JANT 한국항행학회논문지 Journal of Advanced Navigation Technology

기반 기술

J. Adv. Navig. Technol. 18(5): 468-475, Oct. 2014

동시발생 행렬과 하둡 분산처리를 이용한 추천시스템에 관한 연구

A Study On Recommend System Using Co-occurrence Matrix and Hadoop Distribution Processing

김 창 복[1*] · 정 재 필[2]

[1]가천대학교 에너지 IT학과

[2]가천대학교 전자공학과

Chang-Bok Kim[1*] · Jae-Pil Chung[2]

Department of Energy IT, Gachon University, Gyeongi-do 461-701, Korea

Department of Electronic Engineering, Gachon University, Gyeongi-do 461-701, Korea

[요 약]

추천시스템은 선호 데이터가 대형화, 컴퓨터 처리능력과 추천 알고리즘 등에 의해 실시간 추천이 어려워지고 있다. 이에 따라 추천시스템은 대형 선호데이터를 분산처리 하는 방법에 대한 연구가 활발히 진행되고 있다. 본 논문은 하둡 분산처리 플랫폼과 머하웃 기계학습 라이브러리를 이용하여, 선호데이터를 분산 처리하는 방법을 연구하였다. 추천 알고리즘은 아이템 협업필터링과 유사한 동시발생 행렬을 이용하였다. 동시발생 행렬은 하둡 클러스터의 여러 노드에서 분산처리를 할 수 있으며, 기본적으로 많은 계산량이 필요하지만, 분산처리과정에서 계산량을 줄일 수 있다. 또한, 본 논문은 동시발생 행렬처리의 분산 처리과정을 4 단계에서 3 단계로 단순화하였다. 결과로서, 맵리듀스 잡을 감소할 수 있으며, 동일한 추천 파일을 생성할 수 있었다. 또한, 하둡 의사 분산모드를 이용하여 데이터를 처리하였을 때 빠른 처리속도를 보였으며, 맵 출력 데이터가 감소되었다.

[Abstract]

The recommend system is getting more difficult real time recommend by lager preference data set, computing power and recommend algorithm. For this reason, recommend system is proceeding actively one's studies toward distribute processing method of large preference data set. This paper studied distribute processing method of large preference data set using hadoop distribute processing platform and mahout machine learning library. The recommend algorithm is used Co-occurrence Matrix similar to item Collaborative Filtering. The Co-occurrence Matrix can do distribute processing by many node of hadoop cluster, and it needs many computation scale but can reduce computation scale by distribute processing. This paper has simplified distribute processing of co-occurrence matrix by changes over from four stage to three stage. As a result, this paper can reduce mapreduce job and can generate recommend file. And it has a fast processing speed, and reduce map output data.

Key word : Co-occurrence matrix, Collaborative filtering, Hadoop, Matrix factorization, Recommender system.

http://dx.doi.org/10.12673/jant.2014.18.5.468

Received 5 September 2014; **Revised** 20 October 2014

Accepted (Publication) 6 October 2014 (30 October 2014)

***Corresponding Author; Chang-Bok Kim**

Tel : +82-32-820-4294

E-mail : cbkim@gachon.ac.kr

Ⅰ. 서 론

최근 디지털 정보의 폭발적인 증가에 따라 인터넷 쇼핑몰이나 콘텐츠 서비스 제공자들은 정보 과잉에 대한 문제점들을 해결하고, 고객의 성향에 따라 필요한 정보만을 추출하여 제공할 수 있는 개인화 서비스에 대한 관심이 고조되고 있다. 추천시스템(recommender system)은 특정 사용자의 선호도를 예측하여 관심을 가질만한 N개의 아이템을 추천하기 위한 개인화 정보 필터링 기법이다[1].

추천시스템은 콘텐츠필터링(contents filtering)과 협업필터링(collaborative filtering)추천 기법이 있다. 콘텐츠필터링은 아이템의 메타정보를 이용하여 아이템 간 유사도를 기반으로 추천하는 기법이다. 콘텐츠필터링은 아이템 내용의 정의가 어려우며, 메타정보를 수작업으로 분류해야 하며, 유사한 아이템을 반복 추천하는 문제점이 있다[2]. 협업필터링은 사용자의 구매이력이나 평가내역을 이용하여, 사용자 및 아이템 간 유사도를 산출하여 추천하는 기법이다. 협업필터링은 선호데이터가 많지 않은 경우에는 유용하지 못하며, 대용량의 데이터를 처리할 경우 계산량이 많아 속도가 느려지는 단점이 있다[3],[4]. 추천시스템은 초기에 콘텐츠필터링이 다수 연구 되었지만, 양질의 메타정보 구성이 어려우며, 협업필터링 기법에 비해 정확도가 낮은 것으로 알려져 있어, 협업필터링을 기반으로 하는 추천시스템에 대한 연구가 많이 이루어지고 있다.

