https://doi.org/10.5392/JKCA.2018.18.03.070

서브 그래프의 사용 패턴을 고려한 다중 계층 캐싱 기법

Multi-layer Caching Scheme Considering Sub-graph Usage Patterns

유승훈, 정재윤, 최도진, 박재열, 임종태, 복경수, 유재수
충북대학교 정보통신공학과

Seunghun Yoo(yshun2487@chungbuk.ac.kr), Jaeyun Jeong(jjjeong@chungbuk.ac.kr),
Dojin Choi(mycdj91@chungbuk.ac.kr), Jaeyeol Park(yeols@chungbuk.ac.kr),
Jongtae Lim(jtlim@chungbuk.ac.kr), Kyoungsoo Bok(ksbok@chungbuk.ac.kr),
Jaesoo Yoo(yjs@chungbuk.ac.kr)

요약

최근 소셜 미디어와 모바일 장비들의 발달로 인해 그래프 데이터들이 다양한 분야에서 활용되고 있다. 이와 함께 대용량 그래프 데이터 처리 과정에서 입출력 비용을 감소시키기 위한 캐싱 기법에 대한 연구들이 진행되고 있다. 본 논문에서는 그래프 토폴로지의 특징인 그래프의 연결성과 과거의 서브 그래프 사용 이력을 고려하여 다중 계층 캐싱 기법을 제안한다. 제안하는 기법은 캐시를 Used Data Cache와 Prefetched Cache로 구분한다. Used Data Cache는 자주 사용되는 서브 그래프 패턴에 따라 가중치를 부여하여 데이터를 캐싱하고 Prefetched Cache는 사용되지는 않았지만 최근 사용된 데이터의 주변 데이터들이 사용될 가능성이 높은 데이터를 캐싱한다. 그래프 패턴을 추출하기 위해 과거의 이력 정보를 활용하여 패턴을 추출하였다. 자주 사용되는 서브 그래프들이 다시 사용될 것을 예측하여 가중치를 부여하여 캐싱한다. 최근 사용된 데이터의 주변 데이터들이 사용 될 것을 예측하여 캐싱한다. 각각의 캐시에 캐싱된 데이터들을 관리하고 메모리가 가득 찰 경우 사용될 가능성이 낮은 데이터와 새로운 데이터를 교체하는 전략을 제안한다. 성능 평가를 통해 제안하는 캐싱 기법이 기존의 캐시 관리 기법에 비해 우수함을 증명한다.

■ 중심어 : | 그래프 | 인-메모리 | 캐싱 | 데이터 교체 | 빈발 패턴 검출 |

Abstract

Due to the recent development of social media and mobile devices, graph data have been using in various fields. In addition, caching techniques for reducing I/O costs in the process of large capacity graph data have been studied. In this paper, we propose a multi-layer caching scheme considering the connectivity of the graph, which is the characteristics of the graph topology, and the history of the past subgraph usage. The proposed scheme divides a cache into Used Data Cache and Prefetched Cache. The Used Data Cache maintains data by weights according to the frequently used sub-graph patterns. The Prefetched Cache maintains the neighbor data of the recently used data that are not used. In order to extract the graph patterns, their past history information is used. Since the frequently used sub-graphs have high probabilities to be reused, they are cached. It uses a strategy to replace new data with less likely data to be used if the memory is full. Through the performance evaluation, we prove that the proposed caching scheme is superior to the existing cache management scheme.

■ keyword : | Graph | In-memory | Caching | Data replacement | Frequent Pattern Detection |

* 본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업(IITP-2017-2013-0-00881), 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단(No. 2016R1A2B3007527), 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보 · 컴퓨팅기술개발사업의 지원을 받아 수행된 연구입니다.(No. NRF-2017M3C4A7069432)

접수일자 : 2017년 12월 04일
수정일자 : 2018년 01월 11일
심사완료일 : 2018년 01월 11일
교신저자 : 유재수, e-mail : yjs@chungbuk.ac.kr

I. 서 론

소셜 네트워크, 시맨틱 웹, 바이오 정보 시스템 등 다양한 분야에서 객체들 사이에 관계 및 상호 작용을 표현하기 위해 그래프 데이터를 활용하고 있다[1-3]. 그래프 데이터는 정점과 간선들로 구성된다. 그래프 데이터는 각 정점들의 정보뿐만 아니라 객체들 사이의 관계를 통해 데이터 분석, 예측, 추천 등에 사용한다. 소셜 네트워크에서는 인적 네트워크 분석을 통해 사용자 영향력을 판별하며 바이오 정보 시스템에서는 질병진단 및 신약개발 등에 활용하고 있다. 온라인 쇼핑과 같은 전자 거래, 영화와 음악과 같은 콘텐츠 서비스에서는 소비자들에게 아이템 추천을 위해 그래프 데이터들이 사용되고 있다[4][5].

대용량의 그래프 데이터를 처리하고 분석하기 위해 분산 처리[6], 서브 그래프 패턴 검출[7], 데이터의 재사용을 고려한 점진적 그래프 처리[8] 등에 대한 연구들이 진행되고 있다. 그 중에서도 시스템 성능을 효과적으로 향상시키기 위해 디스크에 있는 데이터를 메모리에 상주시키는 인-메모리 캐싱 기법에 대한 많은 연구가 진행되고 있다[9-11]. 인-메모리 캐싱은 디스크에 있는 데이터를 미리 메모리에 상주시키는 것으로 향후 접근되거나 사용될 데이터를 미리 메모리에 상주시켜 디스크에 접근 비용을 감소시키는 기법이다. 사용될 데이터가 메모리에 있는 경우 히트가 되었다고 하며 사용할 데이터는 디스크를 거치지 않고 메모리에 있는 데이터를 사용한다. 그래프 데이터에 대한 인-메모리 캐싱을 위해서는 그래프 데이터의 특성을 고려해야 한다. 그래프의 가장 큰 특성 중 하나는 데이터의 연결성이다. 연결성을 고려하여 앞으로의 접근할 데이터가 공간적으로 가장 가까운 것을 접근할 가능성이 높은 데이터를 예측한다. 한 정점에 대한 접근이 이루어 졌을 때 다음번에 간선으로 연결된 주변 이웃 정점이 접근될 가능성이 높다.

