ISSN(Print): 2287-5905



http://dx.doi.org/10.3745/KTSDE.2012.1.3.159

# Technique for Concurrent Processing Graph Structure and Transaction Using Topic Maps and Cassandra

Jae-Hyun Shin†

## ABSTRACT

Relation in the new IT environment, such as the SNS, Cloud, Web3.0, has become an important factor. And these relations generate a transaction. However, existing relational database and graph database does not processe graph structure representing the relationships and transactions. This paper, we propose the technique that can be processed concurrently graph structures and transactions in a scalable complex network system. The proposed technique simultaneously save and navigate graph structures and transactions using the Topic Maps data model. Topic Maps is one of ontology language to implement the semantic web(Web 3.0). It has been used as the navigator of the information through the association of the information resources. In this paper, the architecture of the proposed technique was implemented and design using Cassandra - one of column type NoSQL. It is to ensure that can handle up to Big Data-level data using distributed processing. Finally, the experiments showed about the process of storage and query about typical RDBMS Oracle and the proposed technique to the same data source and the same questions. It can show that is expressed by the relationship without the 'join' enough alternative to the role of the RDBMS.

**Keywords : Graph, Graph Database, Topic Maps, Bigdata, Cassandra, NoSQL**

# 토픽맵과 카산드라를 이용한 그래프 구조와 트랜잭션 동시 처리 기법

신 재 현†

## 요 약

SNS, 클라우드, Web3.0과 같은 새로운 IT환경은 '관계(relation)'가 중요한 요소가 되고 있다. 그리고 이들 관계(relation)는 거래, 즉, 트랜잭션을 발생시킨다. 그러나 우리가 사용하고 있는 관계형 데이터베이스(RDBMS)나 그래프 데이터베이스는 관계(relation)를 나타내는 그래프 구조와 트랜잭션을 동시에 처리하지 못한다. 본 논문은 확장 가능한 복잡 네트워크 시스템에서 활용할 수 있는 그래프 구조와 트랜잭션을 동시에 처리할 수 있는 방법을 제안한다. 제안 기법은 토픽맵의 데이터 모델을 응용하여 그래프 구조와 트랜잭션을 동시에 저장하고 탐색한다. 토픽맵은 시멘틱 웹(Web3.0)을 구현하는 온톨로지 언어 중 하나로써, 정보자원들 사이의 연관 '관계(relation)'를 통해 정보의 네비게이터로써 활용되고 있다. 또한 본 논문에서는 컬럼형 데이터베이스인 카산드라를 이용하여 제안 기법의 아키텍처를 설계, 구현하였다. 이는 분산처리를 이용하여 빅데이터 레벨의 데이터까지 처리할 수 있도록 하기 위함이다. 마지막으로 대표적인 RDBMS인 오라클과 제안 기법을 동일한 데이터 소스, 동일한 질문에 대해 저장 및 질의를 하는 과정을 실험으로 보였다. 이는 조인(join) 없이 관계(relation)를 표현함으로써 RDBMS의 역할까지 충분히 대체 가능함을 보이고자 한다.

**키워드 : 그래프, 그래프 데이터베이스, 토픽맵, 빅데이터, 카산드라, NoSQL**

## 1. 서 론

페이스북, 링크드인, 구글+와 같은 새로운 세대의 웹 어플리케이션의 등장은 '소셜 네트워크'라는 유행어를 만들어 냈다. 사실, 소셜 네트워크는 복잡계 네트워크의 일종으로 물리학, 사회학 등에서 활발하게 연구되고 있던 분야이다. 이들은 그래프를 이용하여 네트워크를 구현하고 분석하였다. 이렇게 학계에서만 중요하게 여겨지던 소셜 네트워크가

※ 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 육성지원사업의 연구결과로 수행되었음(NIPA-2012-(H0301-12-3001)).

† 준 회 원 : 성균관대학교 임베디드 소프트웨어학과 석사과정

논문접수 : 2012년 7월 13일
수 정 일 : 1차 2012년 9월 10일, 2차 2012년 10월 9일
심사완료 : 2012년 10월 14일

* Corresponding Author : Jae-Hyun Shin(jhyshin@skku.edu)

일상으로 내려와 사회현상을 만들었다는 점에서 아주 중요한 키워드라고 볼 수 있다. 그리고 소셜 네트워크를 구현하고, 분석하는 기술에는 그래프가 사용됨에 따라 그래프 데이터베이스의 중요성도 한층 높아졌다.

현재까지 나타난 소셜 네트워크 서비스들은 '친구인가 아닌가' 라고 하는 1차원적인 관계만 가지고 있다. 좀 더 복잡한 기능을 가지는 시스템[1]이 연구되고 있지만, 1차원적 관계만 유지된다. 그러나 다음과 같은 시스템을 가정할 수 있을 것이다. SaaS로 CRM을 제공하는 시스템은 하나의 DB에 여러 고객의 정보를 저장한다. 각각의 고객이 가지고 있는 데이터 중에는 동일한 객체를 가리키는 데이터가 존재할 수도 있을 것이다. 그러나 보안을 이유로 고객들은 다른 고객의 데이터에 접근할 수 없다. 그러나 그들 고객들 간에 데이터를 서로 공유하겠다는 합의만 이루어진다면, 중복된 데이터가 없이 기업과 기업, 기업과 고객, 고객과 고객 등 다양한 관계를 표현할 수 있을 것이다. 이는 물리적인 저장 공간이 절약할 수 있는 방법일 수 있다. CRM 뿐만 아니라 동일한 DB를 사용하는 ERP, SCM과 같은 다른 시스템들로 확장된다면, 매우 복잡한 관계망이 생성될 것이다. 물론 이 시스템은 하나의 기업만 쓰는 시스템이 아니다. 이 시스템은 수백, 수천의 기업과 고객, 직원들이 네트워크를 이뤄 사용하는 소셜 클라우드 시스템이 될 것이다. 정보공유를 합의한 사회 구성원 간에는 얼마든지 실현 가능한 시스템일 것이다.

그리고 그 많은 관계 각각에서 발생하는 트랜잭션을 처리한다고 가정해 보자. 이러한 시스템은 확장 가능해야 하며, 매우 복잡한 네트워크를 구현해야 하고, 수많은 종류, 엄청난 양의 트랜잭션을 처리해야 한다. 본 논문은 이러한 시스템에서는 어떤 데이터베이스를 사용할까라는 의문에서 시작되었다. 분명한 것은 이 시스템에는 RDBMS가 핵심적인 데이터베이스가 될 수는 없을 것이다.

구글, 아마존과 같은 선구적인 기업들은 페타바이트 스케일의 엄청난 양의 데이터를 저장하고 처리하기 위한 분산 저장 기술과 NoSQL을 개발하였다. 이러한 분산 기술들은 사실상 표준이 되어 가고 있다[2]. 이러한 새로운 기술이 등장하게 된 것은 RDBMS의 한계 때문이다. RDBMS의 주요한 한계로 지적되고 있는 사항은 다음과 같다. 첫 번째, RDBMS는 데이터웨어하우스 스케일의 데이터 저장/처리, 그리드, 소셜 네트워크, Web2.0, Web3.0, 클라우드 응용기술에 사용하기가 어렵다. 두 번째로 RDBMS 내의 데이터가 증가함에 따라 실행되는 SQL의 실행시간이 비선형적인 특성을 보인다. 또한, 옵티마이저에 의한 쿼리 계획이 불안정하고, 정적인 스키마로 인해 확장에 어려움을 보인다.

