MongoDB를 활용한 Jena 프레임워크 기반의 분산 트리플 저장소 구현

안진현*, 양성권*, 이문환*, 정진욱*, 김응희*, 임동혁**, 김홍기*
*서울대학교 의생명지식공학연구실
**호서대학교 컴퓨터공학과
e-mail : {jhahncs, sungkwon.yang, munhwanlee, cooluk, eungheekim, hgkim}@snu.ac.kr, dhim@hoseo.edu

An implementation of MongoDB based Distributed Triple Store on Jena Framework

Jinhyun Ahn*, Sungkwon Yang*, Munhwan Lee*, Jinuk Jung*, Eung-Hee Kim*, Dong-Hyuk Im**, Hong-Gee Kim*
*Biomedical Knowledge Engineering Lab. Seoul National University
**Dept. of Computer Engineering, Hoseo University

요 약

웹을 통한 데이터 공유에 대한 관심의 증가로 RDF 트리플 형태의 데이터가 폭발적으로 증가하고 있다. 대용량 RDF 데이터를 저장하고 빠른 SPARQL 질의 처리를 지원하는 트리플 저장소의 개발이 중요하다. 아파치 프로젝트 중 하나인 Jena-TDB는 가장 잘 알려진 오픈소스 트리플 저장소 중 하나로서 Jena 프레임워크 기반으로 구현됐다. 하지만 Jena-TDB의 경우 단일 컴퓨터에서 작동하기 때문에 대용량 RDF 데이터를 다룰 수 없다는 문제점이 있다. 본 논문에서는 MongoDB를 활용한 Jena 프레임워크 기반의 트리플 저장소인 Jena-MongoDB를 제안한다. Jena 프레임워크를 사용했기 때문에 기존 Jena-TDB와 동일한 인터페이스로 사용할 수 있고 최신 표준 SPARQL 문법도 지원한다. 또한 MongoDB를 사용했기 때문에 분산환경에서도 작동할 수 있다. 대용량 LUBM 데이터셋에 대한 SPARQL 질의 처리 실험결과 Jena-MongoDB가 Jena-TDB보다 빠른 질의 응답 속도를 보여줬다.

1. 서론

RDF (Resource Description Framework)[1]는 시맨틱웹 환경 구현을 위한 표준안의 하나로 웹 상의 정보를 표현하는 데에 널리 사용되고 있다. 최근 오픈 데이터에 대한 관심의 증가로 음악 (Discogs), 유전정보 (Gene Ontology), 의약품 (DrugBank) 등 다양한 분야의 RDF 트리플 데이터가 폭발적으로 증가하고 있다[2]. RDF 트리플 데이터는 RDF 트리플의 집합으로, RDF 트리플은 주어 (subject), 술어 (predicate), 목적어 (object)로 구성된 정보 단위이다. 예를 들어, "대한민국의 수도는 서울이다."라는 정보는 (대한민국, 수도, 서울)과 같은 방식으로 RDF 트리플로 표현될 수 있다. SPARQL[3]은 RDF 트리플 데이터에 대한 질의 언어로 관계형 데이터베이스의 질의 언어인 SQL과 유사하다. 일반적으로 트리플 저장소라하면 RDF 트리플 데이터를 저장하고 SPARQL 질의 처리를 지원하는 시스템을 가리킨다.

트리플 저장소는 단일 머신에서 작동하는 시스템과 분산환경에서 작동하는 시스템으로 분류할 수 있다 [5]. 단일 머신에서 작동하는 트리플 저장소로는 Jena-TDB [1], RDF-3X [6], gStore [7] 등이 있는데 대용량 RDF 트리플 데이터를 처리할 수 없다는 단점이 있다. 분산환경 기반 트리플 저장소로는 Hadoop 기반의 HadoopRDF [8]와 HBase 기반의 H_2RDF [9], Jena-HBase [3] 그리고 분산파일시스템 기반의 Clustered TDB [2] 등이 있다. 대용량 데이터도 처리할 수 있다는 장점이 있지만 SPARQL 지원 범위가 제한적이라는 단점이 있다.

본 연구에서는 분산환경을 지원하는 Jena 프레임워크 기반의 트리플 저장소를 구현했다. Jena 프레임워크를 사용했기 때문에 기존 Jena-TDB와 동일한 익숙한 인터페이스를 제공할 수 있고 표준 SPARQL도 지원한다. 또한 파일시스템 대신 MongoDB에 저장함으로써 분산환경에서 작동될 수 있기 때문에 대용량 RDF 트리플 데이터도 처리할 수 있다는 장점이 있다.