최근 그래프 데이터에 대한 인-메모리 캐싱에 관한 연구들이 많이 수행되었다. [10][11]에서는 임의 노드에 접근한 경우 인접 노드들을 모두 캐시에 저장하는 방법을 제안하였다. [10]에서는 그래프 교체 전략으로 일반적인 LRU[12](least recently used)를 사용하여 그래프 데이터를 데이터의 토폴로지를 고려하지 않고 일반 객체들을 처리하는 방법과 동일하게 사용하였다. [11]에서는 그래프 데이터의 특성에 따른 교체 정책을 제안하였다. 연결된 인접 정점들을 캐싱 할 때 가중치를 부여한다. 메모리에 데이터가 가득 찼을 때, 새로운 데이터와 가중치 값이 낮은 데이터와 교체하는 알고리즘을 제안하였다. [10][11]에서 그래프 데이터의 캐싱 할 때, 연결된 모든 정점들을 모두 메모리에 상주시켜 사용될 가능성이 낮은 데이터들이 메모리 용량을 차지하는 경우가 발생한다. 따라서 이웃 정점이 많은 정점에 접근 할 경우 주변 이웃들을 모두 메모리에 올리기 때문에 메모리 과부하가 발생할 가능성이 커진다. 또한, 한꺼번에 많은 데이터들을 교체하기 때문에 접근될 가능성이 낮은 데이터들까지 메모리에 캐싱한다. 기존 연구[10][11]들은 그래프 데이터를 처리하는데 있어 서브 그래프가 사용될 때 같이 사용된 이웃 정점들을 고려하지 않았다. 그래프 데이터의 메모리에 데이터를 캐싱하는데 있어 주변의 정점을 모두 캐싱한다. 예를 들어, 페이스북의 유명한 사람은 수십 수백만의 이웃을 가지고 있을 때, 그 주변의 이웃 데이터를 모두 메모리에 올리게 된다. 주변 이웃이 많은 정점에 접근이 자주 나타나게 되면 메모리가 과부하가 발생하거나 메모리에서 많은 I/O가 발생할 수 있다.

본 논문에서는 서브 그래프의 사용 패턴을 고려하여 접근 가능성이 높은 서브 그래프를 캐싱하는 다중 계층 캐싱 전략을 제안한다. 이를 통해 제안하는 기법은 메모리의 잦은 교체와 사용될 가능성이 낮은 데이터들을 캐싱하는 것을 방지할 수 있다. 과거에 사용된 데이터와 앞으로 사용될 데이터를 나누어 관리하기 위해 캐시를 Used Data Cache와 Prefetched Cach로 나누어 관리한다. 메모리내의 데이터가 가득 찼을 경우 메모리 내의 데이터를 새로운 데이터와 교체하는 전략이 필요하다. 두 개의 캐시 계층을 각기 다른 방식으로 데이터를 관리한다. Used Data Cache는 TTL(Time To Live)값으로 데이터를 관리하며 TTL 값이 낮은 데이터를 교체 대상으로 삼는다. Prefetcehd Cache는 큐(queue)형식으로 데이터를 관리되어 먼저 들어온 데이터가 가장

먼저 교체 대상이 된다. Prefetched Cache에서 캐시 적중이 발생하면 Used Data Cache로 데이터를 이주하여 관리한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 기존의 그래프 데이터 캐시 관리 기법과 기존 연구의 문제점에 대해서 기술한다. Ⅲ장에서는 그래프 패턴과 그래프 사용 이력 정보를 이용한 캐시 방법 및 관리 기법을 소개한다. Ⅳ장에서는 제안하는 기법의 우수성을 파악하기 위해 실험을 통하여 검증한다. 마지막으로 Ⅴ장에서는 본 논문의 결론과 향후 진행 연구에 대해 기술한다.

Ⅱ. 관련 연구

그래프 데이터의 활용과 관련 어플리케이션들이 많아짐에 따라 많은 기법들이 연구 되었다. Facebook[13]에서는 캐시 계층을 효과적으로 사용하여 시스템 성능을 개선하기 위해 분산 소셜 그래프를 통한 캐시 계층에서 Memcache[9]를 사용하였다. Memcache는 분산 메모리 캐시 환경에서 최근에 사용하지 않은 데이터를 퇴출시키는 교체정책을 사용하여 대형 그래프 처리를 위한 시스템을 제공한다. 하지만 그래프 데이터 처리를 위한 캐싱 계층을 따로 제공하지 않고 LRU를 이용하여 메모리 캐시 교체 정책을 사용하였다. Neo4j[14]는 여러 머신에 걸쳐 데이터를 공유 할 수 있도록 한 오픈 소스 그래프 데이터베이스이다. Neo4j에서는 두 개의 캐시 계층을 제공하고 실제 데이터를 저장하는 FBC(file buffer cache)와 정점, 간선 그리고 메타데이터를 저장하는 OC(object cache)로 나누어 처리한다. 하지만 그래프 데이터의 특징을 고려하지 않고 LRU 교체 방식을 사용하였다. Pregel[15]은 대형 그래프 처리를 위한 시스템을 제공하였다. 하지만 캐싱 계층을 제공하지 않았다.