본 논문에서는 이러한 RDBMS의 한계를 극복하면서 RDBMS의 역할을 수행할 수 있고, 더불어 관계까지 표현할 수 있는 새로운 형태의 데이터 처리 기법을 제안하고자 한다. 제안 기법은 토픽맵 데이터 모델의 개념을 도입한 그래프 구조와 동시에 트랜잭션을 처리할 수 있다. 우리 기법에서는 표준 토픽맵에 변형을 가했다. 그 부분이 가장 차별적인 부분이다. 그것은 토픽맵에서 가장 흥미로운 구성요소인 어커런스를 토픽뿐만 아니라 연관에도 적용시켰다. 그리고 전통적인 토픽맵이 XML로 저장되는데 반해, 우리 제안 모델은 확장가능하고 빅데이터 레벨의 데이터에도 대응하기 위해 대표적인 컬럼형 데이터베이스인 카산드라를 사용하여 구현한다.

그리고 구체적인 사례를 통해 RDBMS에서는 복잡한 조인이 사용해 '관계'를 만들어 데이터를 찾지만 제안 기법에서는 필요한 '관계'만을 찾아서 데이터를 찾는 과정을 보일 것이다. 이를 통해 제안 기법의 효율성과 RDBMS의 역할을 대체해서 수행할 수 있음을 보일 것이다.

본 논문의 구성은 서론 이하 다음과 같다. 2절에서는 제안 기법에 사용된 개념들을 제시하고, 3절에서 우리가 제안하는 기법의 구체적인 데이터 모델을 설명하고 질의문을 처리하는 방법에 대해 논의할 것이다. 4절에서 실제 데이터를 관계형 데이터베이스에 적용하고, 동일한 데이터를 제안 기법에 적용하는 방법을 구체적인 사례로 보여주겠다. 5절에서는 동일한 질의를 어떻게 처리하는지와 실험을 통해 성능을 비교해 볼 것이다. 마지막으로 6절에서는 제안 기법을 평가하고 향후 연구에 대해 언급하고 본 논문을 마무리 한다.

## 2. 관련 연구

### 2.1 그래프 데이터베이스

그래프 데이터베이스는 소셜 네트워크의 부상으로 요즘 더욱 각광받고, 중요해지고 있는 NoSQL 중 하나이다.

그래프 데이터베이스는 과거 지리학, 생물학, 사회학 등 학계에서 많이 활용되고 있다가 SNS의 등장과 함께 데이터 간 상호연결성의 필요성이 증가함에 따라 일상 생활에서도 활용되고 있는 데이터 모델이다[3]. 그래프 데이터베이스는 이미 많은 연구가 있어 왔다. 그래프를 더욱 효율적으로 사용하기 위해 질의를 편리하게 해주는 연구[4]에서부터 요즘 tera-byte 수준의 빅데이터 그래프에 대한 연구[5][6]에 이르기까지 많은 연구가 수행되어 왔다. 그러나 이러한 연구들은 더 많은 노드와 간선을 효율적으로 처리하기 위한 연구에 그치고 있다.

그래프 데이터베이스는 데이터를 표현하기 위해 노드, 간선, 속성의 세 가지 기본적인 구성을 사용하고 방향성 유무의 특성을 가진다. 그래프 데이터베이스는 이러한 구성 외에는 다른 제약이 없기 때문에 데이터를 데이터베이스의 제약 조건에 맞추기 위한 추가적인 변환 단계가 필요하지 않다. 또한 대다수 프로그래밍 언어와 데이터베이스에서는 관계성을 추론하지만, 관계성 그 자체는 외래키나 포인터를 사용해서 간접적이고, 추상적으로 표현되는 반면, 그래프 데이터베이스에서는 관계는 간선에 의해 직접적으로 표현된다. 이러한 장점으로 인해 애플리케이션을 직관적이고, 이해하기 쉽고, 빠르게 모델링할 수 있다. 어떠한 형태의 데이터도 표현할 수 있기 때문에 새로 발견한 관계나 속성에 따라 스키마를 자연스럽게 발전시킬 수 있다. 우리는 이렇게 관

계에 표현하기 위한 그래프를 연구하던 중에 그래프 데이터베이스로써 토픽맵의 기능에 주목하였다.

### 2.2 토픽맵

토픽맵은 주제 중심 지식의 표현과 상호호환을 위해서 정보의 연계를 통한 정보의 발견에 중점을 두고 있다. 이러한 방식으로 지식정보를 체계적인 구조로 표현해 대용량의 지식정보를 효율적으로 검색하고 관리해 줄 수 있는 해결책으로 활용되고 있다[7]. 토픽맵은 토픽, 연관, 어커런스의 주요 구성으로 그래프와 유사한 구조을 가진다. 토픽맵은 계층/수평 구조에 한계가 없으며, 다양한 관계들을 편리하게 생성/변경시킬 수 있다. 관계형 데이터 모델에 비해 클래스와 인스턴스, 관계 등의 확장이 매우 편리하며, 생성에 한계를 두지 않는다. 특히 우리가 주목한 토픽맵의 요소는 어커런스(occurence)로, 어커런스는 다양한 정보자원과 맵핑하여 데이터를 활용할 수 있다[8].

토픽맵은 토픽 간의 관계를 표현하는 방법이 그래프와 유사하다. 또한 두 개의 토픽들 사이에 2개 이상의 관계를 가질 수 있는 다중 관계도 가능하다. 반면 같은 클래스에 포함된 모든 인스턴스에 동일한 관계들을 정의하기 때문에 저장되는 연관의 숫자가 많아진다. 그리고 토픽과 연관은 매우 복잡한 관계를 가지게 된다.

이렇게 복잡한 관계를 표현하기 위해 표준 토픽맵은 XTM(XML Topic Maps)이라는 문서기반의 표현 언어를 사용한다[9]. 이것은 저장되는 데이터 크기에 상당한 제약을 보였다. 이런 점을 개선하기 위해 토픽맵을 관계형 데이터베이스에 저장하는 연구도 진행된 바 있다[10]. 하지만 [10] 논문에서 10만여개의 토픽으로 이루어진 데이터를 관계형 데이터베이스에서 탐색하는데 10초가 걸렸다. 이는 보완이 필요한 사항이며, 수백만개나 그 이상의 대용량 데이터에서의 성능을 보장하기 위해 우리는 분산 처리/분산 저장 기술을 적용하기 위한 연구를 수행하였다.

### 2.3 카산드라

본 논문에서 수행할 실험에서는 866만여개의 토픽이 저장되고, 연관은 1300만여개 정도 된다. 물론 이 정도 데이터는 어느 관계형 데이터베이스라 해도 문제가 없을 수 있다. 하지만 저장되는 데이터가 급격하게 증가하게 되면 토픽과 연관의 탐색은 관계형 데이터베이스에서 심각한 성능 저하는 예측할 수 없게 된다. 분산처리는 이러한 병목을 해소시켜 줄 것이다.