최근 추천시스템은 선호데이터가 대형화되면서, 컴퓨터 처리능력과 추천 알고리즘에 의해 실시간 추천이 어려지고 있다. 하둡은 빅 데이터 분산저장 및 처리를 위한 플랫폼으로, 분산저장을 위한 HDFS(Hadoop distributed file system)와 분산처리를 위한 맵리듀스(MapReduce) 프로그래밍 모델로 구성된다[5],[6]. 하둡은 데이터 마이닝이나 통계 분야 등에서 빅 데이터 처리 기술로 많이 사용되고 있으며, 선호데이터의 대형화에 따라 실시간 추천시스템에 적용하는 연구가 시도되고 있다. 그러나 비교적 복잡한 연산을 수행하는 추천 알고리즘을 맵리듀스 함수로 설계하는 것은 쉽지 않으며, 설계된 맵리듀스 모듈에 따라 전체 시스템의 성능에 많은 영향을 미치게 된다.

본 논문은 선호데이터를 분산처리 플랫폼인 하둡(Hadoop)과 기계학습 라이브러리인 머하웃(Mahout)을 이용하여, 추천용 결과 파일을 생성하는 방법에 대해서 연구하였다. 추천 알고리즘은 아이템 협업필터링과 유사한 동시발생 행렬(co-occurrence matrix)을 이용하였다. 동시발생 행렬은 하둡 클러스터의 여러 노드에서 맵리듀스로 분산처리가 용이한 방법이며, 기본적으로 많은 계산량이 필요하지만, 하둡 분산처리과정에서 계산량을 줄일 수 있다. 본 논문은 동시발생행렬을 처리하기 위해 4단계의 처리과정을 3단계로 축소함으로서, 맵리듀스 잡을 감소하며, 처리속도를 높이고자 시도하였다.

본 논문은 2장에 관련연구로서 대용량 데이터 추천시스템과 동시발생 행렬에 대해서 서술하였으며, 3장에 동시발생행렬을 분산처리하기 위한 4단계 기법을 서술하고, 본 논문에서 제한 3

그림 1. 행렬 분해 협업필터링
Fig. 1. Matrix factorization collaborative filtering.

단계기법에 대해서 서술하였다. 또한, 4장에서 제시한 기법을 실험 및 검토를 하였으며, 최종적으로 결론을 내렸다.

Ⅱ. 관련연구

2-1 대용량 추천시스템

대용량 데이터 추천시스템은 추천 아이템의 정확도 개선에 가장 효과적인 것으로 알려져 있는 행렬 분해(matrix factorization) 협업필터링이 중점적으로 제안되고 있다[7],[8]. 그림 1에 행렬 분해 협업필터링에 대해서 나타냈다.

원본 행렬 [R]은 사용자와 아이템 선호 값에 대한 행렬이다. 행렬 분해 협업필터링은 원본 행렬과 가장 유사한 행렬을 생성하는 사용자 요인 행렬 [U]와 아이템 요인 행렬 [M]을 생성하여, 사용자 평가점수가 없는 아이템에 선호 값을 예측하는 기법이다. 행렬분해 기반 협업필터링은 [U]·[M]과 원본 행렬 [R]의 유사도를 측정할 때, 평가된 정보 값에 대한 RMSE(root mean squared error)를 사용한다. 이를 기반으로 실제 추천 아이템에 대한 정확성을 보장하지 못하는 문제를 해결하기 위해 제한자를 추가할 수 있다. 최종적으로 아래와 같은 식으로 기술할 수 있다.

$$\min \sum_{r_{ij} \neq ?} (r_{ij} - u_i^T \cdot m_j)^2 + \lambda_1 \sum u_i^2 + \lambda_2 \sum m_j^2 \qquad (1)$$

행렬 분해 협업필터링 기법은 다른 알고리즘에 비교하여 많은 양의 계산량을 필요하기 때문에, 실제 서비스에 실시간으로 적용되기에는 문제점이 있다. 이러한 문제를 해결하기 위하여 알고리즘을 분산화 및 병렬화를 위한 기법들이 연구되고 있다. 하둡은 효과적인 병렬처리 기술로 알려지면서 데이터 마이닝이나 통계 분야 등에서 대용량 데이터 처리 기술로 널리 사용되고 있으며, 선호데이터의 대형화에 따라 많은 계산량을 필요로 하는 추천시스템에 적용하는 연구가 시도되고 있다[9],[10].

2-2 동시발생 행렬

동시발생 그룹화(co-occurrence grouping)는 객체 간의 연관성을 찾아내는 데이터 마이닝 기법이다. 동시발생 그룹화는 작

	1	2	3	4	5	6	7		U		R
1	4	1	2	0	2	2	2		1.5		23.0
2	1	3	0	0	0	1	1		3.0		12.5
3	2	0	2	0	1	1	1		0.0		10.0
4	0	0	0	0	0	0	0		0.0		0.0
5	2	0	1	0	3	1	2		5.0		22.0
6	2	1	1	0	2	2	1	×	0.0	=	18.0
7	2	1	1	0	2	0	4		2.0		24.0

그림 2. 동시발생 행렬과 추천 벡터
Fig. 2. Co-occurrence matrix and recommend vector.

업의 틀은 여러 가지 있지만, 아이템 A가 발생하면 아이템 B도 발생할 가능성이 있다는 규칙이 일반적이다. 예를 들어, 분석의 대상이 되는 두 개의 변수는 거래와 아이템으로 구성되며, 거래는 한 사용자에 의한 구매를 의미하고, 아이템은 구매를 통해 구입된 물건이라 할 때, "아이템 A를 포함한 구매는 아이템 B도 포함한다."라 해석하고 A=>B와 같이 표현한다[11].