[10]에서는 소셜 네트워크에서 생성된 대규모 그래프를 분석하기 위한 NYNN이라는 인-메모리 시스템을 제안하였다. 소셜 네트워크 데이터의 특징으로 대규모 데이터와 빈번한 접근과 업데이트를 고려하였다. 캐시 관리 정책으로 향후 접근할 데이터가 공간적으로 가장 가까운 것을 접근할 가능성이 높기 때문에 접근된 노드의 주변 이웃을 같이 메모리로 캐싱하는 방법을 제안하였다. 하지만 캐시 퇴출 및 교체 정책은 범용적으로 사용되는 LRU방식을 사용하였다. [11]에서는 빅데이터 플랫폼에서 분산 그래프 처리를 위한 그래프 데이터 캐싱 정책을 제안하였다. 그래프 데이터의 앞으로 접근할 데이터가 현재 접근하고 있는 정점의 주변 정점임을 예상하여 주변 정점 및 여러 홉의 정점을 메모리에 캐싱하는 캐시 정책을 제안하였다. 또한, 그래프 데이터의 토폴로지를 분석하고 고려하여 메타 데이터의 크기와 주변 이웃 간의 홉 수 그리고 분산처리에 따른 통신 시간을 고려하여 캐시 교체 및 퇴출 정책인 CBGA (clock-based graph aware caching)를 제안하였다.

그래프 사용 패턴을 추출하기 위해서 [16]에서는 여러 트랜잭션에서 동시에 사용된 항목들의 사용 패턴을 추출해 내는 알고리즘인 FP(frequent pattern)-Growth 알고리즘을 사용하였다. FP-Growth 알고리즘은 패턴을 찾기 위해 트리와 노드 링크라는 특별한 자료구조를 사용한다. 트랜잭션들을 관리하는 테이블을 구축하고 구축된 테이블을 통해 FP-트리를 생성하고 패턴을 추출한다. 본 논문에서는 이 방법을 그래프 데이터에 적용시켜 같이 사용된 정점들을 찾아내고 패턴을 추출해 낸다.

[13-15]에서는 그래프 데이터를 처리하기 위한 시스템을 처리하기 위한 시스템이나 저장소를 제공하였다. 하지만 그래프 토폴로지의 특성을 고려하지 않거나 그래프 데이터 처리를 위한 캐시 계층을 제공하지 않았다. 그래프 데이터를 일반 객체로 인식하여 데이터를 처리하였다. 반면, [10][11]에서는 그래프 토폴로지의 특성을 고려하여 캐싱한다. 하지만 인접한 이웃 데이터가 많은 경우 메모리 내 데이터의 잦은 교체와 메모리 과부하가 발생 할 수 있다. 이러한 문제를 해결하기 위해 본 논문에서는 FP-Growth 알고리즘을 사용하여 그래프 사용 패턴을 추출하여 캐싱 시스템에 적용한 캐싱 알고리즘을 제안한다.