많은 분산기술이 적용된 NoSQL[11][12][13] 중 가장 중점을 둔 사항은 신속한 확장 가능성과 데이터 저장 구조와 탐색 방식이었다. 토픽과 연관은 짧은 메시지로 구성되어 있다. 이는 SNS 사용하는 패턴과 비슷하여 SNS에서 많이 활용되고 있는 컬럼기반 NoSQL의 방식이 적절할 것으로 고려되었다. 그래서 대표적인 컬럼기반 데이터베이스인 Hbase와 카산드라가 선택되었다. 이 중 카산드라는 P2P방식의 클러스터 구성으로 단일 장애 지점이 없고, 클러스터 구성이 매우 손쉽다는 강점이 있었다. 또한 Hbase와 카산드라를 사용한 벤치마크 테스트[18]에서 카산드라가 쓰기에서 5배, 읽기에서 4배 좋은 성능을 보인 바 있다. 이런 사항들을 고려하여 우리 연구에서는 카산드라를 사용하였다.

카산드라의 데이터 모델은 Fig. 1과 같다. 다중 머신은 하나의 클러스터를 이루고, 클러스터 내에 복수의 키스페이스를 가질 수 있다. 키스페이스 내부에 여러 개의 컬럼 패밀리를 생성할 수 있으며, 관계형 테이블과 유사한 역할을 한다. 컬럼 패밀리는 슈퍼 타입과 일반 타입이 존재한다. 컬럼 패밀리 내부에는 생성개수에 제한이 없는 로우와 마찬가지로 생성개수에 제한이 없는 컬럼과 값을 가진다. 카산드라는 희소 다차원 해시테이블 구조로 볼 수 있다[15].

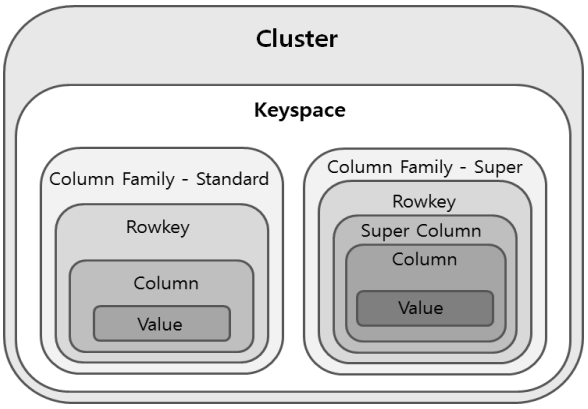


Fig. 1. Cassandra Data Model

카산드라의 로우에는 고유키(로우키)가 있으며, 그 고유키로 데이터에 접근한다. 로우키는 항상 바이트 순서로 정렬된다. 토픽명을 로우키에 부여한다면, 토픽은 사전순으로 저장되고 정렬된다. 또한 이렇게 저장된 데이터가 분산처리 되기 때문에 원하는 토픽을 빠르게 찾을 수 있다. 이것은 사람은 자연어를 그대로 사용하여 검색을 할 수 있고, 컴퓨터는 별다른 변환과정 없이 빠르게 데이터를 찾아 낼 수 있다.

우리는 이러한 토픽맵과 카산드라의 특성을 이용하였다. 토픽맵 데이터 모델을 다차원 컬럼형 데이터베이스 시스템인 카산드라를 이용하여 저장하고 질의한다. 이런 조합의 기술은 스키마가 없는 그래프를 저장하기 편리하고, 토픽맵 질의에 적합한 데이터 정렬 구조를 가지게 된다. 더욱이 카산드라를 사용하면서 빅데이터 사이즈의 그래프 데이터베이스도 처리가 가능해진다. 이제 시작 단계의 구현물이라 한계는 많지만, 관계형 데이터베이스와의 실험에서 탁월한 성능을 보여주었다.

## 3. 제안 기법

우리가 제안하고자 하는 기법의 중요 요소 중 하나인 그래프는 일반적인 그래프 구현방식과 다르다. 노드에 해당되는 토픽과 간선에 해당되는 연관은 카산드라의 슈퍼 컬럼

패밀리(Super Column Family)라 불리는 4차원 해시로 바이트 순으로 저장된다. 그리고 바이트 순으로 분산 저장되어 있는 토픽이나 연관을 빠르게 찾아서 그래프를 구성한다. 이는 연관에 멤버로 지정된 각 토픽들을 그래프처럼 연결하게 된다. Fig. 2와 같이 '홍길동의 친구를 찾으시오.'라는 질의를 처리한다면, 원하는 토픽을 사전순으로 분산 정렬되어 있는 컬럼 패밀리의 로우키에서 찾아내고, 찾아낸 토픽인 홍길동의 고유식별ID를 인출한다. 추후에 설명 되겠지만, 연관명 규칙에 따라 '친구_홍길동ID'인 연관을 찾으면, 해당 연관의 슈퍼 컬럼에서 친구 정보를 인출할 수 있다. 이와 같은 방식으로 그래프를 구성하게 된다.

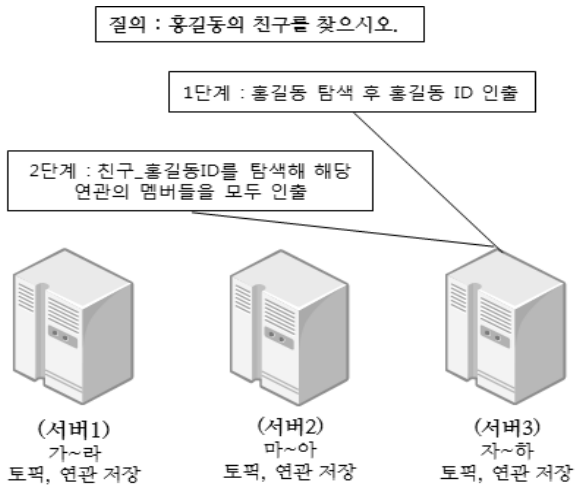


Fig. 2. Querying Processing

그래프 구성방식과 더불어 중요한 특징은 토픽들 간의 관계에서 발생하는 트랜잭션을 저장한다. 이는 다른 그래프데이터베이스에서는 볼 수 없는 방식이다. 예를 들어 A사가 B사에 부품을 공급하고, B사가 C사에 완제품을 공급하는 공급망이 있다면, 각 관계의 어커런스에 각 관계의 트랜잭션을 저장하는 방식이다. 더 많고 복잡한 공급망이 있다하여도 하나의 데이터베이스에 저장할 수 있다. 자세한 설명은 각 주제에서 구체적으로 다루겠다.

### 3.1 데이터 모델

토픽맵 데이터 모델의 개념을 사용하긴 하였으나, 이 논문에서는 ISO 표준에 있는 기능들을 제외하기도 하였고, 표준에 없는 기능을 추가하기도 하였다[8]. 토픽맵과 관련된 문법 키워드는 XTM1.0을 참조하였다[9].

#### 1) 토픽(Topic)

토픽맵에서 토픽은 엔터티-관계(E-R) 모델에서 엔터티와 인스턴스를 같이 포함하고 있다. 예를 들어, E-R모델에서 '학생'은 엔터티가 되고, 학생 중에 학번이 123이고 이름이 '홍길동'인 속성을 가지는 특정 학생은 엔터티의 인스턴스가 된다. 그러나 우리의 모델에서는 '학생'도 토픽이고, '홍길동'도 토픽이다. 단, 학생과 홍길동은 인스턴스 관계(instance-of)라는 연관을 가질 수 있다.

제안 기법에서 토픽은 토픽명, 토픽ID, 속성으로 구성된다. Fig. 3과 같은 구조로 카산드라에 저장된다.

