MapReduce 환경에서 재그룹핑을 이용한 Locality Sensitive Hashing 기반의 K-Nearest Neighbor 그래프 생성 알고리즘의 개선

An Improvement in K-NN Graph Construction using re-grouping with Locality Sensitive Hashing on MapReduce

ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 21, No. 11, pp. 681-688, 2015. 11
http://dx.doi.org/10.5626/KTCP.2015.21.11.681

MapReduce 환경에서 재그룹핑을 이용한 Locality Sensitive Hashing 기반의 K-Nearest Neighbor 그래프 생성 알고리즘의 개선

(An Improvement in K-NN Graph Construction using re-grouping with Locality Sensitive Hashing on MapReduce)

이 인 회 † 오 혜 성 † 김 형 주 ††
(Inhoe Lee) (Hyesung Oh) (Hyoung-Joo Kim)

요 약 k-Nearest Neighbor(k-NN)그래프는 모든 노드에 대한 k-NN 정보를 나타내는 데이터 구조로써, 협업 필터링, 유사도 탐색과 여러 정보검색 및 추천 시스템에서 k-NN그래프를 활용하고 있다. 이러한 장점에도 불구하고 brute-force방법의 k-NN그래프 생성 방법은 $O(n^2)$의 시간복잡도를 갖기 때문에 빅데이터 셋에 대해서는 처리가 곤란하다. 따라서, 고차원, 희소 데이터에 효율적인 Locality Sensitive Hashing 기법을 (key, value)기반의 분산환경인 MapReduce환경에서 사용하여 k-NN그래프를 생성하는 알고리즘이 연구되고 있다. Locality Sensitive Hashing 기법을 사용하여 사용자를 이웃후보 그룹으로 만들고 후보내의 쌍에 대해서만 brute-force하게 유사도를 계산하는 two-stage 방법을 MapReduce환경에서 사용하였다. 특히, 그래프 생성과정 중 유사도 계산하는 부분이 가장 많은 시간이 소요되므로 후보 그룹을 어떻게 만드는 것인지가 중요하다. 기존의 방법은 사이즈가 큰 후보그룹을 방지하는데 한계점이 있다. 본 논문에서는 효율적인 k-NN 그래프 생성을 위하여 사이즈가 큰 후보그룹을 재구성하는 알고리즘을 제시하였다. 실험을 통해 본 논문에서 제안한 알고리즘이 그래프의 정확성, Scan Rate측면에서 좋은 성능을 보임을 확인하였다.

키워드: 빅데이터, 맵리듀스, k-NN 그래프 생성, LSH, MinHash

Abstract The k nearest neighbor (k-NN) graph construction is an important operation with many web-related applications, including collaborative filtering, similarity search, and many others in data mining and machine learning. Despite its many elegant properties, the brute force k-NN graph construction method has a computational complexity of $O(n^2)$, which is prohibitive for large scale data sets. Thus, (Key, Value)-based distributed framework, MapReduce, is gaining increasingly widespread use in Locality Sensitive Hashing which is efficient for high-dimension and sparse data. Based on the two-stage strategy, we engage the locality sensitive hashing technique to divide users into small subsets, and then calculate similarity between pairs in the small subsets using a brute force method on MapReduce. Specifically, generating a candidate group stage is important since brute-force calculation

† 비 회 원 : 서울대학교 전기 컴퓨터공학(Seoul National Univ.)
ihlee@idb.snu.ac.kr
hsoh@idb.snu.ac.kr
(Corresponding author임)
†† 종신회원 : 서울대학교 전기 컴퓨터공학 교수
hjk@snu.ac.kr
논문접수 : 2015년 2월 6일
(Received 6 February 2015)
논문수정 : 2015년 8월 24일
(Revised 24 August 2015)
심사완료 : 2015년 9월 14일
(Accepted 14 September 2015)

정보과학회 컴퓨팅의 실제 논문지 제21권 제11호(2015. 11)

is performed in the following step. However, existing methods do not prevent large candidate groups. In this paper, we proposed an efficient algorithm for approximate k-NN graph construction by re-grouping candidate groups. Experimental results show that our approach is more effective than existing methods in terms of graph accuracy and scan rate.

Keywords: Big Data, MapReduce, k-NN Graph Construction, Locality Sensitive Hashing, MinHash

1. 서 론

k-Nearest Neighbor(k-NN)그래프는 모든 노드(node)에 대한 k-NN 정보를 나타내는 데이터 구조이다. 그림 1은 2-NN그래프의 예를 보여주고 있는데, 이 그래프에서 각 노드의 진출간선(outgoing edge)은 그 노드의 2-NN을 나타내며, 각 진출간선에 대응하는 값은 2-NN에 대한 유사도를 나타낸다.