Ⅲ. 제안하는 다중 계층 캐싱 기법

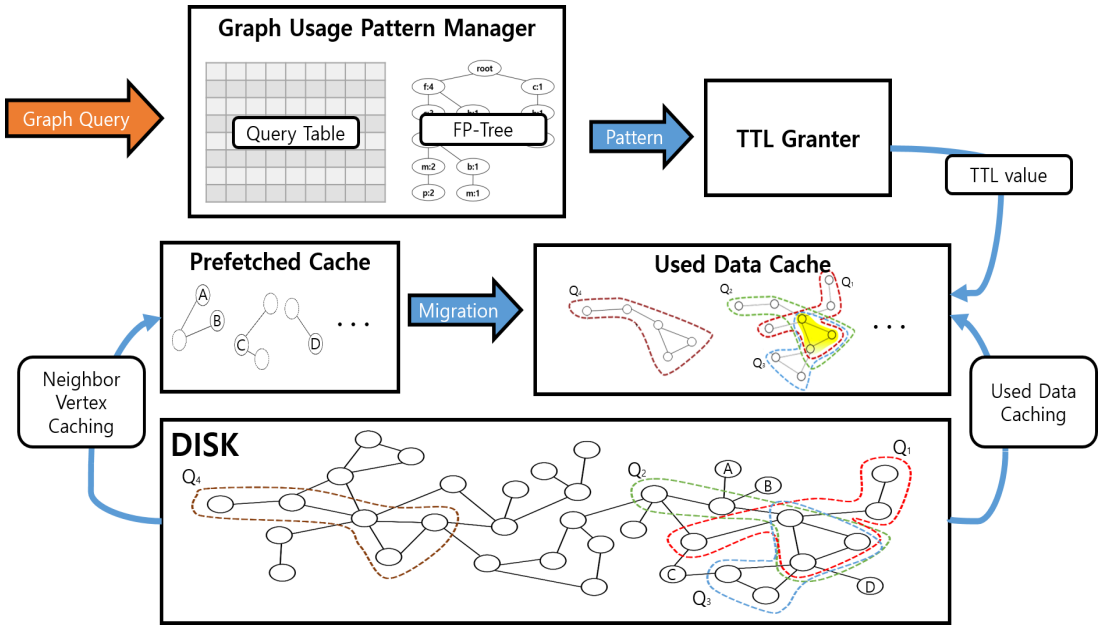

그림 1. 전체 캐싱 시스템 구조

1. 제안하는 기법의 구조

빠른 그래프 데이터의 처리를 위해 그래프 데이터의 특징과 데이터 구조를 고려해야 한다. 그래프의 연결성과 과거의 서브 그래프 사용 이력을 고려하여 서브 그래프의 사용 패턴과 접근 빈도를 고려하여 다중 계층 캐싱 기법을 제안하였다. 제안하는 인-메모리 캐싱 시스템 구조는 캐시 매니저와 다중 계층 캐시로 나뉜다. 캐시 매니저는 질의 관리 테이블에서 사용된 질의 이력을 저장하고 사용된 질의의 빈도수를 통해 FP-트리를 구축하여 질의 패턴을 검출한다. 또한, 캐싱되는 데이터들 관리를 위한 TTL값을 부여한다. 다중 계층 캐시는 Used Data Cache와 Prefetched Cache로 두 개의 계층으로 나뉜다. Used Data Cache는 이전에 사용되었던 데이터들 중 다시 사용될 가능성이 높은 데이터들을 캐싱한다. Prefetched Cache는 사용되지는 않았지만 다음번 질의에 사용될 가능성이 높은 데이터들을 캐싱한다.

제안하는 기법의 전체 구조는 [그림 1]과 같다. 질의 Q_1, Q_2, Q_3, Q_4가 요청이 되었을 때, 각각의 질의에 해당되는 데이터들은 Used Data Cache에 캐싱된다. 이때 앞서 발생한 질의와 현재 질의에 따라 서브 그래프 패턴을 추출하고 각 데이터의 패턴값에 따라 TTL 값을 부여한다. Prefetched Cache에는 요청된 그래프의 주변 정점 A, B, C, D등의 정점을 캐싱한다. Prefetched Cache에서 히트가 발생하였을 때 Prefetched Cache에 있는 데이터를 Used Data Cache로 이주한다.

그래프 질의 요청이 들어오게 되면 Used data Cache와 Prefetched Cache에 요청된 데이터가 있는지 검사한다. 데이터가 있을 경우 메모리 내에 있는 데이터를 사용하고 데이터가 없을 경우 디스크에서 데이터를 가져온다. 질의에 응답하는 동시에 요청된 질의를 질의 이력 관리 테이블에 질의 정보를 저장한다. 질의 이력 관리 테이블을 통해 FP-트리를 생성하고 질의패턴을 통해 TTL값을 계산하여 Used data Cache에 저장되는 데이터에 TTL값을 부여한다. 질의 처리 절차는 [그림 2]과 같다. 그래프 질의 요청이 들어오게 되면 가장먼저 Used Data Cache를 확인하고 다음으로 Prefetched Cache를 확인한다. 메모리에 데이터가 없는 경우 디스크에서 탐색이 이루어지게 된다.

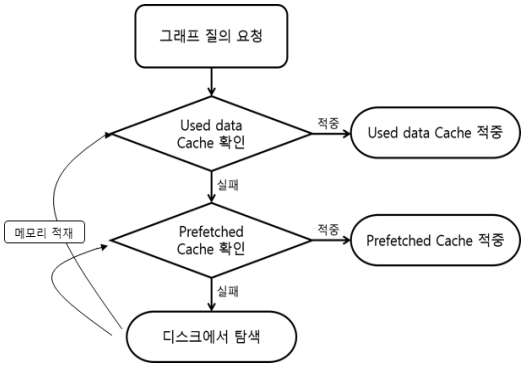

그림 2. 질의 처리 절차

2. 그래프 사용 패턴 관리

다수 사용되는 데이터는 또 다시 사용될 가능성이 높다. 따라서 사용될 가능성이 높은 서브 그래프 패턴의 데이터를 메모리에 적재시키기 위해 패턴을 추출해야 한다. 본 논문에서는 추출된 패턴에 따라 각 서브그래프 에 가중치를 부여한다. 같이 사용된 이력이 많은 정점들은 높은 가중치를 부여 받아 메모리에 오래 상주하게 된다.

Q_ID	Vertex	Frequent Vertex
1	f,a,c,d,g,l,m,p	f,c,a,m,p
2	a,b,c,f,l,m,o	F,c,a,b,m
3	b,f,h,j,o	f,b
.	.	.
.	.	.
.	.	.

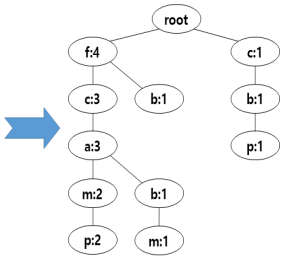

그림 3. 질의 이력 관리 테이블과 FP-트리

서브 그래프의 질의가 들어올 때마다 해당되는 질의 정보를 질의 이력 관리 테이블에 저장한다. 저장되는 질의 정보는 간선의 사용 이력들을 저장한다. 질의 관리 테이블은 큐(queue)형식으로 관리하며 테이블에 질의가 가득 찬 경우 질의가 들어온 순서대로 삭제되게 된다. 질의 이력 관리 테이블에 따라 FP-트리가 생성되게 된다. 테이블을 탐색하면서 빈발된 간선 리스트를 찾아내고 내림차순으로 정렬 한다. 정렬된 빈발 간선 리스트의 순서에 따라 두 번째 검색을 통하여 질의를 검색하면서 재귀적 호출을 통해 FP-트리를 생성하게 된다. 질의 이력 관리 테이블의 질의 이력을 통해 FP-트리를 구축을 한다. 구축된 FP-트리에서 사용이력이 높은 간선들을 찾아내어 패턴 가중치(P)를 추출 해낸다. [그림 3]은 질의 이력 관리 테이블과 FP-트리를 나타낸다. 질의 이력 관리 테이블에는 각 질의가 들어온 순서대로 질의 정보 {Q_1, Q_2, Q_3, ⋯ ,Q_n}가 저장되어 있다. 질의 이력 관리 테이블의 질의 정보를 통해 FP-트리를 구축하여 패턴 P(f, c, a)에 대한 패턴 가중치 값을 추출 해낸다. 이때 P(f, c, a) 값은 3이 되고 P(f, c, a, m, p)의 값은 2가 된다.

