그래프 사전 분할 기법을 활용한 Butterfly Counting MapReduce 알고리즘

허태정○, 이용준, 박하명†
국민대학교 소프트웨어학부
gjdigj145@kookmin.ac.kr, joon0351@kookmin.ac.kr, hmpark@kookmin.ac.kr

A MapReduce algorithm for Counting Butterflies based on Graph Pre-Partitioning

Taejung Heo○, Yong-Joon Lee, Ha-Myung Park†
College of Computer Science, Kookmin University

요 약

이분 그래프에서 네 정점이 서로 연결되어 있는 부분 그래프를 butterfly라고 한다. butterfly는 이분 그래프를 분석하기 위한 기본적인 구조이기 때문에 이분 그래프에서 butterfly 개수를 찾는 연구가 활발하다. 또한, 대용량의 그래프를 처리하기 위한 MapReduce 알고리즘의 필요성도 커지고 있다. 현재까지 개발된 MapReduce 알고리즘들은 butterfly 개수를 찾을 때 발생하고 Shuffle 되는 중간데이터의 크기가 매우 크거나 메모리 공간을 많이 차지하여 대용량 데이터를 처리하지 못한다. 본 연구에서는 그래프 사전 분할 기법으로 중간데이터를 줄이면서 메모리를 효율적으로 사용하는 알고리즘을 제안하고, 비교실험을 통해 기존 방법이 처리하지 못하는 데이터를 빠르게 처리할 수 있음을 보인다.

1. 서 론

이분 그래프는 저자와 출판물, 영화와 평가 그리고 소비자와 상품 같은 두 종류의 객체 간의 관계를 표현할 수 있는 그래프이다. 이분 그래프에서 네 정점이 서로 연결되어 있는 부분 그래프를 butterfly라 한다. 소비자와 상품의 관계를 나타낸 그래프에서 예를 들면, 두 명의 소비자가 두 상품을 동일하게 구매했다면 butterfly를 이룬다. 이분 그래프는 다양한 분야에서 활용되고 있는 관계 모델이고, butterfly는 이분 그래프를 분석하기 위한 기본적인 구조이기 때문에 이분 그래프에서 butterfly의 개수를 찾는 연구가 활발하다.

그래프의 크기가 커짐에 따라 대용량의 그래프를 처리할 수 있는 MapReduce 알고리즘의 필요성도 함께 증가하고 있다. 그러나 현존하는 butterfly 개수를 찾는 MapReduce 알고리즘들[1,2]은 많은 양의 중간데이터를 발생시키거나 많은 메모리를 차지하여 대용량 그래프 처리에 적합하지 않다.

본 연구에서는 butterfly 개수를 계산하는 기존 MapReduce 알고리즘들의 문제점인 중간데이터 폭증 문제와 메모리 과사용 문제를 개선한 BFCPart 알고리즘을 제안한다. 그리고 BFCPart와 BFCSimple[1], BFCWedgeJoin[1,2] 세 MapReduce 알고리즘의 수행시간과 발생하는 중간데이터의 크기를 비교하는 실험을 진행하고 결과를 제시한다.

2. 문제정의

정점들의 서로소 집합 L, R로 이루어진 정점집합 V와 간선집합 E로 이루어진 무 방향, 비가중치 이분 그래프 $G = (V, E)$가 주어질 때, butterfly와 butterfly 계산 문제를 다음과 같이 정의한다.

정의1. 이분 그래프 $G = (V = (L, R), E)$가 주어졌을 때, 정점 $u, u' \in L, v, v' \in R$ 에 대해서 $(u, v), (u, v'), (u', v), (u', v') \in E$ 인 경우, (u, v, u', v')은 butterfly이다.

정의2. Butterfly 계산 문제는 이분 그래프 $G = (V = (L, R), E)$가 주어졌을 때, butterfly 집합 $B = \{(u, v, u', v') | u, u' \in L, v, v' \in R\}$의 크기 $|B|$를 계산하는 것이다.

