷인 핫 세그먼트는 SSD에서 Kd-tree로 인덱싱되고, 나머지 R*-tree의 버킷인 콜드 세그먼트는 HDD에서 Kd-tree로 인덱싱된다. 그림 7은 전체적인 핫-콜드 세그먼트 분류 알고리즘을 프로차트로 표현한 것이다.

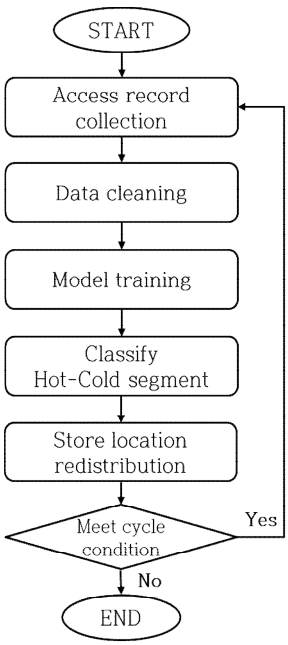


그림 7. 핫-콜드 세그먼트 분류 알고리즘
Fig. 7. Hot-cold segment identification algorithm

3.3 SSD 쓰기 연산

SSD의 치명적인 단점은 덮어쓰기가 불가능하여 새로운 데이터로 업데이트될 때마다 블록 단위로 지우기 연산이 선행되어야 하기 때문에 쓰기 연산의 속도가 느리다는 것과 쓰기 제한이다. 본 논문에서는 불필요한 쓰기 연산을 줄이기 위해 SSD 페이지 크기의 배수가 될 때까지 쓰기 데이터를 수집한 후 쓰기 연산을 수행하는 방법을 채택한다. 이와 같은 방법은 데이터가 수집되는 동안 발생하는 중복 수정 데이터를 제거하고, 빈번하게 수정되는 데이터와 그렇지 않은 데이터를 다른 페이지에 저장하여 Read-Modify-Write 작업을 효율적으로 수행할 수 있도록 한다.

중복 수정은 부차적인 지우기 연산을 발생시켜 쓰기 연산의 속도를 저하시킨다. 빈번하게 수정되는 데이터와 그렇지 않은 데이터가 같은 페이지에 저장된다면 빈번하게 수정되는 데이터가 수정될 때마다 그렇지 않은 데이터까지 함께 Read-Modify-Write 작업에 포함되어 수정이 필요하지 않은 데이터까지 수정해야 하는 문제가 발생한다. 이러한 불필요한 지우기 및 쓰기 연산은 SSD의 셀을 사용 불가능한 상태로 만든다. 사용 불가능 상태는 SSD의 쓰기 횟수 제한을 야기하며 결과적으로 SSD의 수명을 감소시킨다. 페이지 크기의 배수만큼 데이터를 수집하여 쓰기 연산을 수행하면 앞에서 언급한 문제 발생을 줄여 쓰기 연산의 성능을 개선할 수 있다.

그림 8은 쓰기 연산 알고리즘이다. 사용자가 쓰기 연산을 요청하면 해당 쓰기 연산을 버퍼에 저장하고 버퍼의 크기가 저장 장치의 페이지 크기의 배수인지 확인한다.

```
WHILE (((buffer size of write operation % page size of storage) == 0) or (iterate time > predetermined time)) {
        start = current time
        request user's write operation
        save write operation and delimiter in buffer
        end = current time
        iterate time += (end - start)
}
execute write operation stored in buffer
```

그림 8. SSD 쓰기 연산 알고리즘
Fig. 8. Write operation algorithm for SSD

쓰기 연산의 크기를 비교하기 전에 버퍼에 저장하는 이유는 한 번의 쓰기 연산은 높은 확률로 페이지의 크기를 초과하지 않기 때문이다. 그리고 현재 쓰기 연산의 끝을 알리는 구분자와 함께 저장한다. 쓰기 연산의 무한 대기 발생을 제거하기 위해 일정 시간이 지나면 요청된 쓰기 연산의 크기가 저장 장치의 페이지 크기를 초과하지 않더라도 쓰기 연산을 수행한다.