3. 그래프 데이터 캐싱

본 논문에서는 캐싱 계층을 Used Data Cache와 Prefetched Cache로 나눈 다중 계층 캐시 구조이다. 다중 계층 캐시 구조는 계층을 두 개로 나눈 구조로 자주 사용된 데이터와 앞으로 사용될 가능성이 높은 데이터를 각기 다르게 관리한다. 각 계층에서 메모리에 적재하는 기준이 다르기 때문에 기준에 따라 각각 다른 캐시에 저장 하여 쉽게 관리할 수 있다.

3.1 Used Data Cache

그래프 질의가 요청 되었을 때 이전에 사용되었던 데이터들이 다시 사용될 가능성이 높다. 따라서 Used Data Cache는 사용 되었던 데이터를 캐싱하고 자주 사용 되었던 서브 그래프들을 메모리에 오래 상주시킨다. 질의 관리 테이블과 FP-트리를 통해 찾아낸 패턴 가중치(Pattern weight)와 패턴에 포함된 정점 수(Number of vertex)를 통해 TTL값을 부여한다. 패턴 가중치 값은 FP 트리를 통해 추출된 사용된 간선 빈도를 의미하고 패턴에 포함된 정점 수는 같이 사용된 서브 그래프의 크기를 의미하게 된다. TTL 값은 식 (1)과 같다.

$$TTL = \alpha N \times (1-\alpha) P \quad (1)$$

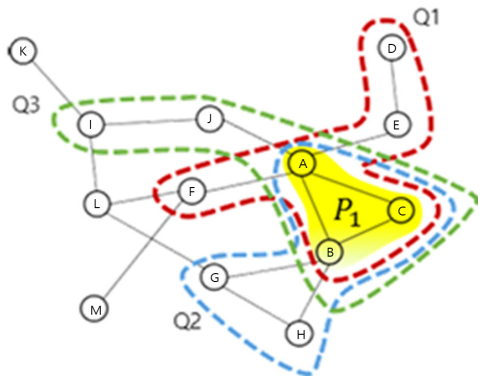

그림 4. 그래프 질의 및 패턴

[그림 4]와 같이 질의 Q1, Q2, Q3가 존재한다고 할 때 각 질의의 정점들을 Used Data Cache에 캐싱한다. 또한, 캐시 매니저의 질의 관리테이블과 FP-트리를 통해 패턴 P1을 검출하고 P1의 정점들은 다른 정점들보다 더 높은 패턴 가중치로 인해 높은 TTL값을 부여 받는다. [표 1]은 질의Q1, Q2, Q3가 요청 되었을 때, TTL값을 나타낸 것이다. 접근 이력이 높은 P1에 해당하는 정점 A, 정점 B, 정점C의 TTL값이 높은 것을 확인 할 수 있다. α값은 0.3으로 하였다.

표 1. 패턴 추출에 따른 TTL 값

$Vertex_{no}$	TTL 값	$Vertex_{no}$	TTL 값
A	3	H	2.2
B	3	I	2.2
C	3	J	2.2
D	2.5	K	0
E	2.5	L	0
F	2.2	M	0
G	2.2		

3.2 Prefetched Cache

그래프 데이터 특성상 전에 사용되었던 데이터들의 주변 이웃 정점 데이터들의 사용 가능성이 높다. 따라서 Prefetched Cache에서 요청된 그래프데이터의 주변 정점들을 캐싱한다. 그래프 데이터의 각 간선에는 같이 사용되었던 데이터의 접근 횟수를 저장한다. 예를 들어, A정점과 B정점이 같이 사용된 경우, A정점과 B정점을 잇는 간선의 접근 횟수를 1 증가 시킨다. 질의가 요청되었을 때 사용된 데이터의 접근 빈도가 높은 주변 정점들을 메모리에 같이 캐싱한다. 그래프 데이터의 간선에 저장된 접근수를 이용하여 접근이력이 높은 Top-k 개의 정점을 선별하여 메모리에 캐싱한다. 또한, 한 홉의 이웃 정점 다중 홉의 이웃 정점까지 메모리에 캐싱한다. 다중 홉 이웃 정점 캐싱의 경우, 이전 홉에서 선별된 정점의 접근이력이 높은 정점을 선별한다. [그림 5]처럼 질의가 요청 되면 주변 정점들 중 접근이력이 높은 정점들을 찾는다. k가 2인 경우, 정점 G과 정점M이 선별되어 메모리에 캐싱된다. 반면에 나머지 주변 정점인 A, D, H는 캐싱되지 않는다. 다중 홉을 고려하였을 경우, 정점 G의 접근 이력이 높은 정점 L과 정점 M의 접근 이력이 높은 정점 O가 선별되어 메모리에 같이 캐싱된다.