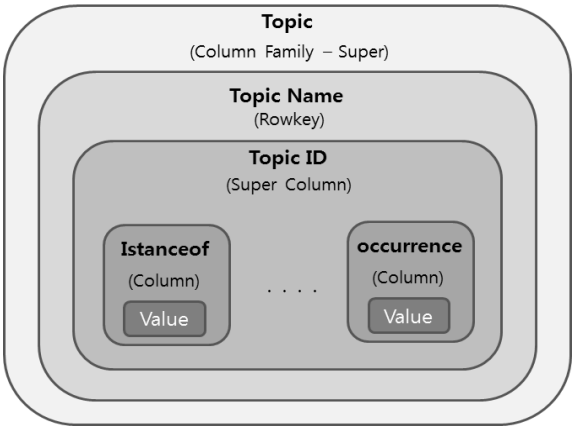


Fig. 3. Structure of Topic

제안 기법에서 토픽은 카산드라의 슈퍼 컬럼 패밀리로 저장된다. 자연어로 표현되는 토픽명이 로우키로 사용되고, 중복가능하다. 로우 내부에 슈퍼 컬럼이 존재하는데, 슈퍼 컬럼명으로 토픽ID가 저장되고, 토픽 컬럼 패밀리에서 유일한 값을 가지는 기본키이다. 슈퍼 컬럼 내부에는 토픽의 속성인 'intanceof', 'identifiertype', 'identifier', 'scopetype', 'scope', 'occurrencetype', 'occurrence'가 하위 컬럼으로 저장되고, 그 값이 컬럼의 값으로 저장된다. instanceof, occurrencetype, occurrence는 필수 속성이지만, 나머지는 선택적으로 사용할 수 있다. 이들 속성은 토픽의 의미를 구성하는 역할을 수행하며 다음과 같은 용도를 가지고 있다. Fig. 4와 같은 관계를 가지고 있다.

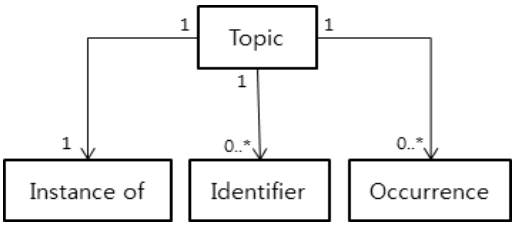


Fig. 4. Topic and Properties

• instanceof : 가장 인접한 상위 엔터티의 토픽명이 들어간다. 상위 엔터티가 없는 최상위 토픽인 경우에는 class라고 기록한다. '배'라는 토픽명을 가진 토픽이 3개가 있다면, 모두 '배'라는 로우에 기록된다. 그리고 각각 토픽ID로 슈퍼컬럼에서 구분되어 저장되지만, 이것으로 이들을 구분할 수 없다. 이들은 instanceof로 구분가능하다. 각각의 토픽에는 instanceof 값으로 운송수단, 과일, 인간신체기관으로 각각 기록되어 있다면, '배'라는 토픽들의 의미를 구분할 수 있다.

• identifiertype, identifier : 선택적으로 기록할 수 있다. 예를 들면, 같은 이름을 쓰는 사람들의 경우, 같은 토픽명을 사용하기 때문에 같은 로우에 저장된다. 같은 로우에서 각각의 슈퍼 컬럼을 가지겠지만, instanceof 값이 '사람'이라는 같은 값을 가진다. 따라서 instanceof로는 구분이 불가능하다. identifiertype의 값에 구분하고자 하는 속성명 넣고, 그 값을 identifier에 기록한다. 예를 들어, indentifiertype 값에 '주민번호'라고 기록하고 identifier 값에 주민번호를 기록한다. 이 속성으로 같은 이름의 사람을 구분할 수 있다.

• occurrencetype, occurrence : 토픽맵에서 중요한 구성요소 중 하나이다. occurrence는 해당 토픽의 상세한 정보를 가지고 있는 저장소의 주소이다. 이 주소값은 카산드라 내부의 키스페이스명에서부터 로우키값까지 어플리케이션에서 필요에 의해 정의할 수 있다. 직접적인 주소를 가지고 있어서 스캔과정 없이, 바로 그 값을 찾을 수 있다. occurrencetype은 그 주소가 어떤 형태의 주소인지를 나타낸다. 저장소는 카산드라 내부일 수도 있고, 다른 파일시스템, 혹은 외부의 관계형 데이터베이스일 수도 있다.

Fig. 5는 토픽 의미 구분 예시를 나타내었다.

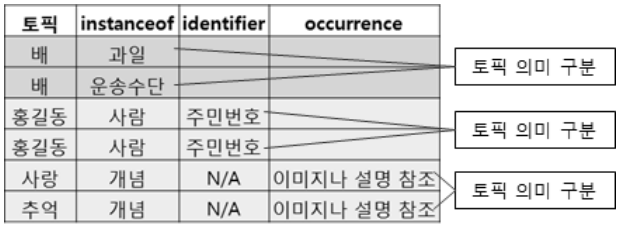


Fig. 5. example for Topic Divide

• scopetype, scope : 이 속성은 어플리케이션에서 토픽을 어느 범위까지 사용할지를 필요에 따라 선택적으로 사용할 수 있다. 또한 필요에 따라 인덱스를 부여해 2개 이상의 scope을 가질 수 있다. 예를 들어, scopetype이 'language'이고 scope이 'kor'라면 한국어 버전의 어플리케이션에서만 사용되는 토픽으로 정의할 수 있다.

위와 같은 속성들은 인간과 컴퓨터 모두가 토픽의 의미를 구분할 수 있게 한다.

2) 연관(Association)

연관은 토픽 간에 관계를 나타낸다. E-R 모델에서는 엔터티들 간의 관계(relation)가 인스턴스들의 관계를 대표한다. 그러나 제안 기법에서 연관은 관계와 달리, 각 토픽은 각자 고유한 연관을 가진다.

연관은 주로 '주어토픽 - 동사 - 목적어토픽'으로 표현된다. 여기서 동사에 해당되는 부분이 연관이다. 연관은 방향성을 가지고 있다. 이 방향은 연관을 표현하는 동사의 형태로 구분할 수 있다. 예를 들어, '홍길동은 - 주문한다. - A핸드폰을'에서 '주문한다'는 '홍길동 토픽과 A핸드폰 토픽의 관계를 나타내고 있다. 두 토픽의 관계는 다른 방향으로도 가능하다. 'A핸드폰은 - 주문된다. - 홍길동에 의해'와 같이 표현할 수 있다. 연관은 반드시 두 개의 토픽을 멤버로 가진다.

연관이 저장되는 방법은 Fig. 6과 같다. 연관도 토픽과 마찬가지로 슈퍼 컬럼 패밀리에 저장된다. 로우키로 사용된 AssociationID는 '연관명_목적어토픽ID'으로 표현된다. 위의 예라면, '주문한다_TABC01234(A핸드폰의 토픽ID)'가 된다. 슈퍼컬럼명으로 주어토픽의 토픽ID가 기록되어 로우키와 슈퍼컬럼명이 복합으로 연관 컬럼 패밀리의 고유키를 만든다.