2. Jena-MongoDB 구조

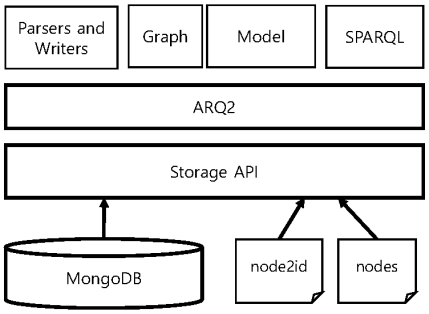
Jena-MongoDB는 분산환경에서 작동하는 트리플 저장

[1] http://www.w3.org/RDF/
[2] http://linkeddata.org/
[3] http://www.w3.org/TR/rdf-sparql-query/

소로 Jena 프레임워크 3.0.0[4]을 사용해서 개발됐다.

기존 Jena-TDB 의 경우 트리플을 B+Tree 기반의 파일에 저장한다. 파일의 종류로는 node2id, nodes, GSPO, GOSP, GOPS, POSG, SPOG 등이 있다. node2id 는 노드 레이블(URI 또는 literal)과 숫자 매핑 파일이고 nodes 는 노드 레이블 자체를 저장하는 파일이다. 노드 레이블에 대한 숫자는 nodes 파일에서 해당 노드 레이블이 기록된 위치(offset)를 가리킨다. 따라서 2 개의 파일을 이용하면 노드 레이블을 숫자로 그리고 반대로 숫자를 노드 레이블로 변환할 수 있다. GSPO 파일은 G (Graph), S (Subject), P (Predicate), O (Object)를 의미하며 숫자로 변환된 트리플을 저장한다. 질의 처리 시 효율적인 트리플 패턴 매칭을 위해 GSPO 의 순서를 바꾼 POSG 등의 파일도 추가로 존재한다. 반면 Jena-MongoDB 는 트리플을 파일이 아닌 MongoDB 에 저장한다.

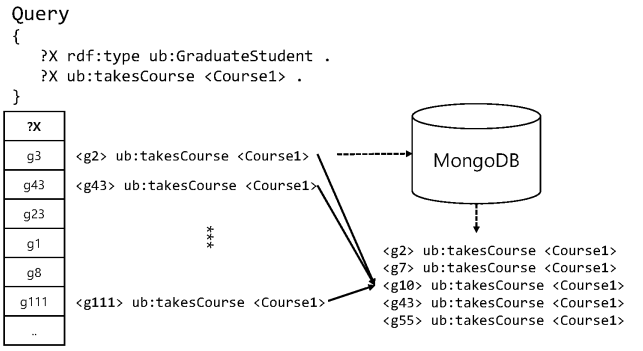
그림 1 은 Jena-MongoDB 의 구조도이다. 얇은 사각형으로 표시된 부분은 Jena 프레임워크이다. Jena-MongoDB 도 node2id, nodes 파일을 사용한다. 하지만 실제 트리플은 MongoDB 테이블(collection)에 저장한다. 파일시스템의 경우 S, P, O 를 구분한 검색 기능을 지원하지 않기 때문에 순서를 바꾼 여러 개의 파일(GSPO, POSG 등)이 필요했지만 MongoDB 의 경우 검색을 지원하기 때문에 GSPO 에 대한 테이블 하나만 있으면 된다. GSPO 테이블은 4 개의 키(G,S,P,O)로 구성됐으며 각 키는 숫자값을 가진다.



(그림 1) Jena-MongoDB 구조도

그림 1 에서 중간부분의 ARQ2 는 기존 Jena ARQ 를 수정한 모듈이다. 기존 Jena-ARQ 의 경우 조인 처리 시 트리플이 존재하는지 여부를 체크할 때 트리플이 기록된 파일(e.g. POSG.dat)을 B+Tree 를 통해 일일이 체크한다. 따라서 디스크 I/O 가 많이 발생할 수 있다. MongoDB 의 경우 독립적인 데이터베이스이기 때문에 네트워크 I/O 도 추가로 발생한다. 따라서, MongoDB 를 활용하기 위해서는 기존 Jena-ARQ 를 개선할 필요가 있다.

그림 2 는 Jena-MongoDB 에서 조인을 처리하는 방식인 ARQ2 를 묘사한 그림이다. 먼저 첫 번째 트리플 패턴를 처리하여 ?X 에 해당하는 URI 들을 얻는다. 각 URI 를 두 번째 트리플 패턴에 대입하여 완성된 트리플을 얻는다. 이 트리플이 실제로 존재하면 주어진 질의에 대한 답이 되고 존재하지 않으면 답이 아닌 것이다. ARQ2 에서는 각 트리플마다 MongoDB 질의를 생성하는 것이 아니라 일정 범위의 트리플 리스트를 가져와서 메모리에 적재한 뒤 트리플 존재 체크는 메모리에서만 수행되도록 한다.