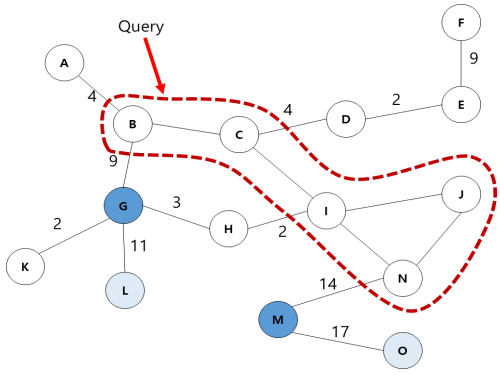

그림 5. 그래프 질의에 따른 주변 정점 캐싱

4. 캐시 교체 및 관리 정책

메모리에 데이터가 가득 차게 되면 더 이상 데이터를 캐싱 할 수 없게 된다. 따라서 메모리 있는 데이터를 축출시키거나 새로운 데이터와 교체하는 전략이 필요하다. 본 논문에서는 그래프 특성을 고려한 교체 및 관리 정책을 제안한다.

Used Data Cache는 사용 되었던 데이터를 캐싱하게 된다. 질의 요청이 들어왔을 때, 캐시 미스가 발생하거나 Prefetched Cache에서 히트가 발생한 경우에 데이터를 Used Data Cache에 캐싱한다. 이때, 메모리가 가득 차 있을 경우 캐시 매니저에 의해 부여된 TTL 값을 데이터 교체 전략을 수행하고 캐시 내 데이터를 관리한다.

TTL 값은 질의가 수행 될 때 데이터의 수만큼 감소되며, 새로운 데이터가 메모리로 캐싱될 때 새로 캐싱된 데이터는 TTL값이 가장 낮은 데이터와 비교하여 새로운 데이터의 TTL값이 큰 경우 기존 데이터와 새로운 데이터가 교체 이루어지고, TTL값이 낮은 경우 교체가 이루어지지 않는다. 히트가 발생하였을 경우 데이터의 기존의 TTL값을 새로운 TTL값으로 교체시킨다.

Prefetched Cache는 사용된 데이터의 이웃 정점들을 캐싱하게 된다. FIFO(first in first out)형식으로 캐시를 관리한다. Prefetched Cache에서 히트가 발생하면 Used Data Cache로 데이터를 보낸다. 동시에, 기존의 Prefetched Cache내에 있던 데이터를 삭제한다. 만약 Used Data Cache로의 캐싱이 실패하게 되면 Prefetched Cache에 재 삽입한다. [그림 6]은 Prefetched Cache의 교체 전략을 보여준다. 정점 V_3이 히트가 되었을 때, Prefetched Cache내의 정점 V_3은 Used Cache로 이주가 되어 Prefetched Cache에서 삭재가 된다. 또한, 새로운 데이터 정점 V_{n+1}과 정점 V_{n+2}가 들어옴으로써 V_1이 교체된다.

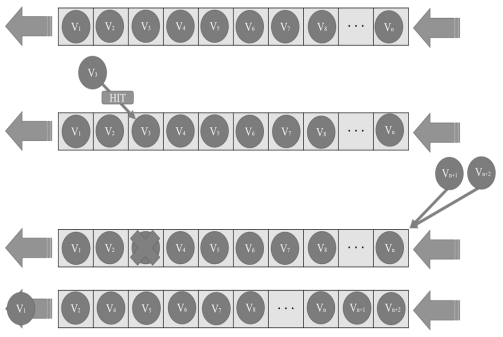

그림 6. Prefetched Cache 전략

Ⅳ. 성능 평가

본 절에서는 제안하는 기법과 기존 기법의 성능 비교 평가를 통해 제안하는 기법의 우수성을 입증한다. 교체 전략을 LRU[12]방식을 사용한 NYNN[10]과 그래프 환경을 고려한 교체전략을 사용한 CBGA[11]와의 비교 평가를 수행하였다. 성능 평가 환경은 [표 2]과 같다. Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz, RAM 8.00GB, Microsoft Window 10 64비트 운영체제를 사용하는 PC에서 자바 프로그램을 통해 시뮬레이션을 수행하였다. 실험 데이터로는 [표 3]과 같이 Twitter[17], Weki-talk[18]와 Youtube[19] 세 가지 데이터 집합을 사용하였다. 각각의 데이터 집합은 사용자들의 데이터와 연결 정보인 간선으로 이루어져 있다. 요청하는 데이터 수는 1,000~10,000개이다. 요청된 데이터수와 히트가 발생한 수에 따른 히트율과 질의 요청에 따른 수행시간에 대해 성능 평가를 진행 하였다. 히트율에 대한 식은 식 (2)과 같다.

$$Hit\ Ratio = \frac{Nmber\ of\ hits}{Nmber\ of\ requested} \quad (2)$$

표 2. 성능 평가 환경

구분	내용
프로세서	Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
메모리	8 GB
디스크	1 TB
프로그램 언어	Java

표 3. 데이터 집합

구분	정점 수	간선 수
Twitter	11,316,811	85,331,846
Weki-Talk	2,394,385	5,021,410
Youtube	1,134,890	2,987,624

Used Data Cache에서 부여받는 TTL값을 결정하기 위해 임계치 값 α를 결정해야한다. Twitter 데이터를 이용하여 식 (1)에서 패턴 가중치 값과 패턴에 포함된 정점 수에 곱해지는 α값을 각각 다르게 하여 히트율이 높은 결과를 내는 α값을 찾았다. 식 (1)의 α값을 0.1부터 0.9 까지 0.05단위로 바꿔가며 성능평가를 진행하였다. [그림 7]은 변경된 α 값에 따른 히트율에 대한 평가 결과를 보여준다. 그림에서 보는 것과 같이 α값이 0.30일 때 가장 높은 히트율을 나타내었다. 따라서 히트율이 가장 좋은 α = 0.30 을 성능 평가에 사용한다.