IV. 실험 및 분석

4.1 데이터 셋

본 논문에서 제안한 솔루션의 성능을 분석하기 위해 DBpedia[18], Drugbank[19], LinkedGeoData[20] 데이터 셋을 사용하였다. DBpedia는 위키피디아에서 추출한 RDF 정보가 포함되어 있고, Drugbank는 약물과 약물 표적에 대한 정보가 포함되어 있다. LinkedGeoData는 공간 지식에 대한 방대한 정보를 가지고 있다. 표 2는 데이터 셋의 특징을 보여준다.

표 2. 데이터 셋의 특징
Table 2. Feature of data sets

Data set	Size (MB)	No. of triples	No. of subjects	No. of predicates	No. of objects
DBpedia	3.94	31,050	4,008	23	16,644
Drugbank	144	766,920	19,693	119	274,864
LinkedGeo Data	327	2,207,295	552,541	1,320	1,017,242

4.2 의사결정 트리 성능 평가

그림 9, 그림 10, 그림 11은 의사결정 트리가 DBpedia, Drugbank, LinkedGeoData의 핫-콜드 데이터를 식별한 과정을 보여준다. 식별을 위한 첫 번째 기준인 X[1]은 Rec 속성이다. Rec 속성은 최근에 해당 데이터에 접근되었는지를 나타내는 속성으로 1 또는 0의 두 가지 값만 가질 수 있다. 의사결정 트리는 Rec 속성 값이 0.5 이하이면 콜드 데이터로 식별하였다. Rec 속성의 식별 기준은 모든 데이터 셋에서 동일하게 적용된다. 식별을 위한 두 번째 기준인 X[2]는 Count 속성으로 데이터의 접근 횟수를 나타낸다.

Rec 속성으로 0.5 이상으로 식별된 데이터 중에서 Count 속성 값이 임계값 이상이면 핫 데이터로 식별한다.