K-NN그래프는 웹과 관련된 많은 애플리케이션에서 중요한 연산이다. 사용자 기반 협업 필터링(user-based collaborative filtering)[1]에서 k-NN그래프는 비슷한 로그패턴을 가진 사용자를 연결하여 그래프를 생성하고, 그래프에서 사용자의 이웃(neighbor nodes)에 기반하여 추천한다. 이 뿐만 아니라, 내용기반 검색 시스템(contents-based search system)에서는 데이터 셋의 변화가 없을 때, k-NN그래프 생성하는 방법이 k-NN탐색을 하는 것보다 더 바람직한 방법이다[2]. 또한, k-NN그래프는 이상 값 탐지(outlier detection)[3], 이미지 분류(image classification)[4] 방법의 핵심 데이터구조이다. 이렇듯 여러 추천 시스템 및 정보검색에서 k-NN그래프를 활용하고 있다. K-NN그래프는 이와 같이 여러 방법에 굉장히 중요한 역할을 하고 있어, 최근 들어 활발히 연구되고 있다[5].

정확한 k-NN그래프를 생성하는 가장 간단한(brute-force)방법은 모든 노드의 쌍에 대하여 유사도를 계산하고, 계산된 유사도 값을 기반으로 각 노드의 k-NN을 찾는 것이다. 즉, 노드가 n개 있을 때 각 노드로 부터 나머지 (n-1)개의 노드와 유사도를 측정해야 하기 때문에 $O(n^2)$의 시간복잡도를 요구한다. 이는 대규모 문제에 대한 처리에는 확장성이 있지 않다. 90%와 100% 정확도를 가진 k-NN그래프를 사용하였을 때 성능 차이가 크지 않기 때문에[6,7] K-NN그래프 생성 속도를 증가

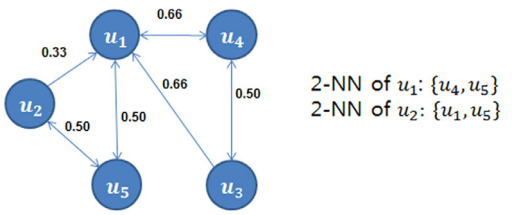


그림 1 2-NN 그래프의 예
Fig. 1 An example of a 2-NN graph

시키기 위해서 근사 k-NN그래프를 생성 알고리즘을 사용한다[2,5].

이전의 NN탐색 문제에 대해서 K-D trees[8], R-tree[9] 등 트리기반의 인덱싱(tree-based indexing)방법이 사용되어 왔다. 하지만 이 방법들은 고차원(high-dimension)의 데이터에 대해서 효율적이지 않다[10]. 반면에, LSH(Locality Sensitive Hashing)방법은 고차원의 데이터에 대해서도 효율적이며[11,12], 대규모(large-scale) 그룹 클러스터링에 적합한[13] 알고리즘 이다.

근사 k-NN그래프를 만드는 대표적인 방법은 LSH를 사용하여 먼저 유사한 노드를 그룹핑하고 그 그룹 안에서 brute-force한 방법으로 유사도를 계산하는 방법이다[5,14]. 즉, 이웃후보 그룹을 생성하는 과정과 이웃후보 그룹에서 유사도를 계산하는 과정으로 이루어진다. 이러한 방법을 사용하면 유사도를 계산하는 횟수를 줄이면서 유사도가 높은 노드를 찾을 수 있다.

다음과 같은 이유에서 k-NN그래프 생성의 과정은 분산환경인 아파치 하둡(Apache Hadoop, High-Availability Distributed Object-Oriented Platform) [15]기반의 맵리듀스(MapReduce)[16] 시스템에서 구현하는 것이 좋다. 아파치 하둡은 대량의 자료를 처리할 수 있는 큰 컴퓨터 클러스터에서 동작하는 분산 응용 프로그램을 지원하는 프리웨어 자바 소프트웨어 프레임워크이다.

맵리듀스(MapReduce)는 아파치 하둡(Apache Hadoop) 기반에서 동작하는 프로그램들의 알고리즘 수행 로직 스타일을 일컫는다. k-NN그래프 생성을 위해 오픈소스인 하둡 맵리듀스 프레임워크를 사용하였다. 맵리듀스는 분산 병렬 시스템에서 대용량 데이터를 처리하기 위해 고안된 프레임워크로써 데이터를 특정 함수 및 패턴에 따라 키와 값 집합으로 나누는 맵(Map) 단계와 이 집합을 키 별로 합치는 리듀스(Reduce) 단계로 구성된다. 하둡 파일 시스템(HDFS)은 대용량 데이터를 여러 노드에 분산시켜 병렬적인 맵(Map) 연산 작업을 처리한다.

LSH를 단일 노드 시스템에서 구현하였을 때의 단점은 여러 개의 해시테이블이 필요하며, 이는 많은 개수의 인덱스를 메모리상에 유지해야 한다는 것이다. 이는 맵리듀스를 이용한 분산환경에서 인덱스를 메시지화하기 때문에 해결할 수 있다. 또한, k-NN그래프 생성 과정에서 LSH로 버켓에 매핑한 후 같은 버켓에 속한 모든 쌍에 대해서 유사도를 검사하는데 상당한 시간을 소요하는 이 과정을 분산 처리하여 효율적이게 할 수 있다.

K-NN그래프 생성을 위한 두 과정 중 이웃후보 그룹을 생성하는 과정은 다음 두 가지가 중요하다. 유사도가 높은 사용자를 같은 그룹으로 만들어야 하며 각 그룹은 가능한 작아야 한다. 그룹의 크기가 커지게 되면 두 번째 과정인 유사도를 계산하는 맵에서 시간이 오래 걸리기 때문이다. 뿐만 아니라 하둡 환경에서 큰 그룹의 경우 맵-사이드 스큐의 원인이 되는 비용이 큰 레코드(Expensive Record)[17]이기 때문에 맵-사이드 스큐가 발생할 수 있다.