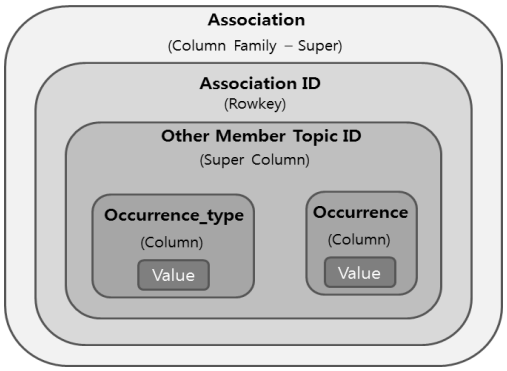


Fig. 6. Structure of Association

사람이라는 토픽의 인스턴스인 홍길동은 다른 토픽과의 관계로 그 역할이 정해진다. B회사라는 토픽과 고용관계에 있다면, 홍길동은 B회사의 직원이 된다. 그리고 C회사와 고객관계에 있다면, 홍길동은 C회사의 고객이 된다. 이렇게 관계는 역할을 설명할 수 있기 때문에 역할을 따로 정의할 필요가 없어진다.

또한 두 토픽간의 관계에서는 트랜잭션이 발생한다. A사람과 B쇼핑몰회사는 주문단위의 트랜잭션이 발생한다. B쇼핑몰회사가 소유한 개별 상품 토픽과 A사람과의 관계에서는 구매라는 트랜잭션이 발생한다고 볼 수 있다. 여기에 우리의 제안 모델이 표준 토픽맵과 다른 부분이 발생한다. 고유한 연관은 두 토픽간의 트랜잭션을 기록하기 위해 토픽처럼 occurrence를 가진다. 해당 어커런스에는 두 토픽 간에 발생한 트랜잭션을 기록한다. 이것은 표준 토픽맵에는 없는 개념이다. 이 어커런스도 마찬가지로 occurrence_type을 통해 카산드라 뿐만 아니라, 외부 저장소를 가리킬 수 있다.

슈퍼 컬럼 내부에는 연관을 구분하기 위한 별도의 구별자가 필요하지 않으므로 occurrence_type과 occurrence만 저장된다. 이 두 속성은 토픽과 동일한 기능과 역할을 한다.

3) 어커런스(Occurrence)

어커런스는 토픽과 연관의 속성으로 이미 많은 설명을 하였다. 토픽의 어커런스는 토픽의 의미나 속성을 확장할 수 있고, 연관의 어커런스는 연관에 포함된 멤버들의 트랜잭션을 나타낼 수 있다.

어커런스는 토픽, 연관의 성격에 따라 저장방법, 저장 구조, 저장위치를 모두 달리할 수 있다. 저장 위치는 필요에 따라 카산드라 외부에 저장될 수도 있다. 카산드라 내부에

저장되는 경우라면, 데이터의 성격에 따라 슈퍼 컬럼 또는 일반 컬럼에 저장되어질 수 있다. 로우키에 어커런스 주소 값이 사용되는 제약 외에는 로우 내부에는 어떤 구조라도 상관없다. 일반적인 컬럼 지향 데이터베이스와 동일한 구조의 저장소가 될 것이다. 어떤 경우라도 로우키를 직접 가리키기 때문에 탐색시간은 매우 짧다.

### 3.2 질의(Querying)

제안 기법에서 질의는 카산드라 질의 모델을 사용한다. 실제, 실험에서 질의를 구현할 때, 카산드라의 대표적인 JAVA 클라이언트 API인 '쓰리프트(Thrift)'를 이용하였다.

카산드라와 RDBMS의 질의 방식에는 아래와 같은 차이가 있다[15].

- 업데이트 질의 부재
- 쓰기에서 레코드 레벨 원자성 제공
- 서버측 트랜잭션 지원 부재
- 중복 키 부재

카산드라에는 조정 가능한 일관성(Consistency) 레벨이 있다. 레벨이 높을수록 더 많은 노드가 응답하기 때문에 각 복제본에 있는 값이 같음을 보장한다는 의미이다. 만약 두 노드가 응답한 타임스탬프가 다르면, 가장 최신의 타임스탬프의 값을 반환한다. 일관성 레벨에는 ZERO, ANY, ONE, QUORUM, ALL이 있다.

카산드라의 기본적인 쓰기 속성이 몇 가지 있다. 첫째, 멤테이블과 SSTable을 사용으로 디스크 읽기나 찾기를 수행할 필요가 없다. 두 번째로, 모든 쓰기는 추가전용(Append-only)이다. 셋째, 커밋 로그와 힌트 핸드오프 디자인 덕분에 항상 쓰기가 가능하며, 컬럼 패밀리 내부에서 쓰기는 항상 원자적이다.

카산드라 클라이언트가 데이터를 읽기 원한다면, 클러스터에 있는 노드 중 아무거나 연결해서 읽기를 수행할 수 있으며, 그 데이터가 복제본인지 알 필요가 없다. 클라이언트가 읽으려는 데이터가 없는 노드에 연결 되였다면, 연결된 노드는 토큰 범위를 식별해 데이터를 가지는 노드에서 데이터를 읽어오는 코디네이터 노드로 동작하게 된다.

우리의 제안 모델에서 읽기는 카산드라 API를 활용하여 질의마다 질의의 답을 가져올 수 있는 프로그램을 만들어야 한다. 이는 대부분의 NoSQL의 특성이다. SQL같은 편리함은 없지만, 사용자가 직접 최적화를 할 수 있기 때문에 빠른 질의처리가 가능하다. 상용 RDBMS의 튜닝에 소요되는 비용과 시간을 감안한다면, 질의를 직접 프로그램하는 것이 오히려 더 이득일 수 있다.

우리 모델에서 질의를 구현할 때 사용하는 주요 API는 다음과 같다[16].

- get : 하나의 컬럼값을 찾는 API이다.
- slice_range_get : 하나의 로우에 있는 슈퍼컬럼이나 컬럼들을 찾는 API이다. 'slice range' 컬럼의 범위를 의미한다.
- key_range_get : 여러 로우에 동일한 컬럼을 찾는 API이다. 'key range'는 로우키의 범위를 의미한다.

이러한 주요 API뿐만 아니라 이외의 API들의 조합으로 제안 기법의 질의문 처리 프로그램이 프로그래밍할 수 있다. 또한 오픈소스인 카산드라를 이용하여 제안 기법에 최적화된 API를 개발하여 새로운 데이터베이스를 개발할 수도 있을 것이다.

## 4. 사례 연구

이번 장에서는 실제로 관계형 데이터베이스에서 사용하는 데이터 집합을 활용하여 해당 스키마가 제안 기법에서 어떻게 적용되는지 살펴보겠다. 의미관계가 명확한 실제 데이터를 사용하는 것이 의미관계를 효율적으로 보여주는 토픽맵으로 옮기는 것이 효율적이다. 하지만 현실적인 어려움으로 인해, 대표적인 관계형 데이터베이스 벤치마크 도구인 TPC-H에서 사용하는 데이터 스키마와 데이터 집합을 사용하여 제안 기법에 적용하는 방법을 제시하겠다.

### 4.1 TPC-H 스키마

실제로 RDBMS에서 사용하는 데이터를 이용하는 것이 좋겠지만, 데이터를 구하는 것에 제약이 있기 때문에 우리가 사용할 데이터 집합은 RDBMS 벤치마킹 도구로 유명한 TPC-H에서 사용하는 데이터 집합이다.