동시발생 행렬은 아이템 간의 동시발생 횟수를 행렬로 나타낸 것으로, 두 아이템이 동시에 발생한 횟수가 많을수록 더 많은 관련이 있거나 유사하다는 개념이다. 그림 2는 동시발생행렬과 사용자 선호 값을 이용하여 추천 벡터인 R을 계산하는 방법에 대해서 나타냈다.

그림은 7개 아이템에 대한 동시발생 행렬으로, 7×7 행렬로 구성된다. 각 행은 특정 아이템과 다른 모든 아이템 간의 동시발생횟수이며, 아이템 A와 아이템 B의 동시발생 횟수와 아이템 B와 아이템 A와의 동시발생 횟수는 동일하기 때문에 좌우 대칭이다. 추천 벡터 R은 각 행과 사용자의 선호벡터의 내적을 통해서 계산된다. 사용자의 아이템 6에 대한 R은 다음과 같이 계산된다.

$$R_6 = (2\times1.5)+(1\times3.0)+(1\times0.0)+(1\times0.0) \\ +(2\times5.0)+(1\times0.0)+(1\times2.0)=18.0 \quad (2)$$

사용자는 아이템 1, 2, 5, 7에 대한 선호를 나타냈다. 따라서 추천은 아이템 3, 4, 5에 대해서 이루어져야 하므로, 가장 좋은 추천은 아이템 6이 된다. 이와 같이 특정 아이템이 사용자가 선호를 표시한 다른 아이템과 동시에 발생하거나, 특정 아이템의 동시발생이 선호가 큰 아이템과 많이 겹친다면 R의 값은 커지게 된다.

Ⅲ. 동시발생 행렬 분산 알고리즘

3-1. 기존 동시발생 행렬 분산 알고리즘

동시발생 행렬은 아이템의 수가 N개라면, 행렬의 크기는 N^2개가 되어, 아이템의 수가 증가할수록 거대한 행렬이 발생되기 때문에 분산처리 기법을 사용하는 것이 필요하다.

분산처리 기법은 기존 비 분산처리 기법과 달리 여러 단계의

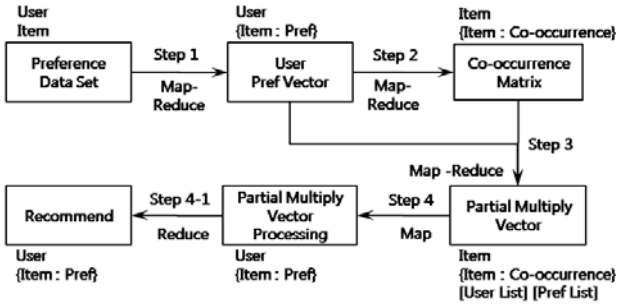


그림 3. 4 단계 동시발생 행렬 분산처리
Fig. 3. Four stage co-occurrence matrix distribution processing.

U1 : I2 I7
U2 : I1 I5 I7
U3 : I1 I2 I6
U4 : I1 I3 I7
U5 : I2
U6 : I5 I7
U7 : I1 I3 I5 I6

U1 {I2 : 1.0, I7 : 1.0}
U2 {I1 : 1.0, I5 : 1.0, I7 : 1.0}
U3 {I1 : 1.0, I2 : 1.0, I6 : 1.0,}
U4 {I1 : 1.0, I3 : 1.0, I7 : 1.0}
U5 {I2 : 1.0}
U6 {I5 : 1.0, I7 : 1.0}
U7 {I1 : 1.0, I3 : 1.0, I5 : 1.0, I6 : 1.0}

그림 4. 선호데이터와 선호벡터
Fig. 4. Preference data and preference vector.

I1 {I7 : 2.0, I6 : 2.0, I5 : 2.0, I3 : 2.0, I2 : 1.0, I1 : 4.0}
I2 {I7 : 1.0, I6 : 1.0, I2 : 3.0, I1 : 1.0}
I3 {I7 : 1.0, I6 : 1.0, I5 : 1.0, I3 : 2.0, I1 : 2.0}
I5 {I7 : 2.0, I6 : 1.0, I5 : 3.0, I3 : 1.0, I1 : 2.0}
I6 {I6 : 2.0, I5 : 1.0, I3 : 1.0, I2 : 1.0, I1 : 2.0}
I7 {I7 : 4.0, I5 : 2.0, I3 : 1.0, I2 : 1.0, I1 : 2.0}

그림 5. 동시발생행렬
Fig. 5. Co-occurrence matrix.

계산과 변환을 수행해야 한다. 그림 3에 추천용 데이터를 생성하기 위해 하둡 맵리듀스로 분산처리 하는 단계에 대해서 나타냈다[12].

단계 1은 맵리듀스 함수로 선호데이터로 사용자 선호벡터를 생성하는 과정이다. 그림 4에 선호데이터와 사용자 선호벡터에 대해서 나타냈다.

선호데이터는 사용자와 관련된 아이템으로 구성된다. 아이템은 한 번의 거래에서 구매된 품목 또는 웹 사이트의 클릭 정보이다. 또한, 사용자 선호벡터의 키-밸류는 사용자와 {아이템 : 선호 값}벡터로 구성된다. 이때 선호 값은 사용자의 선호 값이거나 선호 값이 없는 경우 불린(Boolean)선호 값을 갖는다.