본 논문에서는 맵리듀스 환경에서 LSH기반의 효율적인 k-NN그래프 생성 알고리즘에 대한 연구를 진행하였다. 기존의 k-NN그래프 생성 알고리즘에서 발생하는 문제를 정리하였고 효율적인 이웃후보그룹생성을 위한 알고리즘을 제안하였다.

논문의 구성은 다음과 같다. 2장에서 근사 그래프 생성에 대한 기존 방법들을 살펴보고 3장에서는 본 논문의 방법에 대해 자세히 설명한다. 4장에서 실험 결과를 제시하고 5장에서 결론 및 향후 연구에 대해 언급한다.

2. 관련연구

주어진 노드로부터 유사도를 구하고자 할 때, 모든 쌍의 유사도를 계산하는 것은 $O(n^2)$의 시간복잡도를 요구하기에 확장성이 있지 않다. 또한, 한 개의 해시테이블을 이용하여 사용자를 매핑하면, 적어도 한 개의 같은 아이템에 대한 평점을 공유하는 경우 같은 그룹에 들어가게 된다. 이는 빈도가 높은 아이템 때문에 일부 그룹의 크기가 커지는 문제가 있다. 위의 문제를 해결하고자 [1]에서는 맵리듀스 환경에서 LSH를 이용하여 그룹을 만들어주는 방법을 소개하였다. 해시함수로는 MinHash를 사용하였다.

LSH는 근사 k-NN그래프 생성 문제에서 효율적인 기법으로 본 논문의 기초가 되므로 본 절에서 간략하게 소개한다. LSH는 고차원(high-dimensional)의 데이터를 저 차원의 데이터로 바꾸는 것으로, LSH의 핵심은 벡터간 유사성이 보존되도록 해시(hash)를 하는 것이다. 그러므로, 유사한 노드가 높은 확률로 같은 버켓(bucket) 안에 들어가게 해준다. LSH기법의 매력적인 특징은 이론적으로 최악의 경우에도 성능을 보장하며[11], 실제로도 일정하거나 sub-linear 탐색시간이 걸린다[5].

이를 바탕으로 유사도가 높은 쌍을 찾아주기 위해서 LSH로 그룹을 만들어준 후 그룹내의 모든 쌍에 대해서 유사도 계산을 하는 방법이 소개되었다[5,14,18]. [14]에서는 MinHash, LSH방법을 이용하여 k-NN그래프를 생성하고, [18]에서는 MinHash, LSH방법으로 일정한 유사도 이상의 쌍을 찾아주는 ε-NN탐색을 한다. 그룹내의 모든 쌍에 대해서 유사도를 계산하기 때문에 다음 두 가지 조건이 중요하다[5].

가) 유사도가 높은 노드는 같은 그룹에 속하여야 한다.

나) 각 그룹의 사이즈는 가능한 작아야 한다.

[14,18]에서는 [1]에서의 그룹 사이즈를 작게 하는 방법을 사용한다. 이 방법은 한 개의 MinHash값으로 그룹을 생성하면 빈도가 높은 아이템 때문에 큰 그룹이 생성되는 것을 막고자 p개의 MinHash값을 이어 붙여서 그룹을 생성 한다. 하지만 이 방법은 큰 그룹뿐만 아니라 작은 그룹도 더 작게 만들어서 충분한 쌍을 찾지 못한다는 단점이 있다. 따라서, 위와 같은 n명의 사용자에 대한 그룹생성과정을 q번 수행하여 좀 더 많은 쌍을 갖는 그룹들을 찾고자 하였다. 이 방법은 이웃후보생성과정에서 중간 생성 데이터가 늘어나게 되고 이웃후보생성과정을 q배만큼 더 늘리기 때문에 K-NN그래프 생성을 함에 있어서 한계가 있다.

큰 그룹 사이즈를 방지하기 위한 다른 방법으로 [5]에서는 전체 노드에 대해서 해시의 값으로 정렬한 후 일정한 크기의 블록 사이즈로 그룹을 나누어 모든 그룹의 사이즈를 같게 만드는 알고리즘을 소개하였다. 하지만 이 방법은 병렬처리 알고리즘이 아니며, 전체 노드에 대해서 해시의 값으로 정렬하기 때문에 메모리를 많이 차지한다는 단점과 해시의 값이 다른 노드들도 같은 그룹내에 들어갈 수 있다는 단점이 있다. 위 방법에서 128GB의 RAM으로 실험을 하였는데 우리의 실험환경에서는 2만개의 노드를 가진 데이터에 대해서 힙(heap) 사이즈가 충분하지 않아서 전체노드에 대해서 정렬을 하는 것이 불가능했다.