TPC-H 스키마는 객체들의 관계로 이루어진 스키마는 존재하지만, 그 의미 관계가 정확하지 않다. 테이블 내에 삽입되는 데이터 중 기본키는 순차적인 순서로 입력되지만, 나머지 컬럼들은 의미 없는 값들로 채워진다. Fig. 7은 TPC-H에서 사용하는 데이터 집합의 스키마이다[17]. 테이블 밑에 숫자는 테이블의 레코드수이다. 전체 데이터 집합의 크기는 대략 1GB이다. 자세한 크기는 4장 실험에서 설명

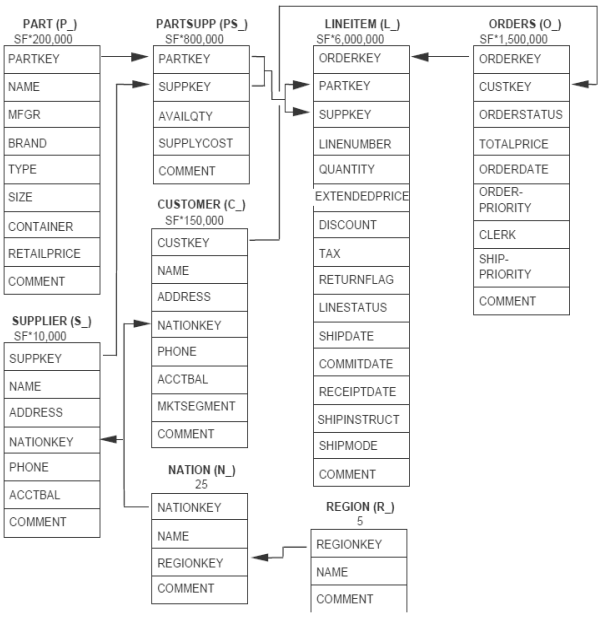


Fig. 7. TPC-H Schema

한다. 이 스키마로 알 수 있는 시나리오는 각 대륙별, 국가별로 분포해 있는 고객과 공급사가 있고, 각 공급사로부터 부품을 공급받아 고객에게 주문을 받아 판매한다는 내용으로 해석할 수 있다.

### 4.2 제안 기법의 스키마

2장에서 설명한 바와 같이 제안 기법의 스키마는 (Figure 5)와 같다. 이 스키마는 의미를 나타낼 수 없다. 이 스키마 내부에 Fig. 7의 의미를 가지는 토픽과 연관, 어커런스가 담기게 된다. 이 절에서는 이 3가지 개념을 이용하여 TPC-H 스키마의 의미를 모두 담는 과정을 보이고자 한다.

```
KeySpace : TopicMaps
Column Family : Topic (column_type : super)
Column Family : Association(column_type : super)
Column Family : Occurrence_N(column_type : standard)
Column Family : Occurrence_S(column_type : super)
```

Fig. 8. Schema

Fig. 8의 스키마에서 도출한 클래스 토픽은 nation, region, person, company, part, lineitem 이다. 그리고 관계 토픽은 belong-to, live-in, is-in, supplier-of, customer-of, manufactured-by, consist-of, ordered-by, order-to이다. 그리고 연관은 Fig. 9같이 정의된다.

```
nation - belong-to - region
person - live-in - nation
company - is-in - nation
company - supplier-of - company
person - customer-of - company
part - manufactured-by - company
lineitem - consist-of - part
lineitem - ordered-by - person
person - order-to - lineitem
```

Fig. 9. Association Definition

customer 테이블과 supplier 테이블은 각각 person과 company 엔터티로 변환하였다. 고객은 person과 company와의 관계로 정의될 수 있기 때문이다. 그리고 company의 인스턴스 중에는 공급자 외에 이 데이터 소스의 소유자격인 TPCH corp.이라는 가상의 회사를 추가해 person의 인스턴스들과 연관관계를 만들었다. 그리고 partsupp 테이블과 order 테이블은 엔터티로 선정되지 않았다. 각 공급자가 제공하는 부품은 부품명이 같더라도 다른 인스턴스가 되어야 하기 때문에 part와 partsupp 테이블을 합친 형태로 '부품명_공급자명'을 형식을 같는 부품으로 변환하였다. 마지막으로 order 테이블의 데이터는 고객과 회사간에 트랜잭션으로 person 과 company의 연관에 어커런스 되도록 처리하였다.

각 테이블들의 저장되어 있는 레코드들의 name 컬럼, 즉, 실제 사용되는 이름들이 로우키의 값으로 사용된다. 그리고 나머지 컬럼들의 값들은 어커런스를 통해 저장된다. 그러나 인덱스 형태의 기본키 컬럼들은 토픽맵에 저장되지 않는다.

저장되는 데이터의 예를 몇 가지 살펴보겠다.

```
AFRICA, A0000001, instanceof, region
AFRICA, A0000001, occurrence_type, comment
AFRICA, A0000001, occurrence, special Tiresias about the furiously even dolphins are furi
```

Fig. 10. AFRICA Topic

Fig. 10은 region의 인스턴스인 AFRICA 토픽의 데이터 입력 포맷이다. 로우키값(토픽명)은 'AFRICA'이고, 슈퍼컬럼명(토픽ID)은 'A0000001'이다. 'instanceof' 컬럼의 값은 region이다. region 테이블이나 nation 테이블은 주키와 참조키 컬럼을 제외하면 name과 comment 컬럼만 남게 된다. 그래서 별도의 어커런스 공간에 저장하는 대신, occurrence_type 컬럼값에 comment를 주고 occurrence 컬럼값으로 comment 내용을 입력하였다.

```
belong-to_A0000001, B0000001, occurrence_type, none
belong-to_A0000001, B0000001, occurrence
belong-to_A0000001, B0000006, occurrence_type, none
belong-to_A0000001, B0000006, occurrence
belong-to_A0000001, B0000015, occurrence_type, none
belong-to_A0000001, B0000015, occurrence
belong-to_A0000001, B0000016, occurrence_type, none
belong-to_A0000001, B0000016, occurrence
belong-to_A0000001, B0000017, occurrence_type, none
belong-to_A0000001, B0000017, occurrence
```

Fig. 11. Nation Association belong to AFRICA

Fig. 11는 'nation - belong-to - region'의 연관정의에 따라 AFRICA에 속한 국가를 표현한 데이터 입력 내용이다. 로우키에 'belong-to_A0000001'값은 연관ID이고, 슈퍼 컬럼명인 'B0000001'는 해당 연관의 주어인 연관의 또 다른 멤버이다. 지역과 국가의 관계에는 별다른 트랜잭션이 없기 때문에 어커런스 타입은 none이고, 어커런스는 빈 값을 갖는다.

이제 어커런스가 존재하는 토픽을 살펴보겠다.

Fig. 12은 part의 인스턴스 중 하나를 저장하기 위한 입력 데이터이다. AFRICA와 같은 구조지만, 어커런스 타입에 'cassandra address'가 나와 있고, 어커런스 값으로 'N|TOC_E0000001'를 가진다. 이것은 Occurrence_N 컬럼 패밀리에, 로우키가 'TOC_E0000001'인 데이터를 찾아가라는 의미이다. 해당 로우에는 관계형 테이블이 가지는 컬럼들이 컬럼/값으로 저장되어 있다.

ghost blue olive sky gainsboro_Supplier#000007514, E0000001, instanceof, part
ghost blue olive sky gainsboro_Supplier#000007514, E0000001, occurrence_type, cassandra address
ghost blue olive sky gainsboro_Supplier#000007514, E0000001, occurrence, N|TOC_E0000001

Fig. 12. part Instance Topic

마지막으로 어커런스가 있는 연관을 살펴보겠다.

order-to_F1801806, D0056299, occurrence_type, cassandra address
order-to_F1801806, D0056299, occurrence, S|AS_ORDERTO_F1801806

Fig. 13. order-to Occurrence

Fig. 13은 lineitem과 person의 관계인 'order-to' 연관을 입력하는 모습이다. 이는 어커런스가 있는 토픽과 마찬가지 구조를 가진다. 어커런스값에 'S'는 Occurrence_S를 가리킨다. 해당 주소에 가면 Fig. 14와 같은 데이터가 입력되어 있다. 로우값은 주소에 표시되어 있는 값이며, 연관의 어커런스는 토픽의 어커런스와 다르게 다른 멤버의 토픽ID를 슈퍼컬럼값으로 갖는다. 그리고 컬럼값에 들어가는 값들은 lineitem 테이블에 있는 값들이다.