단계 2는 맵리듀스 함수로 사용자 선호벡터로 동시발생행렬을 생성하는 과정이다. 그림 5에 동시발생행렬 생성 결과에 대해서 나타냈다. 동시발생 행렬의 키-밸류는 아이템과 {아이템 : 동시발생 횟수}로 구성된다.

단계 3은 각 사용자별로 아이템의 추천 값을 계산하기 위해 부분 곱 벡터를 생성하는 과정이다. 여기서 각 사용자의 추천 벡터 R을 구하기 위해서, 각 사용자 선호벡터의 I 번째 요소가 0일 경우 계산에서 생략할 수 있다. 따라서 계산에 필요한 동시

	I1	I2	I3	I4	I5	I6	I7		U1		R
I1	4	1	2	0	2	2	2		0.0		3.0
I2	1	3	0	0	0	1	1		1.0		4.0
I3	2	0	2	0	1	1	1		0.0		1.0
I4	0	0	0	0	0	0	0		0.0		0.0
I5	2	0	1	0	3	1	2		0.0		2.0
I6	2	1	1	0	1	2	0	×	0.0	=	1.0
I7	2	1	1	0	2	0	4		1.0		5.0

그림 6. 추천 벡터 생성
Fig. 6. Recommend vector generation.

발생 행렬의 열수는 사용자가 선호한 요소의 개수와 동일하게 된다. 그림 6에 추천 벡터를 생성한 예를 나타냈다.

사용자 1은 아이템 2와 7에 대해서 선호를 가지고 있다. 따라서 2번째 열과 7번째 열만을 계산함으로서 R을 계산 할 수 있다. 즉 사용자 1에 대한 아이템 1의 R 값은 다음과 같이 계산된다.

$$R1 = (1.0 \times 1.0) + (2.0 \times 1.0) = 3.0. \quad (3)$$

위 식에서 결국 내적 대신 합으로 결과 값을 얻을 수 있음을 알 수 있다. 단계 3은 이러한 계산을 할 수 있도록 사용자 선호 벡터와 아이템 동시발생 행렬을 입력으로 하여, 아이템과 {아이템:동시발생개수} [사용자 List][선호 List]를 키-밸류로 출력한다. 그림 7에 부분 곱 벡터 생성 결과에 대해서 나타냈다.

단계 4는 부분 곱 벡터를 이용하여, 부분 곱을 처리하기 위한 전처리 과정이다. 예를 들어, 아이템 1의 사용자 [U2, U4, U7, U3]에 대해 각 사용자별로 동시발생 행렬을 연결하는 과정이다. 결과로서 출력의 키는 사용자 ID이며 밸류는 {아이템 : 동시발생 횟수}벡터로 구성되어 있다. 그림 8에 부분 곱 벡터 처리 결과에 대해서 나타냈다.

부분 곱 벡터 처리는 여러 개의 데이터 노드에서 분산처리된다. 각 사용자 별로 처리된 부분 곱 벡터 처리결과는 단계 4-1에서 리듀서로 전송되어, 사용자 별로 정렬되고 처리되어 추천 파일이 생성된다. 그림 9는 최종결과에 대해서 나타냈다.

```
I1  {I7 : 2.0, I6 : 2.0, I5 : 2.0, I3 : 2.0, I2 : 1.0, I1 : 4.0}
    [U2, U4, U7, U3] [1.0, 1.0, 1.0, 1.0]
I2  {I7 : 1.0, I6 : 1.0, I2 : 3.0, I1 : 1.0}
    [U1, U3, U5] [1.0, 1.0, 1.0]
I3  {I7 : 1.0, I6 : 1.0, I5 : 1.0, I3 : 2.0, I1 : 2.0}
    [U4, U7] [1.0, 1.0]
I5  {I7 : 2.0, I6 : 1.0, I5 : 3.0, I3 : 1.0, I1 : 2.0}
    [U2, U6, U7] [1.0, 1.0, 1.0]
I6  {I6 : 2.0, I5 : 1.0, I3 : 1.0, I2 : 1.0, I1 : 2.0}
    [U3, U7] [1.0, 1.0]
I7  {I7 : 4.0, I5 : 2.0, I3 : 1.0, I2 : 1.0, I1 : 2.0}
    [U1, U2, U4, U6] [1.0, 1.0, 1.0, 1.0]
```

그림 7. 부분 곱 벡터 생성
Fig. 7. Partial multiply vector generation.

```
U1  {I7 : 1.0, I6 : 1.0, I2 : 3.0, I1 : 1.0}
U1  {I7 : 4.0, I5 : 2.0, I3 : 1.0, I2 : 1.0, I1 : 2.0}
U2  {I7 : 2.0, I6 : 2.0, I5 : 2.0, I3 : 2.0, I2 : 1.0, I1 : 4.0}
U2  {I7 : 2.0, I6 : 1.0, I5 : 3.0, I3 : 1.0, I1 : 2.0}
U2  {I7 : 4.0, I5 : 2.0, I3 : 1.0, I2 : 1.0, I1 : 2.0}
U3  {I7 : 2.0, I6 : 2.0, I5 : 2.0, I3 : 2.0, I2 : 1.0, I1 : 4.0}
U3  {I7 : 1.0, I6 : 1.0, I2 : 3.0, I1 : 1.0}
U3  {I6 : 2.0, I5 : 1.0, I3 : 1.0, I2 : 1.0, I1 : 2.0}
```