3. LSH를 이용한 k-NN 그래프 생성

3.1 문제 정의

n개의 노드의 집합 $U = \{u_1, u_2, \cdots, u_n\}$과 유사도 $S(u_i, u_j)$가 주어졌을 때, U의 k-NN그래프는 u_j 가 u_i 의 가장 유사한 k개의 노드에 속할 때, 노드 i에서 j로 간선이 있는 방향그래프(directed graph)이다. 본 논문에서는 사용자가 노드이며, 각 노드는 아이템 평점을 준 로그셋 L_u 을 갖는다. S 는 자카드 계수(Jaccard coefficient)를 사용하였고 다음과 같이 표현할 수 있다.

$$S(u_i, u_j) = \frac{\left| L_{u_i} \cap L_{u_j} \right|}{\left| L_{u_i} \cup L_{u_j} \right|}$$

즉, 각 사용자의 자카드 계수가 높은 k개의 사용자를 찾아주는 그래프를 만드는 것이다. K-NN그래프를 생성하기 위해서 그림 2와 같이 크게 이웃후보 그룹을 생성하는 과정과 이웃후보 그룹 내에서의 유사도를 계산하는 과정을 갖는다. 각 과정은 한 개의 맵-리듀스로 구

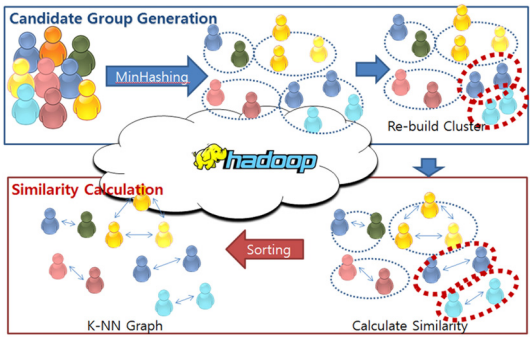


그림 2 K-NN그래프 생성을 위한 과정
Fig. 2 The overview of k-NN graph construction

현되어 총 2개의 맵-리듀스 잡을 통하여 k-NN그래프를 생성한다.

3.2 이웃후보 그룹생성

본 절에서는 MinHash를 이용한 이웃후보 그룹을 생성하는 것과 이웃후보 그룹을 재구성하는 것에 대하여 설명한다.

3.2.1 MinHash를 이용한 이웃후보 그룹 생성

MinHash[19]는 LSH기법의 한가지로 해시 충돌이 일어날 확률이 자카드 계수(Jaccard coefficient)와 같은 해시 함수이다. 유니버셜 해시함수를 사용하며, [1]에서는 한 사용자에 대해서 1개의 해시함수를 사용하여 작은 p개의 해시값을 사용자의 그룹ID로 만들어 그룹을 만든다. 그리고 이 작업을 시드(seed)가 다른 해시함수로 q번 반복하여 한 사용자에 대해서 q개의 그룹ID를 만든다. 본 논문에서도 위와 같은 방법을 사용하여 첫 번째 맵에서 알고리즘1과 같이 그룹ID를 만들어 유사도 후보그룹을 만들었다.

알고리즘 1. 후보 그룹 생성: 맵
Algorithm 1. Candidate group generation Map

```
Input file: user, list<item>
Map(p,q)
read input file
for each user do
    for i..q do
        Hashed vector = MinHash(list<item>)
        group-id = 1~p smallest values in Hashed vector
        Emit(group-id, user)
    end for
end for
```

3.2.2 이웃후보 그룹 재구성

본 절에서는 첫 번째 리듀스에서 사이즈가 큰 그룹에 대해서 처리해주는 두 가지 방법을 제시한다.

3.2.2.1 일정한 그룹사이즈로 그룹 재구성

첫 번째 방법은 사이즈가 큰 그룹을 일정한 사이즈로 다시 나누어 재구성 하는 방법이다. 이 방법은 [5]에서 최대 사이즈를 제한한 방법이 모티브가 되었다. 본 논문에서는 해시값이 같은 노드만 그룹에 들어가게끔 먼저 그룹을 만들어준 후 일정블록 사이즈가 넘는 그룹의 경우 그룹 아이디를 바꿔서 그룹을 일정한 블록사이즈의 그룹으로 나누었다. 이 방법은 기존의 방법에 추가적인 데이터 구조를 만들지 않아도 되며 추가적인 계산을 하지 않기 때문에 추가비용이 적다.

알고리즘 2(a). 후보 그룹 생성: 리듀스
Algorithm 2(a). Candidate group generation: Reduce

```
Input: group-id, list<user>, group-size threshold
Reduce(input)
for user in list<user> do
    userList.add(user)
    if userList.size > group-size threshold
        Emit(group-id + i++, userList)
        userList.clear
end for
Emit(group-id, userList)
```

그림 3에서는 임계 블록사이즈 값이 3인 예를 보여주고 있다. 사용자에 대해서 1개의 해시값을 만들고 그 해시값으로 그룹을 구성한다. 그 후 임계 블록사이즈 값을 넘는 그룹에 대해서 그룹사이즈를 3 이하로 재구성한다.