AS_ORDERTO_F1801806,D0056299,extendedprice, 57020.4
AS_ORDERTO_F1801806,D0056299,quantity,47
...

Fig. 14. order-to Occurrence

이상의 방식으로 RDBMS에 저장되는 데이터를 제안 기법에 따라 입력하였다.

## 5. 실 험

### 5.1 실험 환경

실험은 대표적인 RDBMS인 오라클과 제안 기법을 동일한 질의에 대한 응답속도를 비교해 보았다. 실험환경은 Table 1과 같다.

오라클은 1대의 머신에서 실험하고, Cassandra는 2대의 머신을 클러스터로 묶어서 실험하였다. 그리고 Cassandra의 일관성 레벨은 'ONE'으로 쿼리에 응답한 첫 번째 노드의 레코드를 즉시 반환하도록 하였다. 데이터 복제계수는 0으로 복제를 하지 않았다. 2대의 PC에는 각각 데이터를 나눠 담고 있는 것이다.

Table 1. Experiment Setup

| | | |
|---|---|---|
| 컴퓨터1 | CPU | Intel Core2 Quad Q6600 2.40GHz |
| | 메모리 | 5.00GB |
| | 운영체제 | Windows 7 Ultimate K_sp1 64bit |
| | Oracle | Oracle Database 11g Release 11.2.0.1.0 - 64bit |
| | Cassandra | cassandra-1.1.0, thrift-0.7.0 |
| 컴퓨터2 | CPU | Intel Dual-Core E6600 3.00GHz |
| | 메모리 | 4.00GB |
| | 운영체제 | ubuntu 12.04 32bit, kernel 3.2.0-24 |
| | Cassandra | cassandra-1.1.0, thrift-0.7.0 |

본 실험을 통해 두 가지 사실을 제시하고자 한다. 첫 번째는 동일한 데이터 소스를 이용하여 동일한 의미를 가지도록 데이터를 저장했을 때 오라클과 카산드라에 저장되는 데이터 용량을 비교하였다. 두 번째는 저장된 데이터를 이용하여 동일한 질의에 대한 성능을 비교하는 실험이다.

### 5.2 질의 및 성능 비교

두 데이터베이스에서 사용할 질의 내용은 Fig. 15과 같다.

**(질의)** 'ASIA'지역 고객이 공급사 'Supplier#000005538'이 공급한 part 'almond antique aquamarine indian chartreuse'를 구매한 매출 총액은?

Fig. 15. Query

위 질의를 이용하여 오라클은 SQL을 사용하여 질의에 대한 답을 찾고, 제안 기법은 별도의 질의 응용프로그램을 이용하여 답을 찾는다. 답을 제시하는 걸리는 시간을 측정하여 그 성능을 비교한다.

1) SQL문

해당 질의에 대한 SQL문은 아래와 Fig. 16과 같이 나타낼 수 있다. 오라클의 경우 기본키 외에 다른 인덱스를 만들지 않았다. 또한 어떠한 최적화 기법을 사용하지 않았다. 이것은 이 실험은 관계형 데이터베이스와 제안 기법과의 성능을 비교함이 아니기 때문에 오라클의 최고 성능을 내는 환경을 굳이 만들지 않았다는 것을 알려둔다.

SQL 도구로 오라클에서 제공하고 있는 'SQL Developer'를 사용하였고, 해당 도구에 표시되는 경과시간을 10회에 걸쳐 측정하여 최대값, 최소값을 제외한 값의 평균값을 구하였다.

2) 제안 기법 질의 알고리즘

제안 기법은 질의를 실행하기 위해서는 질의에 해당되는 알고리즘으로 프로그래밍을 해야 한다. Fig. 15의 질의에 대한 동일한 대답을 하기 위한 알고리즘은 Fig. 17와 같다.

질의의 답을 찾는 과정은 다음과 같다. 우선, 질의에 포함되어 있는 ASIA라는 키워드와 파트명과 공급사명으로 토픽을 찾는다. 두 번째, ASIA 내에 포함되어 있는 국가를 belong-to 연관을 통해 찾는다. 그리고 제시한 공급사의 부

```
select sum(ee.l_extendedprice * ee.l_quantity)
  from region aa,
       nation bb,
       customer cc,
       orders dd,
       lineitem ee,
       part ff,
       supplier gg
 where aa.r_regionkey = bb.n_regionkey
   and bb.n_nationkey = cc.c_nationkey
   and cc.c_custkey = dd.o_custkey
   and dd.o_orderkey = ee.l_orderkey
   and ee.l_partkey = ff.p_partkey
   and ee.l_suppkey = gg.s_suppkey
   and aa.r_name = 'ASIA'
   and ff.p_name = 'almond antique aquamarine indian
chartreuse'
   and gg.s_name = 'Supplier#000005538';
```

Fig. 16. Query SQL

1). 토픽명이 'ASIA'인 region 토픽 찾기
2). 토픽명이 'partname_Supplier#000005538'인 part 토픽 찾기
3). ASIA 토픽과 belong-to 연관 관계에 있는 nation 토픽 찾기
4). 해당 Part 토픽과 consist-of 연관 관계 있는 LineItem 토픽 찾기
5). 해당 LineItem의 order-to 연관을 이용하여 주문한 person 토픽 찾기
6). LineItem을 주문한 person 토픽 중 3)에서 찾은 nation 토픽에 live-in 연관이 있는 person 토픽 찾기
7). 6)에서 찾은 person 토픽과 LineItem 토픽의 order-to 연관들의 어커런스 주소 가져오기
8). 7)의 어커런스에서 원하는 매출 관련 컬럼 가져오기

Fig. 17. Query Cassandra Algorithm

품을 사용하고 있는 LineItem을 consist-of 연관을 이용해 찾는다. 그리고 나서, 해당 LineItem을 주문한 사람을 order-to 연관을 이용해 찾는다. 마지막으로 ASIA국가에 살고 있는 사람을 찾아서 그 사람들의 주문에서 어커런스의 주소를 얻어 와서 해당 어커런스에서 매출관련 컬럼 값들을 찾아 결과를 만들어 낸다.

Fig. 17의 알고리즘은 각 클래스들의 인스턴스 토픽의 개수를 알고 있다고 가정하였다. 따라서 연관의 개수가 작은 쪽의 클래스를 선택하여 알고리즘을 만들었다. 알고리즘에서 알 수 있듯이 우리의 제안 모델은 조인이 없다. 토픽을 찾고 연관을 찾는 과정을 통해서 원하는 질의의 답을 찾아나간다.

```
Topic region = findTopic("ASIA");
Topic part = findTopic("partname_Supplier#000005538");
belongID = "belong-to_" + region.topicID;
List<Association> belongs;
belongs = findAssociations(belongID);
List<String> nations;
for(Association a : belongs)
        nations.add(a.memberTopicID);
```

Fig. 18. Query Cassandra Program

추후에 각 클래스의 인스턴스의 개수를 메타데이터로 가지고 있고, 양방향의 연관을 만들어 저장한다면, 시스템 자체적으로 최적화를 수행할 수도 있을 것이다.