본 논문에서 빈번히 사용하는 기호를 다음과 같이 정의한다: $N(u)$는 정점 u의 이웃의 집합을, $\deg(u)$는 정점 u의 이웃의 수를 나타낸다. wedge는 3개의 정점 u, u', v와 두 개의 간선 (u, v) , (u', v) 으로 이뤄지는 구조로 (u, v, u') 로 나타낸다. 그래프의 한 정점 u를 포함한 wedge의 최대 개수는 정점의 이웃들 중 2개를 뽑는 조합의 개수와 같고 $\binom{\deg(v)}{2}$ 로 나타내고, 전체 그래프의 wedge 최대 개수는 $O(\sum_{u \in V}\binom{\deg(v)}{2})$ 으로 나타낸다. $|L|$, $|R|$, $|E|$, $|W|$, $|B|$ 는 차례대로 왼쪽 정점의 개수, 오른쪽 정점의 개수, 간선의 개수, wedge의 개수, butterfly의 개수를 나타낸다. $count(number)$는 전체 butterfly 개수를 입력한 숫자만큼 증가시키는 함수이다.

3. 관련연구

3.1 Butterfly counting

[1]의 연구에서는 wedge (u, v, u')을 $u \in L$, $v \in N(u)$, $u' \in N(v)$ 조건을 이용해 찾아내고, (u, u')을 공통으로 가지고 있는 wedge들 중 두 개를 뽑는 조합을 통해 정확한 butterfly 개수를 구한다. [3]의 연구에서는 [1]의 연구를 개선하기 위해

† 교신저자

이분 그래프의 두 정점집합 L, R의 원소들의 차수의 제곱의 합이 더 큰 쪽을 기준으로 잡고, butterfly (u, v, u', v')의 두 정점이 순서관계$u > u'$을 만족하게 하여 중복된 계산을 줄였다. [4]의 연구는 wedge 나열 기반의 정확한 butterfly 개수를 구하는 알고리즘과 sampling을 통한 butterfly 개수를 근사하는 알고리즘 그리고 외부 메모리 알고리즘을 제안한다. [5]의 연구에서는 vertex의 degree를 기준으로 한 우선순위를 이용해 정렬을 한 후 계산을 하는 알고리즘과 이 알고리즘을 병렬화 하는 방법, wedge 나열 기반 외부 메모리 알고리즘을 제안한다. [6]의 연구에서는 멀티 프로세서 환경에서 병렬로 처리할 수 있는 wedge 나열 방식의 알고리즘을 제안한다.

본 연구와 가장 관련이 있는 [1,2]의 연구에서는 정확한 butterfly개수를 구하는 wedge-join 방식의 MapReduce 알고리즘을 제안한다. 두 연구에서는 butterfly를 찾기 위해 wedge들을 나열하고, wedge (u, v, u')의 끝점 u, u'을 기준으로 중심점 v 을 집계하는 방식을 사용한다. Wedge-join을 기반으로 한 알고리즘은 [4,5,6]의 연구에서도 사용됐지만 총 wedge 개수 $O(\sum_{u \in V}\binom{\deg(v)}{2})$만큼의 중간데이터를 발생시키기 때문에 MapReduce 프레임워크에서는 비효율적이다. [1]의 연구에서는 작업 분할 기법을 활용한 MPI 알고리즘을 제안한다. 그래프를 클러스터의 각 연산장치에 복사한 후 분할된 작업을 처리하는 방식으로 중간데이터가 발생하지 않지만 그래프 전체를 메모리에 올려야 하기 때문에 메인 메모리보다 큰 그래프를 처리할 수 없다는 한계가 존재한다.

3.2 Map Reduce

MapReduce는 병렬 및 분산 컴퓨팅을 지원하는 프로그래밍 모델로 빅데이터 처리를 위해서 고안됐다. MapReduce는 세 단계로 구성되며 입력과 출력으로 key, value 쌍을 사용한다. Map 단계에서는 입력 데이터들을 원하는 포맷의 쌍으로 변환하고, Shuffle 단계에서는 모든 쌍들을 key를 기준으로 묶어서 같은 key를 가지는 value끼리 모은다. 마지막으로 Reduce 단계에서는 key를 기준으로 분리된 value들에 대하여 정해진 처리를 한 후 key, value 쌍을 만들어 출력한다.