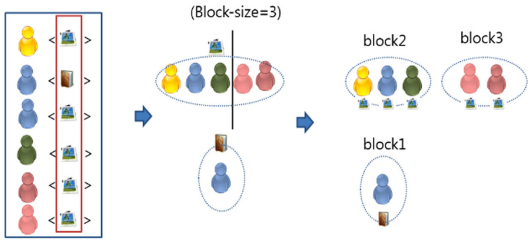


그림 3 일정한 그룹사이즈로 그룹 재구성 예
Fig. 3 An example of candidate group re-building

3.2.2.2 재 해싱을 통한 그룹 재구성

두 번째 방법은 재 해싱(re-hashing)을 통하여 사이즈가 큰 그룹을 작은 여러 개의 그룹으로 나누는 방법이다. 이 방법은 [20]에서 제시한 재 해싱 개념이 모티브가 되었다. [20]에서는 k-NN 탐색을 할 때 쿼리포인

트가 속한 그룹이 임계 값을 넘으면 그 그룹에 속한 모든 포인트에 대해서 재 해시를 하여 그룹을 나누었다.

두 번째 방법 또한 첫 번째 방법과 마찬가지로 일정한 임계 값을 정하고 임계 값을 넘은 그룹의 경우 해싱을 한번 더 하여 해시 값을 하나 더 만들어서 그룹ID에 붙인다. 즉, 이 그룹의 경우 처음 해시를 했던 p값보다 하나 더 많은 p+1개의 해시 값을 그룹ID로 갖는다. 만약 재구성한 그룹이 다시 임계 값을 넘는다면 재귀적으로 앞에서 설명한 방법으로 그룹을 재구성한다. 따라서, 첫 번째 방법과 마찬가지로 모든 그룹의 사이즈는 임계 값을 넘지 않아 큰 그룹을 만들지 않는다.

알고리즘 2(b). 재 해시 후보그룹 생성: 리듀스
Algorithm 2(b). Candidate group generation: Reduce

```
Input: group-id, list<user>, group-size threshold, k
Reduce(input)
for user in list<user> do
    userList.add(user)
    if userList.size > group-size threshold
        oversize=true
end for
    if oversize
      reBuild(userList, ++k, threshold)
    else Emit(group-id, userList)
reBuild(userList,k,threshold)
    map=makeKPlusMap(userList, k)
    for list in lists=listFromMap(map)
        if list.size<threshold
            Emit(group-id, list)
        else reBuild(list, ++k, threshold)
End reBuild
makeKPlusMap(ist, k)
For user in list do
    newGroup-id =MinHash(user.items, k)
    map.add(newGroup-id, user)
Return map
```

첫 번째 방법과 다른 점은 재 그룹을 할 때 재 해시를 하여 kMinHash에서 k를 증가함으로써 그림 4와 같이 더 유사도가 높은 쌍을 같은 그룹에 속하게 한다는 점이다. 첫 번째 방법은 모든 사용자의 경우 일정한 k값으로 그룹ID를 만들지만 두 번째 방법은 큰 그룹에 속한 사용자의 경우 k값이 더 큰 값으로 그룹ID를 만들게 된다. 따라서, 기존의 방법에서 k를 증가 시킬 때, 작은 그룹도 더 작아지는 문제가 있었는데 두 번째 방법을 사용하면 작은 그룹은 k가 더 큰 것으로 해싱하지 않기에 그대로 보존 된다는 장점이 있다. 하지만 이 방법은 추가적으로 해시를 해야 하기에 추가시간이 들어간다. 전체사용자를 n, 임계블록사이즈를 blockSize라고 할 때, 재 해싱을 하지 않는 경우 Minhash를 하여 그룹을 만들어주는데 O(n * k * d)시간 복잡도가 걸렸는데 재 해싱을 하게 되면 O(ε * k * d) 만큼의 시간복잡도가 추가되어 O((n + ε) * k * d)로 증가한다. ε는 n에 비하여 작은 수 이고 재 해싱 방법의 초기 k값을 1로 시작하지 않고 적절하게 올린 후 사용하기 때문에 추가되는 시간은 크지 않으며, k는 비교적 작은 수이기에 재 해싱을 하는 방법과 하지 않는 방법 모두 $O(n*d)$가 된다. 이 두 번째 방법으로 그룹을 나누면 다음 과정인 유사도 계산 과정에서 유사도 계산을 하는 쌍을 줄이게 되는 장점을 이용하였다. 한 그룹에 대해서 최악의 경우 유사도 계산의 시간복잡도를 $O(n^2)$에서 $O(blockSize^2)$로 줄였으며, 전체 사용자에 대하여 MinHash테이블을 만드는 과정을 q번 반복 할 때, 두 번째 과정에서 한 사용자에 대하여 가장 유사도가 높은 사용자를 추출하는 과정에서의 정렬에 대한 최악의 경우 시간복잡도를 $O(n*logn)$에서 $O(q*blockSizelog(q*blockSize))$로 줄였으며 공간복잡도를 $O(n)$에서 $O(q*blockSize)$로 줄여서 메모리에 부담을 덜어주었다.