그림 8. 부분 곱 벡터 처리
Fig. 8. Partial multiply vector processing.

```
U1  {I7:5.0, I2:4.0, I1:3.0, I5:2.0, I3:1.0, I6:1.0}
U2  {I7:8.0, I1:8.0, I5:7.0, I3:4.0, I6:3.0, I2:2.0}
U3  {I1:7.0, I6:5.0, I2:5.0, I7:3.0, I3:3.0, I5:3.0}
U4  {I1:8.0, I7:7.0, I5:5.0, I3:5.0, I6:3.0, I2:2.0}
U5  {I2:3.0, I7:1.0, I6:1.0, I1:1.0}
U6  {I7:6.0, I5:5.0, I1:4.0, I3:2.0, I6:1.0, I2:1.0}
U7  {I1 :10.0, I5:7.0, I3:6.0, I6:6.0, I7:5.0, I2:2.0}
```

그림 9. 최종 결과
Fig. 9. Final result.

3-2. 개선 동시발생 행렬 분산 알고리즘

동시발생 행렬 분산 알고리즘은 단계별로 많은 데이터를 처리하기 때문에, 많은 계산량과 입력되는 선호데이터보다 더 큰 중간 결과 데이터가 발생된다. 따라서 본 논문은 기존의 동시발생 행렬 알고리즘의 처리 단계를 줄일 수 있는 방안을 제시한다. 그림 10에 제시한 동시발생 행렬 분산처리 알고리즘에 대해서 나타냈다.

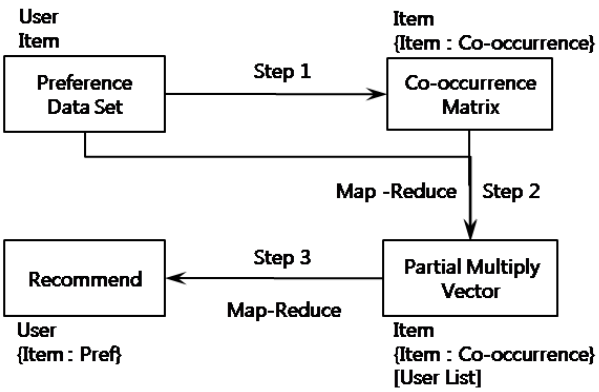


그림 10. 개선 동시발생 행렬 분산처리
Fig. 10. Improved co-occurrence matrix distribution processing.

표 1. 맵리듀스 키-밸류(단계 1)
Table 1. Map reduce key-value(stage 1).

	키	밸류
맵 입출력	오프셋	라인
	아이템 ID	아이템 D
리듀서 입출력	아이템 ID	아이템 ID[Iterable]
	아이템 ID	{아이템 ID : 동시발생 횟수}

```
1    {7:2.0,6:2.0,5:2.0,3:2.0,2:1.0,1:4.0}
2    {7:1.0,6:1.0,2:3.0,1:1.0}
3    {7:1.0,6:1.0,5:1.0,3:2.0,1:2.0}
5    {7:2.0,6:1.0,5:3.0,3:1.0,1:2.0}
6    {6:2.0,5:1.0,3:1.0,2:1.0,1:2.0}
7    {7:4.0,5:2.0,3:1.0,2:1.0,1:2.0}
```

그림 11. 단계 1 결과
Fig. 11. Stage 1 result.

개선 동시발생 행렬 분산처리 처리는 동시발생 행렬생성, 부분 곱 벡터 생성 및 추천 파일 생성 등 3단계로 구성된다. 단계 3은 기존의 동시발생 분산처리에서 단계 4와 단계 4-1과 동일한 처리이다.

단계 1은 선호데이터로 동시발생 행렬을 생성하는 단계이다. 선호데이터는 사용자, 아이템, 선호 등 동시발생 행렬을 생성하기 위해 필요한 모든 데이터가 있기 때문에, 기존 알고리즘의 단계 2를 생략할 수 있다. 단계 1의 맵입력 키 밸류는 오프셋-라인 이며, 리듀서 출력 키 밸류는 아이템 ID -{아이템 ID : 동시발생 횟수}이다.

단계 2는 선호데이터와 동시발생 행렬을 이용하여 부분 곱 벡터를 생성하는 단계이다. 단계 2는 선호데이터와 동시발생 행렬을 입력하기 위해서 2개의 매퍼로 구성된다. 매퍼 1은 아이템 ID 와 동시발생 행렬을 입력 받아, VectorOrPrefWritable형식으로 동시발생 열을 래핑하여 출력한다. 매퍼 2는 선호데이터를 입력하여, 아이템 ID와 {사용자 ID : 선호} 벡터를 출력으로 한다. 리듀서는 두 맵의 출력 결과를 아이템 ID로 정렬하여, 아이템 ID와 {아이템 ID : 동시발생횟수} {사용자 ID : 선호} 리스트를 입력으로 하여, 최종적으로 아이템 ID와 {아이템 ID : 동시발생횟수}[사용자 ID List] [선호 List]를 출력한다. 단계 2의 맵입력 키 밸류는 아이템 ID -{아이템 ID : 동시발생 횟수} 이며, 리듀서 출력 키 밸류는 아이템 ID-{아이템 ID : 동시발생 횟수}[사용자 ID List] [선호 List]이다.