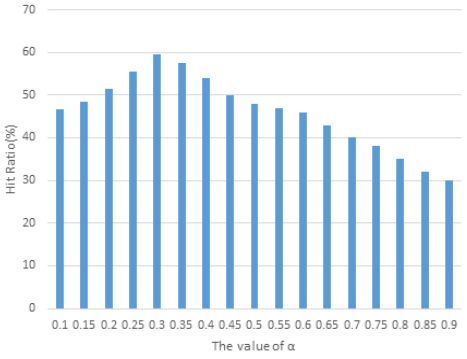

그림 7. α 값에 따른 히트율

[그림 8]은 Twitter, Weki-talk와 Youtube 데이터를 가지고 히트율에 대한 평가 결과를 보여준다. 기존 기법들인 NYNN과 CBGA를 비교하여 제안하는 기법이 가장 높은 히트율을 나타내었다. NYNN은 메모리 내의 데이터를 교체하는데 있어 그래프에 대한 특성을 고려하지 않았기 때문에 전체 데이터 집합에 대한 평가에서 가장 낮은 히트율이 나타났다. CBGA는 그래프의 특성을 고려하였지만 메모리내의 핫 데이터에 대해 가중치를 주지 않았다. 따라서 잦은 데이터 교체로 인해 제안하는 기법보다 낮은 히트율을 나타내었다. 성능평가 결과, 제안하는 기법의 히트율은 NYNN에 비해 평균 약 223% 증가, CBGA에 비해 약 129% 증가하였다.

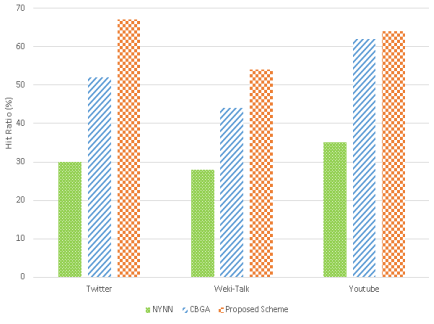

그림 8. 각각의 데이터 집합에 대한 히트율

[그림 9]은 이전 기법들과 제안하는 기법에 따른 수행 속도에 대한 평가 결과를 보여준다. 1,000개부터 10,000개까지의 데이터를 수행하였다. 제안하는 기법이 이전 기법인 NYNN과 CBGA에 비해 수행 속도 성능의 우수함을 보여준다. 기존 기법인 NYNN과 CBGA는 주변 이웃들을 모두 메모리에 캐싱하기 때문에 데이터의 입/출력이 많아진다. 하지만 제안하는 기법은 Top-k개의 데이터들을 캐싱하기 때문에 기존 기법보다 적은 수행시간이 걸렸다. 성능평가 결과, 제안하는 기법의 처리 시간은 LRU[12]에 비해 평균 약 186%, CBGA[11]에 비해 약 127% 향상되었다.

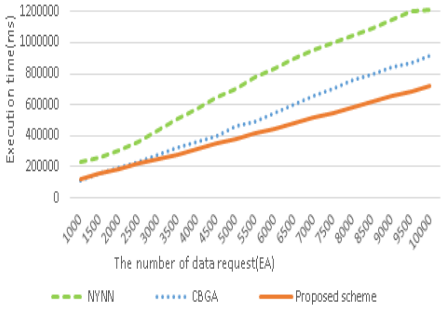

그림 9. 데이터 요청에 따른 수행 시간

V. 결 론

본 논문에서는 서브 그래프의 사용 패턴을 고려하여 접근 가능성이 높은 서브 그래프를 캐싱하는 다중 계층 캐싱 기법을 제안하였다. 제안하는 기법은 그래프 패턴을 캐시에 적용하기 위해 그래프 데이터의 과거 이력 정보를 활용하여 질의 이력 관리 테이블과 FP-트리를 통해 패턴을 찾아내었다. 자주 사용되는 서브 그래프들이 또다시 사용될 가능성이 높기 때문에 더 높은 가중치를 부여하여 Used Data Cache에 캐싱한다. 또한, 최근 사용된 데이터의 주변 데이터들이 사용 될 것을 예측하여 주변 데이터들을 Prefetched Cache에 캐싱한다. 각각의 캐시에 캐시된 데이터들을 관리하고 메모리가 가득 찰 경우 캐시된 데이터와 새로운 데이터를 교체를 수행 하였다. 성능 평가에서는 기존 기법과 비교를 통해 본 논문의 기법의 우수성을 증명하였다. 성능평가 결과 제안하는 기법이 기존 기법보다 히트율에서 우수한 성능을 나타내었다. 하지만 질의 요청 수가 적은 경

우 질의 수가 부족하여 질의 이력 관리 테이블의 정보가 충분하지 않아 처리 속도 측면에서 기존 기법과 성능 차이가 적었다. 본 연구를 통해 소셜 네트워크에서의 인적 네트워크 분석과 바이오 정보 시스템 등의 다양한 분야에서 그래프 데이터를 빠르게 처리 할 수 있다. 향후 연구로는 요청된 데이터가 적을 때에도 기존 연구보다 좋은 성능을 나타낼 수 있도록 보안할 계획이다. 또한 분산 메모리 환경에서 그래프 데이터 캐싱 기법에 대한 연구를 진행할 예정이다.

참 고 문 헌

[1] A Cuzzocrea, F Furfaro, G. M. Mazzeo, and D. Saccà, "A grid framework for approximate aggregate query answering on summarized sensor network readings," Proc. OTM Workshops, pp.144-153, 2004.