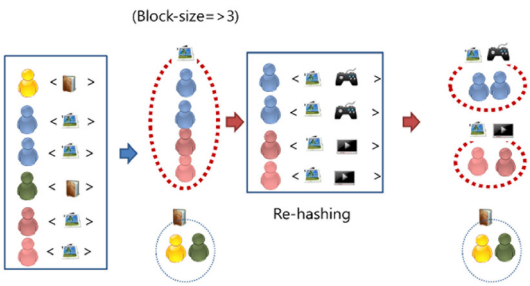


그림 4 재해싱을 통한 그룹 재구성 예
Fig. 4 An example of candidate group re-building using hashing

3.3 이웃후보 그룹내의 유사도 검사 및 k-NN추출

이웃후보그룹이 완성된 후 이웃후보 그룹내의 모든 쌍에 대해서 유사도를 검사한다. 한 개의 맵-리듀스로 구성되는데 먼저, 맵은 그룹내의 $\frac{group\ size(group\ size-1)}{2}$ 쌍을 만들고 각각의 쌍에 대해서 유사도를 계산한다. 그리고 아웃풋으로 사용자를 key로 후보사용자와 유사도 값을 value로 하여 리듀스 단계로 보낸다. 리듀스에서는 각 사용자에 대하여 후보사용자와 유사도 값의 리스트를 받게 되고 각 사용자에 대해서 후보사용자를 유사도 값으로 정렬하여 유사도가 높은 k명의 사용자를 찾는다. 위의 모든 과정을 통해 근사 k-NN그래프를 만든다. 앞에서 이웃후보 그룹 생성과정에서 그룹사이즈의 제한을

두었기에 이 과정에서 한 사용자에 대하여 가장 유사도가 높은 사용자를 추출하는 과정에서의 정렬에 대한 시간복잡도를 $O(n*logn)$에서 $O(q*blockSizelog(q*blockSize))$로 줄였으며 공간복잡도를 $O(n)$에서 $O(q*blockSize)$로 줄였다.

알고리즘 3. 유사도 계산: 맵, 리듀스
Algorithm 3. Similarity Calculation: Map, Reduce

```
Input: group-id, list of users
Map(intput)
make all pair   for users
for each user do
  sim=similarity calculation(ui, uj)
  Emit(ui, (uj,sim))
  Emit(uj, (ui,sim))
end for

Input: user, list<(user,sim)>   as key, values
Reduce(intput)
for each (user ,sim) in values do
    map.add(user,sim)
end for
Map.sort by sim
Emit(top k user,sim)
```

4. 실험

본 실험은 잡트래커(job tracker)인 마스터(master) 컴퓨터 1대와 태스크트래커(task tracker)인 슬레이브(slave) 컴퓨터 10대의 하둡 클러스터에서 수행했다. 각 노드에 해당하는 컴퓨터는 3.1GHZ 쿼드코어 Intel Core i5-2400 CPU, 4GB RAM과 4TB의 하드디스크로 구성된다.

4.1 실험설정

본 절에서는 본 논문에서 사용한 데이터 셋, 비교 알고리즘, 성능평가 기준에 대해서 설명한다.

4.1.1 데이터 셋

데이터는 표 1과 같이 무비렌즈(MovieLens)[21]에서 제공하는 252MB의 무비렌즈 데이터와 223MB의 뉴욕타임즈 데이터[22]를 사용하였다. 무비렌즈 데이터를 [1]에서 사용한 방법으로 이진화 하였다. 각 사용자에 대해서 점수가 사용자의 평균점수보다 높을 경우 1로 이진화 하였고 그렇지 않을 경우 0으로 하였다. 뉴욕 타임즈 데이터는 bags-of-word형식의 데이터이다. 본 논문에서는 20,000개의 문서에 대해서 이진화하기 위해 [16]에서의 방법처럼 빈도가 있는 경우 1로 그렇지 않을 경우 0으로 이진화하였다.

표 1 데이터 셋 설명
Table 1 Data set descriptions

	MovieLens	NY Times
Size	71,567	20,000
Dimensionality	10,681	102,660

4.1.2 비교 알고리즘

본 실험에서 제시한 알고리즘과 기존의 알고리즘을 비교하기 위해서 본 실험에서 제시한 첫 번째 방법인 일정한 그룹사이즈로 재구성한 방법을 LSH-R로, 두 번째 방법인 재 해싱을 이용한 방법을 LSH-K+로 표현하고 [14]에서 제시한 방법의 그래프생성 알고리즘을 LSH-B로 표현한다.

4.1.3 성능 평가 기준

본 실험의 평가 기준으로는 Scan Rate와 정확도를 나타내는 Accuracy를 사용하였다. G가 정확한 k-NN그래프를 나타내고 E(·)가 그래프에서의 간선(directed edge)를 나타내고 |·|가 집합의 개수일 때, 근사 k-NN 그래프 G′의 정확도는 다음과 같이 정의한다.

$$Accuracy(G') = \frac{|E(G') \cap E(G)|}{|E(G)|}$$

데이터의 전체 노드가 n개 있을 때, Brute-force방법은 n(n-1)/2번의 유사도 계산한다. Scan Rate는 실제로 유사도를 계산한 횟수와 Brute-force방법에서의 유사도 계산 횟수의 상대적 비율로써 다음과 같이 정의한다.