Map단계에서 버퍼에 저장하지 못한 데이터들은 로컬 디스크에 저장된다. Reduce 단계에서는 네트워크를 통해 이 데이터들을 읽으면서 Shuffle이 발생하게 된다. 이때 많은 양의 I/O와 네트워크 트래픽이 발생하여 성능에 영향을 주므로, 성능 향상을 위해서는 Shuffle 데이터의 크기를 최대한 줄여야 한다.

4. 제안 방법: BFCPart

BFCPart 알고리즘은 그래프 사전 분할 기법을 적용하여 기존 방법의 단점인 중간데이터 폭증 문제와 메모리 과점유 문제를 동시에 해결한다. BFCPart는 Graph partitioning과 Butterfly counting 두 개의 MapReduce 작업으로 구성된다. Algorithm 1은 BFCPart의 의사코드이다.

Graph partitioning 단계에서는 $u \in L$ 에 해시함수 $\xi: L \rightarrow \{0, \dots, \rho-1\}$ 을 적용한 값을 기준으로 간선집합 E 를 ρ 개의 파티션으로 분할한다. $L_\theta = \{u \in L | \xi(u) = \theta\}$라 할 때, 파티션

Algorithm 1: BFCPart

```
   Graph partitioning
1  Map      input: ⟨ϕ, (u, v) ∈ E⟩
2  emit ⟨h(u), (u, v)⟩
3  Reduce   input: ⟨θ, E_θ⟩
4  Build graph G_θ = ((L_θ, R_θ), E_θ) from E_θ
5  emit G_θ to a distributed storage

   Butterfly Counting
6  Map      input: ⟨ϕ, problem = (i, j)⟩
7  Load graph G_i = ((L_i, R_i), E_i) from distributed storage.
8  if i == j then
9     countButterflies (L_i, R_i, L_i)
10 else
11    Load graph G_j = ((L_j, R_j), E_j) from distributed storage.
12 countButterflies (L_i, R_j, L_j)

   Function countButterflies (L, R, L`)
13    for u ∈ L do
14       C ← hashmap
15       for v ∈ N(u) and v ∈ R do
16          for w ∈ N(v) and w ∈ L`:  w ≺ v do
17             C[w] ← C[w] + 1
18       for w ∈ C:  C[w] > 0 do
19             count (C[w] choose 2)
```

id $\theta \in \{0, \dots, \rho-1\}$에 대해서, θ 번째 파티션을 $E_\theta = \{(u, v) \in E | u \in L_\theta\}$로 정의한다. 간선 집합 E_θ로 유도된 그래프는 $G_\theta = ((L_\theta, R_\theta), E_\theta)$ 로 나타낸다. 즉, 오른쪽 정점집합은 $R_\theta = \{v | (u, v) \in E_\theta\}$이다.

Butterfly counting 단계에서는 전체 문제를 ρ^2개의 소문제로 나누고 여러 연산장치에서 나누어 처리한다. $i, j \in \{0, \dots, \rho-1\}$에 대해서, 각 소문제는 분산 저장소에서 두 그래프 파티션 G_i 와 G_j 를 읽어 메모리에 저장하고 $u \in L_i, u' \in L_j$를 만족하는 모든 butterfly의 수를 Algorithm 1의 countButterflies() 함수를 이용해 계산한다. 이 함수는 $u \in L_i$의 이웃인 $v \in N(u)$가 R_j에 속하는 정점이어서 파티션 j에 속하는 왼쪽 정점 $u' \in L_j$로 가는 간선이 존재할 때 만들어지는 wedge를 이용한다.