[2] A. Fariha, C. F. Ahmed, C. K. Leung, S. M. Abdullah, and L. Cao, "Mining frequent patterns from human interactions in meetings using directed acyclic graphs," Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp.38-49, 2013.

[3] 임종태, 복경수, 유재수, "대용량 그래프 환경에서 스카이라인을 이용한 서브 그래프 유사도 측정 기법," 한국콘텐츠학회 종합학술대회, pp.47-48, 2017.

[4] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," IEEE Internet Computing, Vol.7, No.1, pp.76-80, 2003.

[5] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan, "Large-Scale Parallel Collaborative Filtering for the Netflix Prize," Proc. International Conference on Algorithmic Aspects in Information and Management, pp.337-348, 2008.

[6] J. E. Gonzalez, Y. Low, and H. Gu, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," Proc. USENIX Symposium on Operating Systems Design and Implementation, pp.17-30, 2012.

[7] 서복일, 김재인, 황부현, "스트림 데이터 환경에서 배치 가중치를 이용하여 사용자 특성을 반영한 빈발항목 집합 탐사," 한국콘텐츠학회논문지, 제11권, 제1호, pp.56-64, 2011.

[8] U. Gupta and L. Fegaras, "Distributed Incremental Graph Analysis," Proc. IEEE International Congress on BigData, pp.75-82, 2016.

[9] https://memcached.org/

[10] P Ran, W Zhou, and J Han, "NYNN: An In-Memory Distributed Storage System for massive graph analysis," Proc. International Conference on Advanced Computational Intelligence, pp.383-389, 2015.

[11] H. Aksu, M. Canim, Y. Chang, I. Korpeoglu, and Ö. Ulusoy, "Graph Aware Caching Policy for Distributed Graph Stores," Proc. International Conference on Cloud Engineering, pp.6-15, 2015.

[12] T. R. F'uzak, *Analysis of cache replacement algorithms,* Ph.D. dissertation, University of Massachusetts Amherst, 1985.

[13] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling Memcache at Facebook," Proc. USENIX Symposium on Networked Systems Design and Implementation, pp.385-398, 2013.

[14] https://neo4j.com/docs/

[15] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph

processing," Proc. ACM SIGMOD International Conference on Management of data, pp.135-146, 2010.

[16] P. Braun, J. J. Cameron, A. Cuzzocrea, F. Jiang, and C. K. Leung, "Effectively and Efficiently Mining Frequent Patterns from Dense Graph Streams on Disk," Proc. International Conference in Knowledge Based and Intelligent Information and Engineering Systems, pp.338-347, 2014.

[17] http://socialcomputing.asu.edu

[18] https://snap.stanford.edu/

[19] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," Proc. ACM SIGCOMM Internet Measurement Conference, pp.29-42, 2007.

[20] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. ACM SIGMOD International Conference on Management of Data, pp.1-12, 2000.

[21] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, and H. C. Li, "Tao: Facebook's distributed data store for the social graph," Proc. USENIX Annual Technical Conference, pp.49-60, 2013.

[22] Han, Jiawei, Jian Pei, and Yiwen Yin, "Mining frequent patterns without candidate generation," Proc. ACM SIGMOD International Conference on Management of Data, pp.1-12, 2000.

[23] C. Borgelt, "An Implementation of the FP-growth Algorithm," Proc. International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations, pp.1-5, 2005.

저 자 소 개

유 승 훈(Seunghun Yoo) 준회원

- 2016년 2월 : 충북대학교 정보통신공학과(공학사)
- 2016년 3월 ~ 현재 : 충북대학교 정보통신공학과(석사과정)

<관심분야> : 인-메모리 시스템, 데이터베이스 시스템, 분산 컴퓨팅, 인-메모리 시스템, 빅데이터 등

정 재 윤(Jaeyun Jeong) 준회원

- 2016년 2월 : 충북대학교 정보통신공학과(공학사)
- 2016년 3월 ~ 현재 : 충북대학교 정보통신공학과(석사과정)

<관심분야> : 그래프 분석, 빈발 패턴 마이닝, 데이터베이스 시스템, 분산 컴퓨팅 등

최 도 진(Dojin Choi) 정회원

- 2014년 2월 : 한국교통대학교 컴퓨터공학과(공학사)
- 2016년 2월 : 한국교통대학교 컴퓨터공학과(공학석사)
- 2016년 3월 ~ 현재 : 충북대학교 정보통신공학과 박사과정

<관심분야> : 연속 질의 처리, 위치 기반 서비스, 그래프 스트림 처리, 빅데이터 등

박 재 열(Jaeyeol Park) 준회원

- 2014년 2월 : 충북대학교 정보통신공학과(공학사)
- 2016년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2017년 3월 ~ 현재 : 충북대학교 정보통신공학과(박사과정)

<관심분야> : 데이터베이스 시스템, RDF, 실체화 뷰, 빅데이터 등

임 종 태(Jongtae Lim) 정회원

- 2009년 2월 : 충북대학교 정보통신공학과(공학사)
- 2011년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2015년 8월 : 충북대학교 정보통신공학과(공학박사)
- 2015년 9월 ~ 현재 : 충북대학교 정보통신공학과 박사후연구원 (Post.doc)

<관심분야> : 시공간 데이터베이스 시스템, 이동 객체 질의 처리, 위치기반 서비스, P2P 네트워크 등

복 경 수(Kyungsoo 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 초빙교수

<관심분야> : 데이터베이스 시스템, 이동객체 데이터베이스, 소셜 네트워크, 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

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