$$\text{Scan Rate} = \frac{\#\ actual\ similarity\ calculation}{n(n-1)/2}$$

4.2 실험 결과

4.2.1 시간과 정확도

다양한 정확도의 그래프를 만들기 위해서 해시테이블을 만드는 개수(변수q)를 다르게 하며 실험하였다. 그림 5

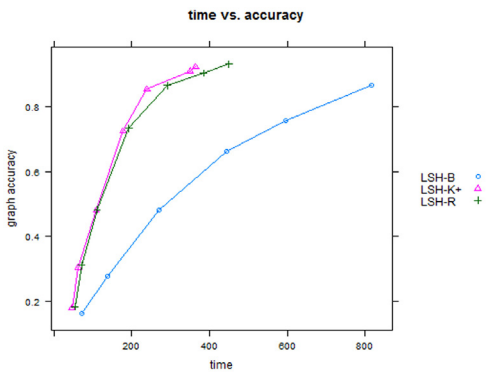


그림 5 여러 방법에 대한 시간과 정확도 비교
Fig. 5 Graph accuracy versus construction time for different methods

는 여러 가지 방법에 대해서 다양한 정확도의 그래프를 생성하는데 걸리는 시간을 측정하여 비교하였다. 그래프의 x축은 수행시간(초)를 나타내며, y축은 정확도를 나타낸다.

LSH-K+방법이 가장 빠른 시간에 90%정확도의 근사 그래프를 생성함을 확인하였으며, 이는 LSH-B방법보다 두 배 이상 빠르게 그래프를 생성함을 볼 수 있다.

4.2.2 정확도와 상대 유사도 계산 횟수(Scan Rate)

그림 6은 알고리즘이 얼만큼의 유사도 계산을 하고 어떠한 정확도의 그래프를 생성하는지 나타낸다. x축은 Scan Rate이며, y축은 생성한 그래프의 정확도를 나타낸다. LSH-K+알고리즘이 가장 적은 유사도 계산을 하면서 높은 정확도의 그래프를 생성한다.

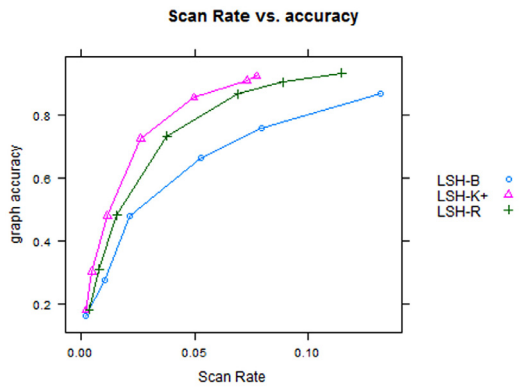


그림 6 정확도와 Scan Rate 비교

Fig. 6 Scan rate of the proposed method with respect to graph accuracy

4.2.3 클러스터 증가에 따른 영향

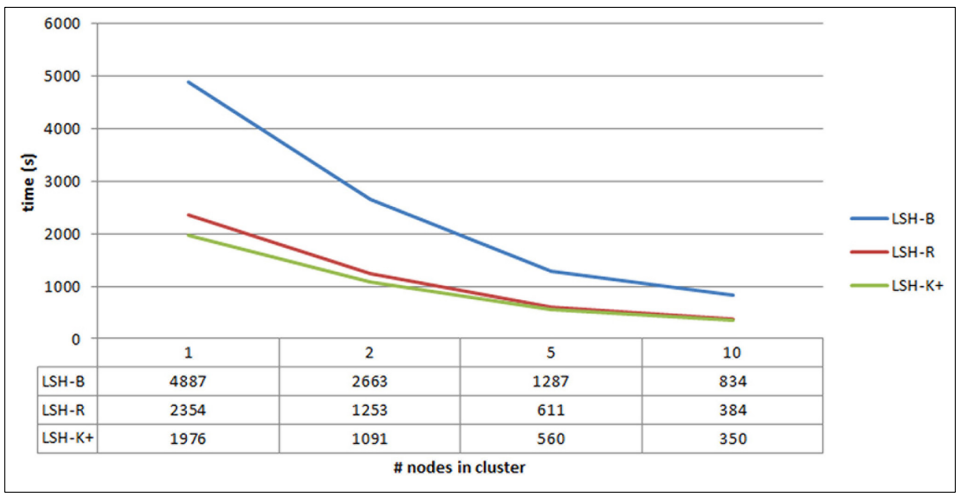
본 논문이 제안하는 알고리즘이 클러스터 노드의 변화에 따른 성능의 변화를 확인하고자 무비렌즈 데이터셋에 대하여 태스크 트래커인 슬레이브 컴퓨터의 개수를 1개에서 10개로 변화시켜 가며 실험하였다. 그림 7은 태스크 트래커의 개수의 변화에 따른 90%정확도의 그래프를 생성하는데 걸리는 시간이다. x축은 태스크 트래커의 개수, y축은 소요되는 시간을 의미한다. 태스크 트래커가 증가함에 따라 그래프생성에 소요되는 시간이 짧아지는 것을 볼 수 있다.

5. 결론 및 향후 연구

본 논문에서는 맵리듀스(MapReduce)환경에서의 효율적인 k-NN그래프 생성 방법을 제시하였다.