표 2. 맵리듀스 키-밸류(단계 2)
Table 2. Mapreduce key-value(stage 2).

	키	밸류
맵 1 입출력	아이템 ID	{아이템 ID : 동시발생횟수}
	아이템 ID	{아이템 ID : 동시발생횟수}
맵 2 입출력	사용자 ID	{아이템 ID : 선호}
	아이템 ID	{아이템 D : 선호}
리듀서 입출력	아이템 ID	{아이템 ID : 동시발생횟수} {아이템 ID : 선호} [Iterable]
	아이템 ID	{아이템 ID : 동시발생횟수} [사용자 ID List] [선호 List]

단계 2의 실행 결과인 부분 곱 벡터는 추천용 파일 생성을 위한 모든 데이터가 있다. 추천용 파일을 생성하기 위해 부분 곱 벡터에서 사용자 리스트에 있는 모든 사용자별 동시발생 행렬을 분리하면 된다.

단계 3은 부분 곱 벡터로부터 추천용 파일을 생성하는 단계이다. 맵의 입력은 단계 2의 출력을 사용한다. 또한 맵의 출력은 사용자 ID와 부분적인 {아이템 : 선호} 벡터이다. 또한, 리듀스는 입력으로 사용자 ID와 {아이템 : 선호} 리스트이며, 최종적으로 동시발생행렬에서 각 사용자 별 동시발생 횟수를 더하게 된다. 단계 3의 맵입력 키 밸류는 아이템 ID-{아이템 ID : 동시발생횟수}[사용자 ID List] [선호 List]이며, 리듀서 출력 키 밸류는 최종 출력 파일이다.

표 3. 맵리듀스 키-밸류(단계 3)
Table 3. Mapreduce key-value(stage 3).

	키	밸류
맵 입출력	아이템 ID	{아이템 ID : 동시발생횟수} [사용자 List] [선호 List]
	사용자ID	{아이템 선호}
리듀서 입출력	사용자ID	{아이템 선호} [Iterable]
	사용자ID	{아이템 선호}

```
1    [7:5.0,2:4.0,1:3.0,5:2.0,6:1.0,3:1.0]
2    [7:8.0,1:8.0,5:7.0,3:4.0,6:3.0,2:2.0]
3    [1:7.0,6:5.0,2:5.0,7:3.0,3:3.0,5:3.0]
4    [1:8.0,7:7.0,5:5.0,3:5.0,6:3.0,2:2.0]
5    [2:3.0,7:1.0,6:1.0,1:1.0]
6    [7:6.0,5:5.0,1:4.0,3:2.0,6:1.0,2:1.0]
7    [1:10.0,5:7.0,3:6.0,6:6.0,7:5.0,2:2.0]
```

그림 13. 단계 3 실행 결과
Fig. 13. Stage 3 result.

```
1    {7:2.0,6:2.0,5:2.0,3:2.0,2:1.0,1:4.0} [4, 7, 2, 3]      [1.0, 1.0, 1.0, 1.0]
2    {7:1.0,6:1.0,2:3.0,1:1.0}  [5, 1, 3]   [1.0, 1.0, 1.0]
3    {7:1.0,6:1.0,5:1.0,3:2.0,1:2.0}  [4, 7]      [1.0, 1.0]
5    {7:2.0,6:1.0,5:3.0,3:1.0,1:2.0}  [2, 6, 7]   [1.0, 1.0, 1.0]
6    {6:2.0,5:1.0,3:1.0,2:1.0,1:2.0}  [7, 3]      [1.0, 1.0]
7    {7:4.0,5:2.0,3:1.0,2:1.0,1:2.0}  [2, 4, 6, 1]      [1.0, 1.0, 1.0, 1.0]
```

그림 12. 단계 2 실행 결과
Fig. 12. Stage 2 result.

본 연구는 4단계에서 3단계를 축소하였을 때 동일한 결과를 얻었다. 그림 13에 단계 3의 최종 실행결과에 대해서 나타냈다.

Ⅳ. 실험 및 검토

본 논문은 동시발생 행렬 알고리즘을 분산처리하기 위해 pentium(R) dual-core CPU, 클록 스피드 2 GHz, 메인 메모리 2GB의 개인용 컴퓨터, ubuntu 12.04, mahout-distribution-0.7, hadoop-1.2.1, java-1.7.1_51을 사용하였다. 표 4에 본 논문의 실험 환경에 대해서 나타냈다.