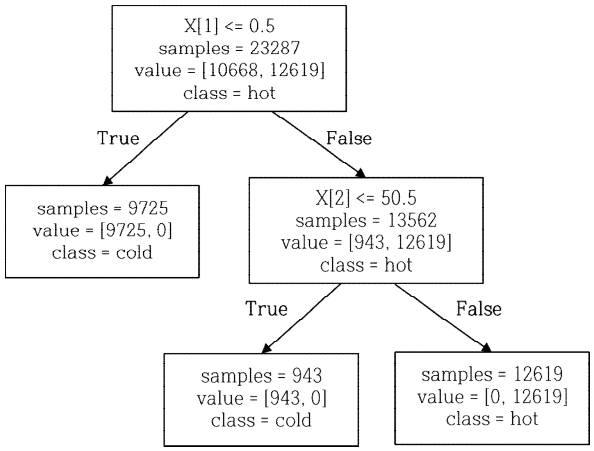


그림 9. DBpedia 식별 과정
Fig. 9. DBpedia identification processing

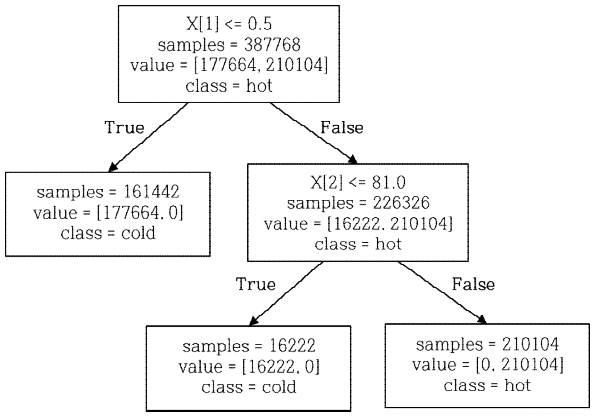


그림 10. Drugbank 식별 과정
Fig. 10. Drugbank identification processing

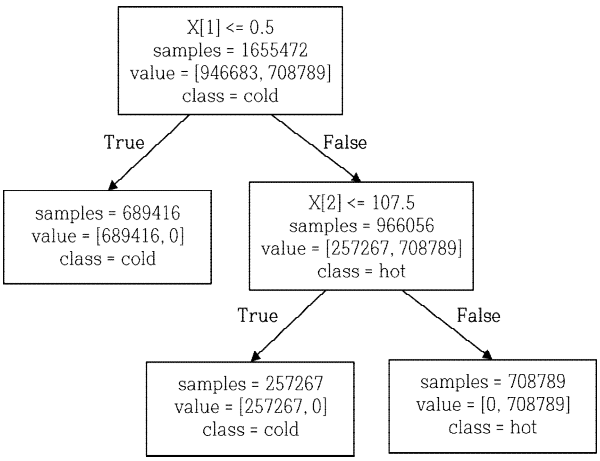


그림 11. LinkedGeoData 식별 과정
Fig. 11. LinkedGeoData identification processing

임계값 역시 의사결정 트리가 트레이닝 데이터를 분석하여 적절한 값으로 설정한다. DBpedia의 임계값은 50.5이고, Drugbank의 임계값은 81.0이며 LinkedGeoData의 임계값은 107.5이다.

그림 12는 의사결정 트리가 각 데이터 셋을 식별한 결과의 정확도를 그래프로 나타낸 것이다. 의사결정 트리의 깊이가 깊을수록 오버피팅(Over fitting)이 일어나 정확도는 낮아진다. 본 논문에서 구축한 의사결정 트리는 깊이가 3으로 그다지 깊지 않기 때문에 실제 데이터에 일반화하여 사용하여도 오류가 없는 높은 정확도를 만족시킨다. DBpedia, Drugbank, LinkedGeoData의 평균 정확도는 각각 약 95%, 98%, 98%로 매우 높은 정확도를 보인다.

그림 13은 의사결정 트리가 식별한 핫-콜드 데이터를 사용하여 각 데이터를 적절한 저장 장치로 재배치한 결과이다. 모든 데이터 셋이 핫 세그먼트를 저장하기 위해 SSD를 평균 42% 사용하였고, 콜드 세그먼트를 저장하기 위해 HDD를 평균 58% 사용하여 SSD를 최소한으로 사용하였다.