본 논문의 방법은 LSH기법인 MinHash로 사용자를 작은 그룹으로 나누고, 각 그룹내에서 유사도를 측정한다. 그룹 내에서 brute-force하게 유사도를 계산하기 때문에 유사도가 높은 사용자를 그룹에 속하게 하면서 작은 그룹을 만드는 것이 중요하다. 본 논문에서는 그룹의 재구성방법을 제안하였다. 그룹의 최대 사이즈를 조절하여 큰 그룹에 속한 일부 사용자에 대해서는 해시 값의 개수를 다르게 하는 방법으로 그룹을 재구성하였다. 실험 결과 LSH-K+를 사용한 본 논문의 방법이 기존의 방법보다 더 적은 비율의 유사도 측정을 통해 정확도가 더 높은 그래프를 생성하는 것을 확인하였다.

향후 연구로는 더 정밀한 그룹생성을 위한 기법에 대해 연구하고자 하며, 구축한 맵리듀스 알고리즘의 최적화 기법을 연구할 계획이다. 또한 다양한 유사도 계수에 적용 가능한 알고리즘에 대해 연구하고자 한다.



	1	2	5	10
LSH-B	4887	2663	1287	834
LSH-R	2354	1253	611	384
LSH-K+	1976	1091	560	350

그림 7 클러스터 개수에 따른 성능변화

Fig. 7 Scalability of Algorithm

References

[1] A. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," *Proc. 16th Int. Conf.*, pp. 271-280, 2007.

[2] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," *Proc. 20th Int. Conf. World wide web - WWW '11*, pp. 577-586, 2011.

[3] M. R. Brito, E. L. Chávez, A. J. Quiroz, and J. E. Yukich, "Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection," *Statistics & Probability Letters*, Vol. 35. pp. 33-42, 1997.

[4] O. Boiman, E. Shechtman, and M. Irani, "In defense of nearest-neighbor based image classification," *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[5] Y. Zhang, K. Huang, G. Geng, and C. Liu, "Fast k NN Graph Construction with Locality Sensitive Hashing," *Knowl. Discov. Databases*, pp. 660-674, 2013.

[6] J. Chen, H. Fang, and Y. Saad, "Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection," *J. Mach. Learn. Res.*, Vol. 10, No. 2009, pp. 1989-2012, 2009.

[7] Y. Park, S. Park, S. Lee, and W. Jung, "Fast collaborative filtering with a k-nearest neighbor graph," *BigComp*, pp. 92-95, 2014.

[8] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, Vol. 18. pp. 509-517, 1975.

[9] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," *Proc. of the 1984 ACM SIGMOD International Conference on Management of Data - SIGMOD '84*, pp. 47-57, 1984.

[10] R. Weber, H. J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," *Proc. 24th VLDB Conf.,* Vol. New York C, pp. 194-205, 1998.

[11] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," *STOC '98: Proc. of the thirtieth annual ACM symposium on Theory of computing*, pp. 604-613, 1998.

[12] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, "Efficient search for approximate nearest neighbor in high dimensional spaces," *STOC '98: Proc. of the thirtieth annual ACM symposium on Theory of computing*, pp. 614-623, 1998.

[13] L. Li, D. Wang, T. Li, D. Knox, and B. Padmanabhan, "SCENE: a scalable two-stage personalized news recommendation system," *SIGIR*, pp. 125-134, 2011.

[14] L. Hsieh and G. Wu, "Two-stage sparse graph construction using MinHash on MapReduce," *ICASSP*, pp. 1013-1016, 2012.

[15] "Apache Hadoop," [Online]. Available: http://hadoop.apache.org/.

[16] J. Dean and S. Ghemawat, "MapReduce : Simplified Data Processing on Large Clusters," *Commun. ACM*, Vol. 51, pp. 1-13, 2008.

[17] Y. Kwon and M. Balazinska, "A study of skew in mapreduce applications," Open Cirrus Summit, 2011.

[18] R. Szmit, "Locality Sensitive Hashing for Similarity Search Using MapReduce on Large Scale Data," *IIS*, 2013, Vol. 7912, No. LNCS, pp. 171-178.

[19] A. Z. Broder, "On the resemblance and containment of documents," *Proc. Compression Complex. Seq. 1997 (Cat. No.97TB100171)*, 1997.

[20] Z. Yang, W. Oop, and Q. Sun, "Hierarchical non-uniform locally sensitive hashing and its application to video identification," *ICIP*, pp. 743-746, 2004.

[21] "MovieLens," [Online]. Available: http://grouplens.org/datasets/movielens/.

[22] "NYTimes news articles," [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Bag+of+Words.

이 인 회

2013년 아주대학교 정보컴퓨터공학 학사 2015년 서울대학교 컴퓨터공학부 석사 2015년 삼성전자 무선사업부 Mobile Security Technology Group 사원. 2015년~현재 금융결제원 전자금융부 사원. 관심분야는 데이터베이스, 빅데이터 처리, 컴퓨터 보안, IC카드

오 혜 성

2012년 서울대학교 컴퓨터공학부 학사 2012년~현재 서울대학교 컴퓨터공학부 석박사통합과정. 관심분야는 데이터베이스, 빅데이터 처리

김 형 주

1982년 서울대학교 전산학과 학사. 1985년 Univ. of Texas at Austin 석사. 1988년 Univ. of Texas at Austin 박사. 1988년~1990년 Georgia Institute of Technology 조교수. 1991년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 데이터베이스, 시멘틱웹, 온톨로지, 빅데이터