BFCPart는 ρ^2 개의 소문제에서 예상 크기가 $O(|E|/\rho)$ 인 그래프 파티션 2개를 I/O를 통해 읽으므로, 예상 I/O 복잡도는 $O(2 \times (|E|/\rho) \times \rho^2) = O(|E| \times \rho)$이다. 각 소문제에서 두 그래프 파티션을 메모리에 정점집합을 포함하여 보관하는데, $E(|L_i|) = E(|L_j|) = |L|/\rho$ 이고, $E(|R_i|) = |R|$ 이므로 예상 Memory 복잡도는 $M = O(|E|/\rho + |L|/\rho + |R|)$이다.

5. 실험 및 결과

실험에서는 제안 방법 BFCPart의 우수성을 보이기 위해 알고리즘 수행시간 및 중간데이터 크기를 기준으로 다음 두 방법과 비교한다: (1) [1]의 논문에서 제안한 MPI알고리즘을 MapReduce로 구현한 BFCSimple(Algorithm 2), (2) [1,2] 논문에서 사용한 wedge-join 방식의 MapReduce 알고리즘을 구현한 BFCWedgeJoin(Algorithm 3). 모든 알고리즘은 실행하기 전에 정점집합 L, R 의 차수의 제곱의 총합을 계산하여[3] 기준 정점집합을 구한 후 실험했다.

5.1 데이터셋

표 1. 데이터셋 명세

이분그래프	$\|L\|$	$\|R\|$	$\|E\|$	$\|W\|$	⧖
Wiki-fr	757K	8.87M	52.9K	6.75T	4.49T
LiveJournal	3.2M	7.48M	112M	2.70T	3.29T
Deli-ui	833K	33.7M	101M	69.2T	56.8G
Wiki-en	8.1M	42.6M	255M	6.29T	21.3T
Orkut	2.7M	8.7M	327M	2.52T	22.1T
Web trackers	27.6M	12.7M	140M	106T	20T

Algorithm 2: BFCSimple

```
1  Map      input: θ (partition id)
2  Load graph G = (V = (L,R),E) distributed storage.
3  if ∑_{u∈L}(deg(u))² < ∑_{v∈R}(deg(v))² then
4     countButterflies (L_θ, R, L)
5  else
6     countButterflies (R_θ, L, R)
```

Algorithm 3: BFCWedgeJoin

```
   Wedge Listing
1  Map      input: ⟨ϕ,(u,v) ∈ E⟩
2  emit ⟨u,v⟩
3  Reduce   input: ⟨u,N(u)⟩
4  for v ∈ N(u) do
5     for v′∈ N(u) do
6        if v < v′ then
7           emit ⟨(v,v′),ϕ⟩
7        else then
9           emit ⟨(v′,v),ϕ⟩

   Butterfly Counting
10 Map      input: ⟨(u,v),ϕ⟩
11 emit ⟨(u,v),1⟩
12 Reduce   input: ⟨(u,v), counts[c1,c2,c3 ...]⟩
13 sum ← 0
14 for c ∈ counts do
15    sum = sum + c
```

실험에서는 공개된 실세계 이분그래프를 사용했고, 중복된 간선은 미리 제거했다. 표1의 K는 10^3 배, M은 10^6 배, G는 10^9 배, T는 10^{12} 배를 나타낸다.

5.2 실험환경

모든 실험들은 총 10개의 노드를 가지고 있는 Hadoop cluster에서 이뤄졌다. 각 노드들은 Intel® Xeon® CPU E31220 @ 3.10GHZ 와 RAM 16 GB, 그리고 Ubuntu 18.04 64비트 운영체제를 사용했다. BFCPart Butterfly Counting 단계와 BFCSimple의 Map에는 4GB, BFCPart의 Graph Partitioning 단계의 Map에는 2GB, Reduce에는 4GB의 메모리를 할당했다. BFCWedgeJoin의 경우에는 Map, Reduce 모두 1GB를 할당했으며, 가장 빠른 성능을 보인 설정이다.