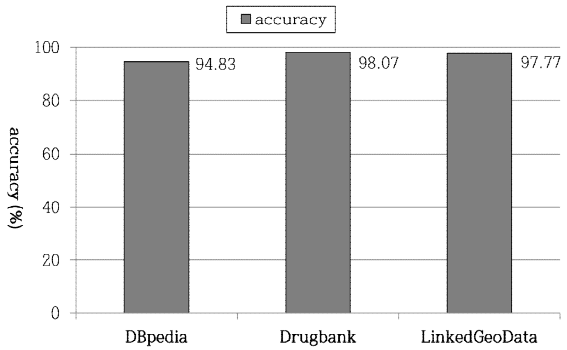


그림 12. 의사결정 트리 정확도
Fig. 12. Accuracy of decision tree

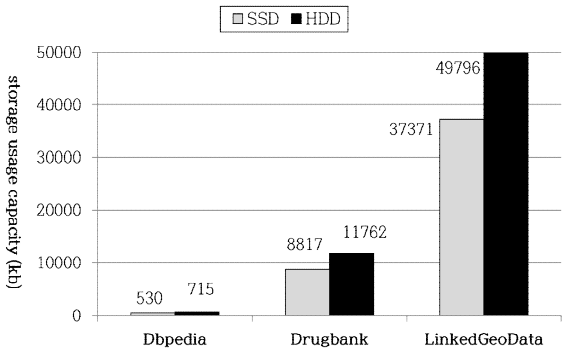


그림 13. 데이터 셋의 저장 장치 사용량
Fig. 13. Storage usage of data sets

4.3 검색 성능 평가

그림 14는 SSD만 사용한 시스템(SSD라 함)과 본 논문에서 제안한 하이브리드 저장 장치를 사용한 시스템(Hybrid Storage System으로 HSS라 함), 그리고 HDD만 사용한 시스템(HDD라 함)의 검색 성능을 비교한 그래프이다. 모든 데이터 셋에서 SSD의 성능은 HDD에 비해 1.6배 이상 빠른 성능을 보이지만 비용은 2.6배나 더 비쌌다. 이에 비용과 성능의 최적 균형을 맞춘 HSS의 성능은 비싼 SSD를 최소한으로 사용하면서 SSD 검색 성능의 70% 이상을 유지하였다.

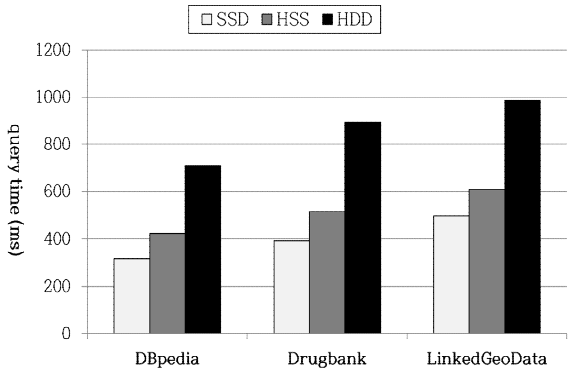


그림 14. 검색 성능
Fig. 14. Searching performance

4.4 SSD 쓰기 연산 성능 평가

그림 15는 페이지 크기의 배수 쓰기 연산과 즉시 쓰기 연산을 수행했을 때의 성능을 비교한 그래프이다. Page size write는 SSD의 페이지 크기의 배수가 될 때까지 쓰기 데이터를 수집한 후 쓰기 연산을 수행하는 것을 의미하고, Immediate write는 즉시 쓰기 연산을 의미한다. 본 실험에서 사용한 SSD의 페이지 크기는 4KByte이다. 4KByte만큼 쓰기 데이터가 수집되는 동안 발생하는 중복 수정 데이터를 제거하고, 빈번하게 수정되는 데이터와 그렇지 않은 데이터를 서로 다른 페이지를 사용하도록 함으로써 Read-Modify-Write 작업이 효율적으로 수행되도록 하였다. 결과적으로 불필요한 지우기 및 쓰기 연산과 셀의 사용 불가능 상태를 감소시켜 쓰기 연산의 성능 향상과 쓰기 횟수 제한을 개선했다. 쓰기 연산 횟수가 적을 때도 5.9배 이상의 성능 향상을 보였으며 쓰기 연산의 횟수가 늘어날수록 본 논문에서 제안한 방식의 성능이 더욱 우수함을 보였다.

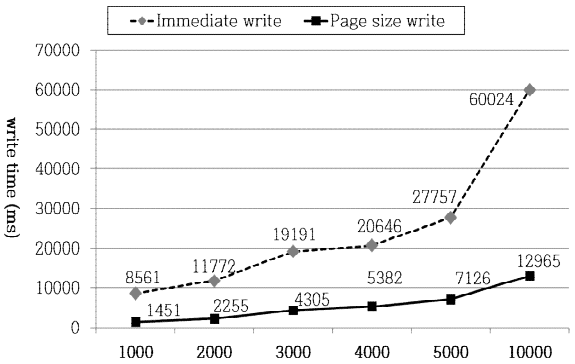


그림 15. 쓰기 성능
Fig. 15. Writing performance

V. 결 론

기존 링크드 데이터 검색 및 저장 기술의 이슈들을 보완한 융합 방식으로 다차원 인덱스 구조인 R*-tree와 Kd-tree를 결합한 하이브리드 인덱스 시스템을 제안하였다. 링크드 데이터의 기술적인 요소 중 RDF 트리플은 3차원 데이터이며 그래프로 모델링되기 때문에 기존 데이터베이스에서 사용하는 인덱스 구조를 그대로 적용하는 것은 불가능하다. 이를 해결하기 위해 R*-tree를 3차원 데이터 환경에 적합하게 다시 프로그래밍 하였고, 플래시 메모리 환경에 적합하게 개선한 Kd-tree와 함께 사용함으로써 SSD 환경에 최적화 시켰다.