(그림 2) Jena-MongoDB 에서 Join 처리 개념도

이것이 가능하기 위해서는 처음에 ?X 에 해당하는 URI 를 가져올 때 정렬을 해서 가져와야 한다 (실제로는 URI 가 아니라 숫자로 저장됐기 때문에 정렬해서 가져올 수 있다). 예를 들어, 각 URI 에 해당하는 숫자 값이 g3=135, g43=140, ub:takesCourse=201, Course1=3 이었다면, S 가 135 보다 크거나 같고 1135 보다 작으면서 P 는 201 그리고 O 는 3 인 트리플 리스트를 MongoDB 테이블로부터 가져오고 메모리에 적재한다. 그런 후 ?X 에 해당하는 g3=135, g43=140 등이 존재하는지 메모리에서 체크하면 된다.

3. 실험

기존 Jena-TDB와 본 논문에서 제안한 Jena-MongoDB을 3 대의 머신(2.4GHZ, 60GB RAM)으로 구성된 환경에서 실험하였다. Jena-TDB의 경우 단일머신 기반 시스템이기 때문에 머신 하나에서만 작동시켰다. MongoDB 계열 시스템 중 TokuMX[5]를 사용했다. 실험 데이터로는 LUBM100 (약 1 천 4 백만 트리플)과 LUBM1000 (약 1 억 4 천만 트리플)을 사용했다 [4].

<표 1> RDF 데이터 적재 시간 (단위: min.)

	LUBM100	LUBM1000
Jena-TDB-1	6	95
Jena-MongoDB-1	19	491
Jena-MongoDB-2	12	326
Jena-MongoDB-3	13	277

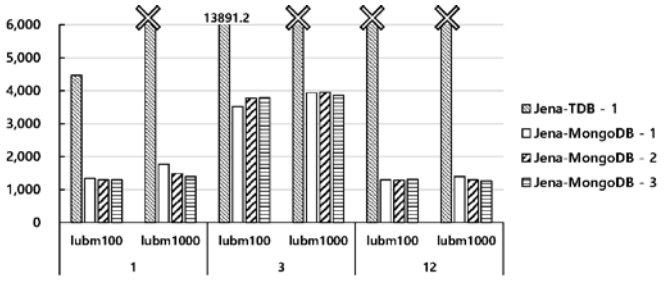
표 1 은 LUBM100/1000 을 각 시스템에 적재하는 데에 걸린 시간이다. Jena-MongoDB-2 는 2 개의 머신을 사용한 Jena-MongoDB 를 의미한다. 샤드 키(Shard Key)는 Predicate 을 지정했다. 모든 경우에 대해 Jena-MongoDB 가 시간이 많이 걸렸다. MongoDB 의 경우 네트워크를 통해 데이터를 전송해야 하기 때문에 더 시간이 많이 걸릴 수밖에 없다. 일반적으로 데이

[4] https://github.com/apache/jena

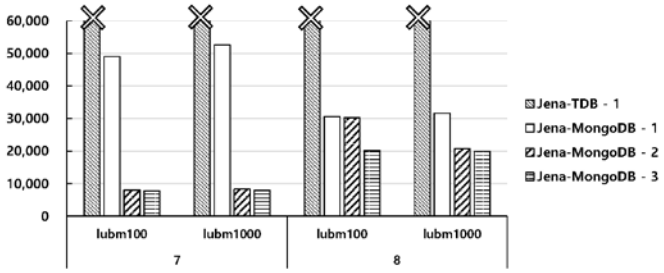
[5] https://www.percona.com/

터를 적재하는 작업은 질의 처리 작업에 비해 덜 빈번하게 발생하기 때문에 데이터 적재 시간보다는 질의 처리 시간이 더 중요하다.

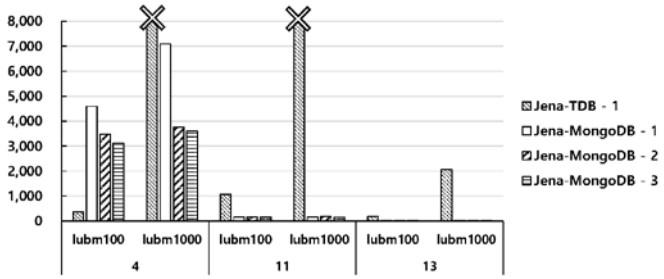
그림 3, 4, 5 는 LUBM100/1000 에 대한 질의 처리 시간으로 5 번 실험 평균이다. LUBM 에서 제공하는 14 개의 질의 중 6 번과 14 번 질의는 한 개의 트리플 패턴으로 구성된 질의이기 때문에 제외시켰다. 또한 5 번과 10 번 질의 경우 1 번 질의와 형태가 동일하기 때문에 제외시켰다. 그래프에서 X 표시는 timeout(3 분)을 의미한다. LUBM100 에 대한 4 번 질의를 제외하고는 모든 경우에 있어서 Jena-MongoDB 의 성능이 월등히 좋다. 2 번과 9 번 질의의 경우 두 개의 시스템 모두 timeout 이었기 때문에 그래프에서는 제외시켰다.