Fig. 18는 알고리즘을 JAVA로 구현한 프로그램 코드의 일부이다. 이는 Fig. 17의 알고리즘의 1), 2), 3)에 해당되는 코드이다. 각 메소드는 카산드라 API를 사용하여 구현하였다. 그리고 이 프로그램은 모든 토픽의 사용과 토픽 간의 관계가 정의되어 있다는 가정에서 질의문을 작성하였다고 가정한다.

질의문 경과시간은 IDE 도구인 이클립스에서 프로그램의 시작과 끝에 로그를 남겨 명령창에 표시되는 시간의 차를 계산하였다. 경과시간을 10회에 걸쳐 측정하고, 오라클과 마찬가지 방법으로 평균을 구했다.

3) 결과

실험 결과 두 가지 질의실행 결과 동일한 '5630399.88'값을 제시하였다. 경과시간은 오라클 SQL은 8.73초가 나왔고, 카산드라 프로그램은 0.26초가 나왔다. 물론, 오라클의 경우 질의 SQL에 적합한 인덱스를 만들어주고, 조인 방법 등에서 최적화를 적용하면, 지금 나온 결과의 1/3 수준 이하로도 나올 수 있을 것이다. 그러나 실험 결과에서 보여주고자 했던 것은 제안 기법으로 동일한 데이터 소스에 대해 RDBMS와 동일한 대답을 내줄 수 있고, RDBMS 이상의 성능도 보여준다는 점이다.

## 6. 결론 및 향후 연구과제

소셜 네트워크의 등장으로 시스템에서 '관계'가 아주 중요한 요소가 되었다. 뿐만 아니라 사회관계망이 복잡해 지면서 사회구성원들 간의 데이터교환이 많아지고 있다. 하나의 시스템에서 한 종류의 트랜잭션을 처리하는 방식은 다른 트랜잭션과의 관계를 고려해야 한다는 요구사항을 반드시 해결해야 할 것이다. 본 논문은 관계와 트랜잭션을 효과적으로 관리하기 위한 방법을 제안 하였다. 또한 하나의 시스템에서 사용하는 RDBMS의 대체DB로써 사용가능함을 사례와 실험을 통해 보여주었다.

RDBMS의 등장은 시스템 개발에 획기적인 진보를 이룰 수 있는 기회가 되었다. 하지만 시스템에 누적되는 데이터

가 많아지고, 시스템 간의 데이터 교환, 네트워크가 복잡해지면서 RDBMS의 한계를 보였고, 이에 대한 대안으로 많은 NoSQL이 등장하게 되었다. 물론 이러한 한계가 있다고 해서 RDBMS가 사라질 것이라 생각하는 사람은 없을 것이다. 그것은 한계 내에서는 탁월한 성능과 편리한 사용성을 가지고 있기 때문이다.

RDBMS의 한계 중 한 가지는 최초 설계 이후 시스템 확장에 너무나 많은 비용이 든다는 점이다. 빅데이터, 클라우드를 넘어 이제는 소셜 클라우드가 언급되고 있는 시대이다. 이처럼 시스템이 복잡해지고, 이종 시스템간의 네트워크가 증가되는 환경에서는 RDBMS가 효율적이지 않을 수 있다. 그래프데이터베이스는 이런 시스템 확장이 빈번한 시스템에서 매우 유용할 것으로 보인다.

제안 기법은 이런 RDBMS의 한계를 극복할 수 있으면서 그래프의 특성을 가지고 복잡한 관계들 사이에 발생하는 트랜잭션을 동시에 처리하기 위한 아주 간단한 기법이다. 단지, 현재는 자유도가 높아서 프로그램의 역량에 데이터베이스의 성능이 좌우된다는 점은 장점일 수 있지만, 최대의 단점이 될 수도 있다.

앞으로 오픈소스인 카산드라를 이용하여 제안 기법에 최적화할 수 있는 데이터 저장 기법이나 질의 API를 개선 혹은 추가한다면 새로운 DBMS가 될 수 있을 것이라 생각된다. 향후 연구를 통해 이를 연구해 보고자 한다.

## 참 고 문 헌


[1] Ching-Yung Lin, Nan Cao, Shixia Liu, Spiros Papadimitriou, Jimeng Sun, and Xifeng Yan. Smallblue: Social network analysis for expertise search and collective intelligence. In ICDE, pp.1483.1486, 2009.

[2] Padhy, R. P., Patra, M. R., Satapathy, S. C., RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's, International Journal of Advanced Engineering Sciences and Technology, 11(1):15 - 30, 2011.

[3] R. Angles, C. Gutierrez. Survey of graph database models. ACM Comput. Surv., 40(1), pp.1 - 39. 2008.

[4] G¨UTING, R. H. GraphDB: modeling and querying graphs in databases. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB). Morgan Kaufmann, pp.297-308, 1994.

[5] U Kang, C.E Tsourakakis, Ana Paula Appel, C Faloutsos,and Jure Leskovec. Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations. SIAM International Conference on Data Mining, 2010.

[6] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. GBASE: a scalable and general graph management system. In ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), pp.1091-1099, 2011.

[7] Sang-Kyu Noh, Jin-Su Park. 『Ontology』. god's Toy business. 2007(in Korean).

[8] National Information Society Agency. Standard Development of Korea Knowledge Portal Ontology. National Information Society Agency. 2006(in Korean).

[9] Pepper, S. and Moore, G (eds.): XML Topic Maps (XTM) 1.0. TopicMaps.Org, http://www.topicmaps.org/xtm/1.0/, 2001.

[10] Yeo-Sam Park, Ok-Bae Chang, Sung-Kook Han. X-TOP:Design and Implementation of TopicMaps Platform for Ontology Construction on Legacy Systems. Journal of KIISE : Computing Practices and Letters, Vol.14, No.2, pp.130-142, Apr., 2008(in Korean).

[11] Kristóf Kovács. Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase vs Membase vs Neo4j comparison. http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis. 2010.

[12] Hyung-Jun Kim, Jun-Ho Joe, Sung-Hwa Ahn, Byung-Jun Kim. 『Cloud Computing』. Acorn. 2010(in Korean).

[13] Daniel J. Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In SIGMOD Conference, pp.967.980, 2008.

[14] Dominic Williams. HBase vs Cassandra: why we moved. http://ria101.wordpress.com/2010/02/24/hbase-vs-cassandra-why-we-moved/

[15] Hewitt, Eben. 『Cassandra: the definitive guide』. O'Reilly Media. 2010.

[16] http://wiki.apache.org/cassandra/API/

[17] http://www.tpc.org/tpch/spec/tpch2.14.4.pdf.

[18] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing, pp.143{154, New York, NY, USA, 2010. ACM.



**신 재 현**

e-mail : jhyshin@skku.edu

2005년 부산대학교 물리학과(이학사)

2005년~2007년 LG CNS

2011년~현 재 성균관대학교 임베디드 소프트웨어학과 석사과정

관심분야 : 그래프 데이터베이스, 온톨로지, NoSQL, 클라우드 컴퓨팅