본 논문의 접근 방식은 인덱스 방식과 실시간 탐색 방식을 융합하여 분산되어 있는 최신 데이터 셋들을 효율적으로 탐색하고 데이터가 검색될 때 필요한 부분만 빠르게 필터링하여 검색 속도를 향상시켰다. 링크드 데이터와 같은 대용량 데이터를 효율적으로 저장하고 정보를 빨리 찾기 위해 HDD보다 SSD를 사용하는 것이 효율적일 수 있지만, SSD의 성능은 HDD에 비해 1.6배 이상 빠른 성능을 보이지만 비용은 2.6배나 더 비싸기 때문에 전체 저장 장치를 SSD로 대체하는 것은 장기적으로 비효율적일 수 있다. 이를 보완하기 위해 SSD와 HDD를 함께 사용하는 HSS를 여과와 정제 단계가 분리된 인

덱스 시스템에 활용하였다. 본 논문에서는 데이터를 적절한 저장 장치에 재배치하기 위해 핫-콜드 세그먼트 분류 알고리즘을 제안했다. 빈번히 접근되는 핫 세그먼트를 SSD에 저장하고, 접근이 적은 콜드 세그먼트를 HDD에 저장함으로써 SSD의 사용량을 최소화시켰다. 또한, SSD의 쓰기 성능을 개선하기 위해 SSD 페이지 크기의 배수가 될 때까지 쓰기 연산을 수집한 후 쓰기 연산을 실행하는 방법을 채택하였다. 결과적으로 비싼 SSD를 최소한으로 사용하면서 SSD 검색 성능의 70% 이상을 유지하여 비용과 성능의 최적 균형을 맞추었다.

References

[1] Y. J. Lee, "Linked Data Indexing Techniques Using 3D R*-tree", Journal of KIIT, Vol. 15, No. 11, pp. 31-38, Nov. 2017.

[2] C. Weiss, P. Karras, and A. Bernstein, "Hexastore: Sextuple Indexing for Semantic Web Data Management", The 34th International Conference on Very Large Data Bases (VLDB), pp. 1008-1019, Aug. 2008.

[3] T. Neumann and G. Weikum, "RDF-3X: A RISC-style Engine for RDF", The 34th International Conference on Very Large Data ases (VLDB), pp. 647-659, Aug. 2008.

[4] B. Quilitz and U. Leser, "Quering Distributed RDF Data Sources with SPARQL", the 5th European Semantic Web Conference (ESWC), Vol. 5021, pp. 524-538, Jun. 2008.

[5] A. Langegger, W. Woß, and M. Blochl, "A Semantic Web Middleware for Virtual Data Integraion on the Web", The 5th European Semantic Web Conference (ESWC), Vo1. 5021, pp. 493-507, Jun. 2008.

[6] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K. Sattler, and Jurgen Umbrich, "Data Summaries for On-Demand Queries over Linked Data", The 19th International Conference on World Wide Web (WWW), pp. 411-420, Apr. 2010.

[7] J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres, "Comparing Data Summaries for Processing Live Queries over Lined Data", World Wide Web (WWW), Vol. 14, No. 5, pp. 495-544, Oct. 2011.

[8] https://www.samsung.com/sec/memory-storage/all-memory-storage/ [accessed: Nov. 12. 2020]

[9] Y. Keon, H. Kim, J. Y. Choi, D. Kim, S. Y. Kim, and S. Kim, "Call Center Call Count Prediction Model by Machine Learning", Journal of JAITC, Vol. 8, No. 1, pp. 31-42, Jul. 2018.