(그림 3) 질의 처리 시간 (단위: ms) (1, 3, 12 번 질의)



(그림 4) 질의 처리 시간 (단위: ms) (7, 8 번 질의)



(그림 5) 질의 처리 시간 (단위: ms) (4, 11, 13 번 질의)

4, 7, 8 번 질의의 경우 머신 추가에 따라 질의 처리 시간이 많이 감소했다. 이 질의들의 경우 트리플 패턴에서 Predicate 의 종류가 다양한 질의들이다. 예를 들어, 7 번 질의의 경우 rdf:type, ub:takesCourse, ub:teacherOf 등의 predicate 를 사용했다. 본 실험에서는 샤드 키로 Predicate 를 사용했기 때문에 질의에 해당하는 트리플들이 각 머신에 고르게 분산됐을 경우 머신이 많을 경우에 질의 처리 시간을 줄일 수 있다. 하지만, 1, 3, 12 번 질의의 경우 머신 추가에 따른 질의 처리 시간의 차이가 거의 없었다. 3 개의 머신을 사용하더라도 질의에 해당되는 트리플들이 고르게 분산되지 못했기 때문에 머신이 추가되더라도 성능의 향상이 없었다.

4. 결론

본 논문에서는 Jena 프레임워크를 활용하여 구현한 분산환경 기반 트리플 저장소를 제안했다. 특히 Jena 프레임워크에 기반해 개발됐기 때문에 기존 Jena-TDB 와 같은 방식으로 Jena API 를 통해 본 저장소를 사용할 수 있고 최신 SPARQL 문법을 처리할 수 있다. 또한 MongoDB 를 활용했기 때문에 대용량 데이터에 대해서도 빠른 질의 처리 시간을 보였다. 향후 MongoDB 에 저장하는 구조를 개선하여 성능을 높일 계획이다. 또한 빈번히 사용되는 질의 형태에 적합한 샤드 키를 선택적으로 정하는 방법을 고안함으로써 실제 사용 환경에서 좋은 성능을 보장하는 정책을 고안할 계획이다.

Acknowledgement

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임. (No. R0101-15-0054, WiseKB: 빅데이터 이해 기반 자가학습형 지식베이스 및 추론 기술 개발)

참고문헌

[1] McBride, Brian. "Jena: Implementing the RDF Model and Syntax Specification." Semantic Web Workshop 2001, Hong Kong, China

[2] Owens, Alisdair, Andy Seaborne, and Nick Gibbins. "Clustered TDB: a clustered triple store for Jena." WWW 2009, Madrid, Spain

[3] Khadilkar, Vaibhav, et al. "Jena-HBase: A distributed, scalable and efficient RDF triple store." ISWC 2012 Posters Track, Boston, USA

[4] Guo, Yuanbo, Zhengxiang Pan, and Jeff Heflin. "LUBM: A benchmark for OWL knowledge base systems." Web Semantics: Science, Services and Agents on the World Wide Web 3.2 (2005): 158-182.

[5] Zoi Kaoudi and Ioana Manolescu. “RDF in the clouds: a survey” The VLDB Journal 24, 1, 2015

[6] Neumann, T., Weikum, G. “The RDF-3X engine for scalable management of RDF data” The VLDB Journal 19(1), 2010

[7] Lei Zou, Jinghui Mo, Lei Chen, M. Tamer Özsu, and Dongyan Zhao "gStore: answering SPARQL queries via subgraph matching" Proceedings of the VLDB Endowment 4(8), 2011

[8] Husain, M., McGlothlin, J., Masud, M.M., Khan, L., Thuraisingham, B. "Heuristics based query processing for large RDF graphs using cloud computing" Knowledge and Data Engineering, IEEE Transactions on 23(9), 2011

[9] Nikolaos Papailiou, Dimitrios Tsoumakos, Panagiotis Karras, Nectarios Koziris, “Graph-Aware, Workload-Adaptive SPARQL Query Caching” SIGMOD’15, May 31–June 4, 2015, Melbourne, Victoria, Australia