본 논문의 선호데이터는 위키피디아에서 문서별로 문서 링크만 추출한 데이터로서, split 명령으로 1.5 M로 나누어 사용하였다. 데이터는 10000개의 문서(사용자)에 대략 10000개 정도의 링크문서(아이템)가 연결된 형태이다. 본 논문은 4 단계로 처리되는 맵리듀스를 하나의 명령으로 처리하기 위해, 하둡 드라이버 클래스에서 여러 개의 잡을 연결하여 연속으로 실행하는 방법을 이용하였다. 4단계 실행방법은 다음과 같다.

hadoop jar JobChain1.jar JobChain /test.dat /one1/ one2 /one3 /one4

각 잡의 결과는 one1, one2, one3 디렉토리에 저장되며, 최종 추천용 파일결과는 one4 디렉토리에 저장된다. 또한 3단계로 처리하는 명령은 다음과 같다.

hadoop jar JobChain2.jar JobChain2 /test.dat /two1 /two2 /two3

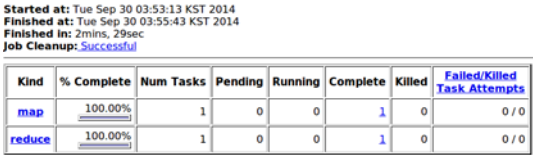
그림 14에 4단계 동시발생 행렬의 분산처리 실행 결과에 대해서 나타냈다.

그림 14 (a)는 선호 데이터로 사용자 선호벡터를 생성한 맵리듀스 결과로서, 1개의 매퍼로 21초가 소요되었다. 그림 14(b)는 사용자 선호벡터로 동시 발생행렬을 생성한 결과로 1개의 매퍼로 2분 29초가 소요되었다. 동시발생 실행결과는 150 MB 이다. 그림 14(c)는 사용자 선호벡터와 동시발생 행렬로 부분 곱 벡터를 생성한 결과로서 4개의 매퍼로 1분 12가 소요되었다. 그림 14 (d)는 최종 추천용 결과 파일을 생성한 결과로서 2개의 매퍼로 16분 51초가 소요되었다. 그림 15에 3단계 실행 결과에 대해서 나타냈다.

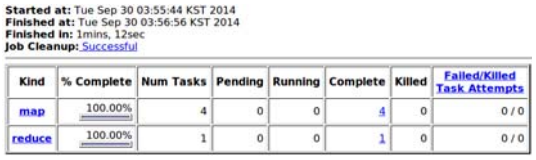
표 4. 실험 환경
Table 4. Experiment environment.

하드웨어	Pentium Dual-Core CPU, 2GHz, 2GB
시스템 플랫폼	ubuntu 12.04 LTS
하둡 버전	hadoop-1.2.1
머하웃 버전	mahout-distribution-0.7
JDK	java-1.7.1_51
개발환경	이클립스

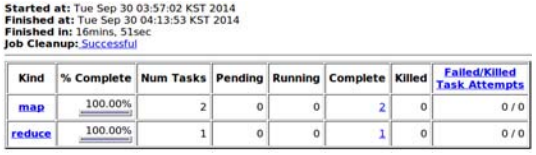
(a) 단계 1 실행결과



(b) 단계 2 실행결과

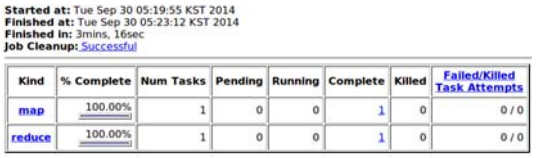


(c) 단계 3 실행결과

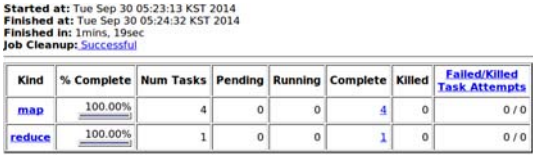


(d) 단계 4 실행결과

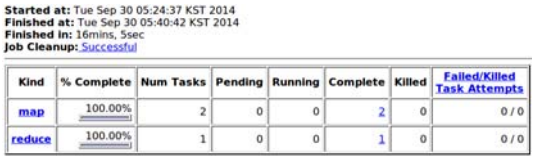
그림 14. 4 단계 분산처리 결과
Fig. 14. Result of four stage distribution processing.



(a) 단계 1 실행결과



(b) 단계 2 실행결과



(c) 단계 3 실행결과

그림 15. 3 단계 분산처리 결과
Fig. 15. Result of three stage distribution processing.

표 5. 단계별 소요시간
Table 5. Stage use time.

단계	기존 분산처리	제시 분산처리
1단계	21초	196초
2단계	149초	79초
3단계	72초	965초
4단계	1011초	
	1,253초	1,240초

표 6. 단계별 결과 데이터
Table 6. Stage result data.

단계	기존 분산처리	제시 분산처리
선호데이터	1,553,810	1,553,810
1단계	2,397,285	157,723,477
2단계	157,375,268	105,953,801
3단계	105,606,172	951,077
4단계	951,051	
	266,329,776	266,182,165

그림 15(a)는 선호데이터로 동시발생 행렬을 생성한 결과로 1개의 매퍼로 3분 16초가 소요되었다. 동시발생 실행결과는 4단계 기법과 마찬가지로 150 MB에 달하였다. 그림 14(b)는 선호 데이터와 동시발생 행렬 결과 데이터로 부분 곱 벡터를 생성한 결과로서 4개의 매퍼로 1분 19초가 소요되었다. 그림 14(c)는 최종 추천용 결과 파일을 생성한 결과로서 2개의 매퍼로 16분 5초가 소요되었다.