[10] H. Shi, R. V. Arumugam, C. H. Foh, and K. K. Khaing, "Optimal Disk Storage Allocation for Multi-tier Storage System", 2012 Digest APMRC, Jan. 2013.

[11] J. H. Choi, B. J. Lee, D. Y. Jung, and H. Y. Youn, "An SSD-Based Accelerator Using Partitioned Bloom Filter for Directory Parsing", International Conference on IT Convergence and Security (ICITCS), Oct. 2015.

[12] A. Fevgas, L. Alrotodos, and M. Alamaniotis, "A Study of R-tree Performance in Hybrid Falsh/3DXPoint Storage", 2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA), Nov. 2019.

[13] D. C. Park and D. Du, "Hot Data Identification for Flash-based Storage Systems Using Multiple Bloom Filters", Mass Storage Systems and Technologies (MSST), Jul. 2011.

[14] D. C. Park, "Hot and Cold Data Identification: Applications to Storage Devices and Systems", The University of Minnesota, Aug. 2012.

[15] N. Beckmann, Hans-Peter, Kriegel, R. Schneidr, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", ACM SIGMOD, pp. 322-331, May 1990.

[16] J. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching", Association for Computing Machinery, Vol. 18, No. 9, pp. 509-517, Sep. 1975.

[17] C. Jin, S. De-lin, and M. Fen-xiang, "An Improved ID3 Decision Tree Algorithm",

International Conference on Computer Science & Education (ICCSE), pp. 127-130, Sep. 2009.
[18] DBpedia, http://downloads.dbpedia.org/2016-04/ [accessed: Sep. 08. 2020]
[19] DrugBank, https://www.drugbank.ca/releases/latest [accessed: Sep. 08. 2020]
[20] LinkedGeoData, https://hobbitdata.informatik.unilei-pzing.de/linkedGeodata/Downloads.linkedgeodata.org/releases/ [accessed: Sep. 08. 2020]

저자소개

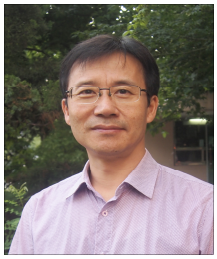
윤 슬 기 (Seulgi Yoon)

2018년 : 경북대학교 컴퓨터학부(학사)
2020년 : 경북대학교 컴퓨터학부(공학석사)
관심분야 : 링크드 데이터, 빅데이터, 머신 러닝

전 석 주 (Seok-Ju Chun)

2002년 : 한국과학기술원 컴퓨터 공학박사
2003년 : 서강대학교 정보통신대학원 외래강사
2004년 ~ 현재 : 서울교육대학교 컴퓨터교육과 교수
관심분야 : 소프트웨어교육, 데이터마이닝, 피지컬 컴퓨팅, 인공지능교육

이 석 룡 (Seok-Lyong Lee)



1984년 : 연세대학교 기계공학과 학사
1993년 : 연세대학교 산업공학과 석사
2001년 : 한국과학기술원 정보 및 통신공학과 박사
1984년 ~ 1995년 : 한국 IBM 소프트웨어 연구소 선임연구원
2002년 ~ 현재 : 한국외국어대학교 산업경영공학과 교수
관심분야 : 데이터마이닝, 멀티미디어데이터베이스, 정보검색

김 상 철 (Sang-Cheol Kim)

1989년 : 고려대학교 경제학사
1991년 : 고려대학교 대학원 경제학석사
2005년 : 독일 브레멘(Bremen)대학 경제학 박사
2011년 3월 ~ 현재 : 한세대학교 사회복지학과 교수
관심분야 : 장애인복지, 노인복지, 사회정책,

이 용 주 (Yongju Lee)

1985년: 한국과학기술원 정보검색전공(공학석사)
1997년 : 한국과학기술원 컴퓨터공학전공(공학박사)
1998년 8월 ~ 현재 : 경북대학교 IT대학 컴퓨터학부 교수
관심분야 : 링크드 데이터, 시맨틱 웹, 빅데이터, 지식 그래프