5.3 수행 시간 비교

BFCWedgeJoin은 데이터가 커질수록 수행시간이 매우 큰 폭으로 증가했고, Web trackers 데이터는 13시간 이상이 지나도 처리하지 못했다. 이는 BFCWedgeJoin 방식이 대량의 중간데이터를 생성하기 때문이며, 5.4장에서 자세히 살펴본다. BFCSimple은 원본 그래프 크기가 큰 Wiki-en과 Orkut 데이터를 처리할 때 out-of-memory 에러가 발생하여 실패했다. 이는, BFCSimple이 각 연산장치마다 그래프 전체를 메모리에 로드하기 때문이다. BFCPart가 Wiki-fr, LiveJournal를 처리할 때 BFCSimple 보다 느린 이유는 Algorithm1의 Graph Partitioning을 할 때 소요되는 시간이

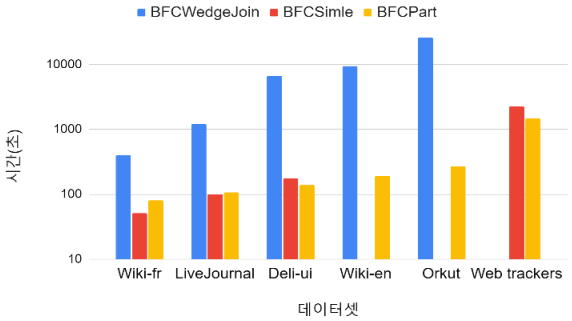

그림 1. 수행시간 비교 그래프

표 2. 원본, Wedge, Partition 파일 크기 및 배수

이분그래프	원본	Wedge	Partition	배수
Wiki-fr	0.6G	20.6G	0.52G	약 40배
LiveJournal	1.5G	75.6G	1G	약 76배
Deli-ui	1.4G	421G	1.2G	약 351배
Wiki-en	3.3G	433G	2.8G	약 155배
Orkut	4.7G	1.25T	2.7G	약 463배
Web trackers	1.8G	>13H	1.7G	–

Butterfly Counting이 단축된 시간보다 크기 때문이다. BFCPart는 유일하게 실험에 사용한 모든 그래프를 처리했으며 대용량 데이터에서 가장 빠른 성능을 보였다.

5.4 중간데이터 크기 비교

BFCWedgeJoin 방식은 Shuffle 단계에서 대량의 중간데이터를 생성하며, 이를 Wedge 파일로 저장한다. 반면, BFCPart는 그래프를 분할한 Partition 파일을 중간데이터로 저장한다. 표2를 확인하면 Partition 파일의 크기가 Wedge 파일 크기보다 최대 463배 작은 결과를 보여 사전 분할 기법이 효과적임을 확인했다.

6. 결론

본 연구에서는 사전 분할 기법을 이용하여 이분 그래프에서 butterfly 개수를 효율적으로 계산하는 확장성 있는 MapReduce 알고리즘인 BFCPart를 제안했다. BFCPart는 기존 MapReduce 알고리즘의 중간데이터 폭증 문제와 메모리 과사용 문제를 해결하여 기존 방법이 처리하지 못한 2조 개가 넘는 butterfly를 포함하는 그래프를 처리하는데 성공했다.

참고문헌

[1] Wang J, Fu AW, Cheng J. Rectangle counting in large bipartite graphs. Big Data, 17—24, 2014.

[2] Cohen J. Graph twiddling in a mapreduce world. Computing in Science & Engineering, 11, 4, 29—41, 2009.

[3] Sanei-Mehri S, Sariyuce AE, Tirthapura S. Butterfly counting in bipartite networks. SIGKDD, 2150—2159, 2018.

[4] Zhu R, Zou Z, Li J. Fast rectangle counting on massive networks. ICDM, 847—856, 2018.

[5] Wang K, Lin X, Qin L, Zhang W, Zhang Y. Vertex priority-based butterfly counting for large-scale bipartite networks. VLDB, 12, 10, 1139—1152, 2019.

[6] Shi J, Shun J. Parallel algorithms for butterfly computations. APOCS, 16—30, 2020.