기존의 분산처리는 선호데이터를 이용하여 사용자 선호벡터와 동시발생행렬을 생성하고, 선호데이터와 동시발생행렬로 부분 곱 벡터와 최종 추천 선호벡터를 생성하였다. 제안된 분산처리는 두 번째 단계인 사용자 선호벡터 단계를 없애고 선호데이터로 동시발생 행렬을 직접 생성하고, 선호데이터와 동시발생행렬을 이용하여, 부분곱 벡터와 최종 추천 선호벡터를 생성함으로써, 분산 처리 단계를 4단계에서 3단계로 축소하였다. 표 5에 단계별 소요시간을 나타냈으며, 표 6에 단계별 결과 데이터에 대해서 나타냈다. 결과로서, 맵리듀스 잡을 줄일 수 있었으며, 이로 인해 분산 처리속도가 1,253초에서 1,240초로 단축되었으며, 맵 출력 데이터가 266,329,776에서 266,182,165로 감소되었다. 이와 같이 분산처리 단계를 축소함으로서 맵리듀스 잡의 실행을 축소하고 처리속도와 맵 출력 데이터가 감소됨을 확인하였다. 그러나 동시발생 행렬을 생성하고 부분 곱을 계산하는 시간으로 인해 기대만큼 효과를 보지 못한 한계가 있었다.

Ⅴ. 결 론

본 논문은 하둡과 머하웃을 이용하여, 선호데이터를 분산처리하여, 어플리케이션에서 사용할 수 있는 아이템 추천용 선호벡터를 도출하는 방법을 제시하였다. 본 논문의 추천 알고리즘인 동시발생 행렬 알고리즘은 계산의 많은 부분이 관련 데이터를 효율적으로 수집해서 한 곳으로 모으는 작업이 대부분이기 때문에, 레코드 단위로 계산하고 정렬이 용이한 하둡 분산처리 기법이 적합하였다. 본 논문의 동시발생 행렬 알고리즘은 다양한 도메인의 추천 서비스에 응용될 수 있다. 빅데이터 처리 플랫폼인 하둡은 대형 선호데이터의 실시간 추천시스템에 다양하게 응용될 수 있을 것이다. 선호 데이터는 끊임없이 커질 것이며, 추천시스템의 실시간 처리를 위해 분산처리 알고리즘에 대해서 지속적인 연구가 진행되어야 할 것이다.

참고문헌

[1] Y. S. Kim, “Research trend recommend service for personalization service,” *Korean Institute of industrial Engineers, ie magazine*, Vol. 19, No 1, pp. 37-42, May 2012.

[2] R. V. Meteren, and M. V. Someren, “Using content-based filtering for recommendation,” in *Proceedings of ECML 2000 Workshop: Machine Learning in New Information Age*, Barcelona: Spain, pp. 47-56, Mar. 2000.

[3] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th International Conference on World Wide Web. ACM*, Hong Kong, pp. 285-295, May 2001.

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

[5] J. H. Jung, *Beginning Hadoop Programming*, 2nd ed, Wikibooks, 2013.

[6] K. Shvachko, et al, “The hadoop distributed file system,” in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on IEEE*, Incline Village: NV, pp. 1-10, May 2010.

[7] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, Vol 42, No 8, pp. 30-37, Aug. 2009.

[8] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, “Large-scale matrix factorization with distributed stochastic gradient descent,” in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Diego: CA, pp. 69-77, Aug. 2011.

[9] S. Schelter, C. Boden, and V. Markl, “Scalable similarity-based neighborhood methods with mapreduce,” in *Proceedings of the Sixth ACM Conference on Recommender Systems*, Dublin: Ireland, pp. 163-170, Sep.

2012.

[10] Z. D. Zhao, and M. S. Shang, “User-based collaborative-filtering recommendation algorithms on hadoop,” in *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on IEEE*, Phuket: Thailand, pp. 478-481, Jan. 2010.

[11] F. Provost, T. Fawcett, *Data Science for Business*, California, CA: O'Reilly Media, 2013.

[12] S. Owen, R. Anil, T. Dunning and E. Friedman, *Mahout in Action*, New York, NY: Manning Publications, 2011.

김 창 복 (Chang-Bok Kim)

1986년 2월 : 단국대학교 전자공학과(공학사)
1989년 2월 : 단국대학교 전자공학과(공학석사)
2008년 2월 : 인천대학교 컴퓨터 공학과(공학박사)
1994년 ~ 현재 : 가천대학교 IT대학 에너지 IT학과 교수
관심분야 : 인터넷보안, 클라우드 컴퓨팅, 분산처리시스템

정 재 필 (Jae-Pil Chung)

1985년 2월 : 단국대학교 전자공학과 (공학사), 1989년 8월 : 단국대학교 대학원 전자공학과 (공학석사)
2000년 8월 : 한국항공대학교 대학원 통신정보공학과 (공학박사)
1989년 8월~1990년 12월 : (주)동양전자통신 중앙연구소 연구원
1990년 12월~1992년 3월 : (주)케피코 기술연구소 연구원
1994년 2월~현재 : 가천대학교 IT대학 전자공학과 교수
[주 관심분야] 무선통신, 통신신호처리, 컴퓨터 네트워크